

QBUS2310 Decision Analytics

Convex Optimisation for Business Analytics

Course Notes

Dr. Li Chen

The University of Sydney Business School

Semester 1, 2027

About these notes

These notes accompany the thirteen lectures of QBUS2310 *Decision Analytics*. They follow the lectures week by week, in the same order, with the same notation and the same running examples, but they are written to be read on their own: each section develops the week's ideas in connected prose, states the important results precisely, works the examples through step by step, and explains how each model is solved and how its solution should be read. A student who has missed a lecture should be able to study the corresponding section and recover the essential concepts, models, methods and examples of that week.

What the unit is about

A great many decisions in business analytics—how to split a marketing budget, which assets to hold, which coefficients to put on which features—can be written as the problem of choosing a vector x to make an objective $f(x)$ as small (or as large) as possible while respecting a set of constraints. The unit studies the class of such problems that can be solved reliably and at scale: *convex* optimisation problems. The first five weeks build the language. [Week 1](#) fixes the vocabulary of optimisation and reviews the linear algebra and calculus we need; [Weeks 2 and 3](#) develop the geometry of convex sets and convex functions; [Week 4](#) turns that geometry into a modelling discipline, with models solved in Python through RSOME; and [Week 5](#) explains how we can *certify* that a solution is optimal, through optimality conditions and duality. The remaining eight weeks are about algorithms. [Week 6](#) builds gradient descent for a smooth objective without constraints, and each later week relaxes one assumption of that setting or removes one bottleneck: the objective may have kinks, which the subgradient method handles ([Week 7](#)); the decision may be restricted to a simple set, which projected gradient handles ([Week 8](#)); the objective may carry a nonsmooth penalty with a cheap proximal operator, which proximal gradient handles ([Week 9](#)); gradient descent may crawl on an ill-conditioned problem, and momentum, Nesterov's method and FISTA accelerate it ([Week 10](#)); one gradient may cost a full pass over too large a data set, and stochastic gradients sample it ([Week 11](#)); the problem may have too many variables, or be split among decision makers tied together by a shared constraint, and coordinate descent, dual decomposition and ADMM solve smaller subproblems ([Week 12](#)); and the feasible set may be too complicated to project onto, which leads to the Frank–Wolfe, splitting and interior-point methods of [Week 13](#), whose second half draws the whole unit together in one map. The algorithm weeks therefore fan out from [Week 6](#) rather than form a chain: [Week 10](#) accelerates the methods of [Weeks 6, 8 and 9](#), [Week 11](#) reuses the analysis of [Week 7](#), and [Week 13](#) starts by removing the premise of [Week 8](#) that projection is cheap. Every algorithm week answers the questions that [Week 6](#) poses for gradient descent—why iterate, in which direction to move, how far to step, how fast the method converges and when to stop—and ends with an honest verdict on when the method should and should not be used. [Week 13](#) condenses these questions into one: *what is cheap to compute here, and what does it certify?*

How to read them

Definitions, theorems, propositions, examples, models and remarks are numbered consecutively within each week, so that Definition 3.2 is the second numbered item of Week 3. Optimisation models that recur through the unit are set out in *Model* environments, followed by a discussion of their decision variables, objective, constraints, assumptions, solution method and interpretation. Algorithms are given in pseudocode. Shaded *Key idea* boxes record the one or two sentences worth carrying away from a subsection. Proofs are included where the argument teaches the technique of the week; a proof marked as a sketch gives the main steps only. Each week ends with a summary that looks ahead to the next week, and with exercises drawn from the week's

tutorial.

The notes treat the lecture slides as the primary source. Where a slide appears to contain an error, an ambiguity, or an inconsistency with another week, the notes give the statement they believe to be correct and record the discrepancy in a box headed *Note on the lecture slides*, so that it can be checked rather than silently propagated. Every departure from the slides, boxed or not, is also listed week by week in the appendix *Discrepancies between the notes and the lecture slides* at the end of the notes.

Software

Models are implemented in Python with RSOME as the modelling layer and a solver underneath: RSOME's default SciPy-based solver for first experiments, and the commercial solvers Gurobi or MOSEK for linear, quadratic and second-order-cone models, with MOSEK for exponential-cone models (Sections 1.6 and 4.5); the algorithm weeks also use plain NumPy, because the point of those weeks is to see the iterations themselves. Figures described as *measured* were produced by the companion Jupyter notebooks of Weeks 6–13 from real runs of the algorithms.

Notation

The notes use one convention per concept throughout, the same as the lecture slides. The table lists the symbols that keep their meaning across weeks, with the week in which each is introduced. A few letters are deliberately reused for different objects, sometimes in neighbouring weeks; these are listed at the end, and each is pointed out in the text where it first matters.

Symbol	Meaning	Week
<i>Sets, vectors and matrices</i>		
$\mathbb{R}, \mathbb{R}_+, \mathbb{R}_{++}$	real numbers; nonnegative reals; positive reals	1
$x \in \mathbb{R}^d$	decision variable (a column vector); d is its dimension	1
N	number of data points (records) in a data-fitting problem	1
$a^\top x, \quad Ax \leq b$	inner product; m linear inequalities read row by row	1
$\mathbf{1}$	the vector of all ones	1
$\ x\ _2, \ x\ _1, \ x\ _\infty$	Euclidean, ℓ_1 and ℓ_∞ norms; $\ x\ $ without subscript is Euclidean	1
$\sup S, \inf S$	least upper and greatest lower bound of a set $S \subseteq \mathbb{R}$	1
$Q \succeq 0, \quad Q \succ 0$	Q symmetric positive semidefinite; positive definite	3
<i>Optimisation models</i>		
f (or f_0)	objective function	1, 4
$g_i, i = 1, \dots, m$ (or f_i)	inequality-constraint functions, $g_i(x) \leq 0$	1, 4
$h_j, j = 1, \dots, p$	equality-constraint functions, $h_j(x) = 0$; affine, and written $a_j^\top x = b_j$ or $Ax = b$, from Week 4	1, 4
$\mathcal{X}; \quad C$	feasible set ($\mathcal{X}$ in Weeks 1–6 and 11; C in Weeks 8–10 and 13)	1, 8
$x^*; \quad z^*, p^*, f^*$	an optimal solution; the optimal value	1, 5, 6
$\operatorname{argmin}, \operatorname{argmax}$	the set of minimisers (maximisers)	6
s.t.	subject to	1
<i>Convex analysis</i>		
$\operatorname{conv} S$	convex hull of the set S	2
$\operatorname{dom} f, \operatorname{epi} f$	domain and epigraph of f	3
$\nabla f(x), \nabla^2 f(x)$	gradient and Hessian of f at x	1, 3
$\partial f(x); \quad g \in \partial f(x)$	subdifferential of f at x ; a subgradient	3, 5, 7
I_C	indicator function of the set C (0 on C , $+\infty$ outside)	8
$N_C(x)$	normal cone of C at x	8
$\Pi_C(x)$	Euclidean projection of x onto C	8
$\operatorname{prox}_{\alpha h}(x)$	proximal operator of h with parameter $\alpha > 0$	9
$\operatorname{soft}_\tau(v)$	soft-thresholding of v at level τ (coordinatewise for vectors)	9
<i>Duality</i>		
$L(x, \lambda, \nu)$	Lagrangian	5
$\lambda \geq 0; \quad \nu$	multipliers on inequality constraints (prices); on equality constraints	5
$g(\lambda, \nu); \quad q(\lambda)$	dual function (Week 5 name; Week 12 name)	5, 12
$p^*; \quad d^*$	optimal values of the primal and of the dual problem	5
<i>Algorithms</i>		
$x^k, \quad x^0$	the k -th iterate (a superscript, not a power); the starting point	6
d^k	search direction at iteration k	6
$t_k; \quad \alpha_k$	step size: written t_k in Week 6 and α_k from Week 7 onwards	6, 7

Symbol	Meaning	Week
$L; \mu$	smoothness constant (curvature cap); strong-convexity constant (curvature floor)	6
$\kappa = L/\mu$	condition number	10
G	bound on subgradient norms	7
$G_\alpha(x)$	projected-gradient (Week 8) and proximal-gradient (Week 9) mapping; the stopping residual	8, 9
$F = g + h$	composite objective: smooth part g plus a simple (proximable) part h (Week 11 writes the smooth finite sum as F and the composite as $F + h$)	9, 11
$\Delta_k = \ x^k - x^\star\ _2^2$	squared distance to the solution; $A_k = \mathbb{E} \Delta_k$ in Week 11	7, 11
$\bar{x}^k; \bar{x}^T$	averaged iterate; averaged iterate over a horizon of T steps	7, 11
$\beta; \theta_k$	momentum coefficient; Nesterov/FISTA schedule scalar	10
$\mathbb{E}, \text{Var}; s$	expectation, variance; mini-batch size	11
$J; R; \rho$	number of blocks; shared resource; ADMM penalty parameter	12
$r^k; s^k; u = \lambda/\rho$	ADMM primal residual; dual residual; scaled dual variable	12
$s^k; \gamma_k; g_k$	Frank–Wolfe vertex; Frank–Wolfe step; Frank–Wolfe gap	13
t	barrier parameter of the log-barrier method	13
$O(\cdot)$	order of growth: $a_k = O(1/k)$ means $a_k \leq C/k$ for some constant C	6

Letters with more than one job

The decision dimension is the plain letter d , whereas a search direction is the superscripted vector d^k (in Week 11, d^k is the momentum or Adam running direction, stepped along as $-\alpha d^{k+1}$); $d^\star$ is the optimal value of the dual problem in Week 5 but the unit steepest-descent direction in Week 6. The letter D is a domain restriction in Week 1, a scaling matrix in Week 6 and the diameter of C in Week 13. The letter t is an auxiliary (lifting or epigraph) variable in Weeks 2–4 and 7, the step size t_k in Week 6, and the barrier parameter in Week 13; from Week 7 on the step size is α_k , whereas in Week 6 α is the sufficient-decrease fraction of backtracking.

The Greek letter λ has three jobs. It is a Lagrange multiplier in Weeks 5, 8, 12 and 13 (in Week 12 it also multiplies an equality constraint, which Week 5 wrote ν); it is a regularisation weight in Weeks 3 and 4, in the lasso previewed in Week 5 and studied in Weeks 9, 10 and 12, and in the ridge and weight-decay terms of Week 11; and it is an eigenvalue, but only inside the quadratic analysis of Week 10. The letter β is a vector of regression coefficients in the data-fitting examples, a slab bound in Week 2, a price sensitivity in Week 4, the backtracking shrink factor in Week 6, a step-schedule constant in Weeks 7 and 11, and a momentum coefficient in Weeks 10–11; β_1, β_2 are Adam’s averaging constants in Week 11. The letter γ is a risk-aversion weight in the portfolio objective of Weeks 1 and 5 but a risk budget (a cap on the standard deviation) in Week 4; it is also a step-schedule constant in Weeks 7 and 11, and γ_k is the Frank–Wolfe step in Week 13. The letter μ is a vector of expected returns in the portfolio model and the strong-convexity constant in the algorithm weeks, whose prototype is the μ of $A^\top A + \mu I$ in Week 3. The letter ρ is the contraction factor of the fixed-step bound in Week 11 and the ADMM penalty in Weeks 12–13.

The letter L is the Lagrangian, always written with its arguments, in Weeks 5, 12 and 13, and a bare smoothness constant from Week 6 onwards. The letter g is an inequality-constraint function g_i (Weeks 1 and 8), a subgradient (Weeks 3, 5 and 7), the dual function $g(\lambda, \nu)$ (Week 5), the smooth part of a composite objective (Weeks 9–10), a sampled gradient g^k (Week 11) and the second ADMM term $g(z)$ (Week 12), while g_k is the Frank–Wolfe gap (Week 13). h_j is an equality-constraint function, whereas a plain h is the regulariser from Week 9 on. The letter F is

the composite objective $g + h$ in Weeks 9–10, the smooth finite sum in Week 11, and $f + \lambda \sum_i h_i$ in Week 12. The residual G_α of Weeks 8–10 and 13 is unrelated to the bound G on (sub)gradient norms of Weeks 7 and 11. R is a revenue function in Week 4, an initial distance in Week 7 and a shared resource in Week 12. s is the mini-batch size in Week 11, while s^k is the ADMM dual residual in Week 12 and the Frank–Wolfe vertex in Week 13.

The running examples keep their formulation throughout, but not always their letters. The coefficient vector of the data-fitting examples is β in Weeks 1, 5, 7 and 9, θ in the logistic model of Week 4, w in the logistic loss of Week 6 and x elsewhere. Portfolio weights are w in Weeks 1, 2, 5 and 13 and the generic decision vector x in Weeks 4 and 8. The feasible set is $\mathcal{X}$ in the early weeks and C in Weeks 8–10 and 13; Week 3 also uses $\mathcal{X}$ for the domain of f in its derivative tests. Week 12 makes a_i a *column* of the data matrix and lets m count business units or ADMM blocks; Week 13 returns to rows $a_i^\top$ and to m inequality constraints. Wherever two meanings meet in one week, the text says which is meant at the point of change.

Contents

About these notes	ii
Notation	iv
Week 1 Introduction: Optimisation Models and Foundations	1
1.1 Decisions as optimisation models	1
1.2 The vocabulary of optimisation	4
1.3 The role of convexity	6
1.4 Linear algebra review	7
1.5 Calculus review	10
1.6 A first model in RSOME	14
1.7 Summary and looking ahead	15
1.8 Exercises	15
Week 2 Convex Sets, Cones and Convexity-Preserving Operations	17
2.1 Convex combinations and convex sets	17
2.2 A catalogue of standard convex sets	21
2.3 Operations that preserve convexity	24
2.4 Recognising convex feasible regions	29
2.5 Summary and looking ahead	30
2.6 Exercises	31
Week 3 Convex Functions and How to Recognise Them	32
3.1 From convex sets to convex functions	32
3.2 Convex functions of one variable	34
3.3 Convex functions of several variables	35
3.4 Operations that preserve convexity	40
3.5 Recognising convex functions in practice	44
3.6 Summary and looking ahead	46
3.7 Exercises	46
Week 4 Convex Optimisation Modelling and the RSOME Workflow	48
4.1 From a business problem to a standard convex model	48
4.2 Model families and reformulation patterns	51
4.3 Convexity by modelling	55
4.4 Conic representation and modelling hygiene	58
4.5 The RSOME workflow	60
4.6 Summary and looking ahead	62
4.7 Exercises	63
Week 5 Optimality Conditions, Duality and Shadow Prices	64
5.1 Certifying an unconstrained minimum	64
5.2 Constrained optimality: active constraints and balancing normals	66
5.3 Lagrangian duality	69
5.4 The Karush–Kuhn–Tucker conditions	71
5.5 Linear programming duality, shadow prices and diagnostics	74
5.6 Summary and looking ahead	78
5.7 Exercises	78

Week 6 Smooth Unconstrained Optimisation and Gradient Descent	80
6.1 From optimality conditions to an iterative template	80
6.2 Local models and the descent direction	81
6.3 Choosing the step size	84
6.4 How fast, and when to stop	87
6.5 Applications and verdict	91
6.6 Summary and looking ahead	94
6.7 Exercises	94
Week 7 Nonsmooth Convex Optimisation and the Subgradient Method	96
7.1 Why nonsmooth problems need their own treatment	96
7.2 Subgradients and the subdifferential	97
7.3 Optimality and the subgradient update	100
7.4 Behaviour and convergence of the subgradient method	102
7.5 Diagnostics, robust regression, and the verdict	108
7.6 Summary and looking ahead	110
7.7 Exercises	111
Week 8 Simple Constraints, Projection and Projected Gradient	112
8.1 When the unconstrained step fails	112
8.2 Constrained optimality: indicators and normal cones	114
8.3 Projection as a computable primitive	117
8.4 The projected-gradient method	120
8.5 Worked examples and extensions	123
8.6 Summary and looking ahead	125
8.7 Exercises	126
Week 9 Regularisation and the Proximal Gradient Method	128
9.1 Why penalise, and why not the subgradient method	128
9.2 The composite model and its optimality condition	130
9.3 The proximal operator	132
9.4 The proximal gradient method	135
9.5 Convergence, tuning and implementation	138
9.6 Summary and looking ahead	142
9.7 Exercises	142
Week 10 Accelerated First-Order Methods	144
10.1 Why gradient descent crawls, and the momentum idea	144
10.2 The quadratic laboratory: why momentum works	147
10.3 Rates and schedules for general convex problems	150
10.4 Accelerating projection and prox	152
10.5 Practice: restart, stopping and benchmarks	155
10.6 Summary and looking ahead	158
10.7 Exercises	158
Week 11 Optimisation for Large Datasets: Stochastic Gradient Methods	160
11.1 The finite-sum model and the cost of one gradient	160
11.2 Sampling the gradient	161
11.3 What the analysis says	164
11.4 Step sizes, batching, momentum and adaptive steps	170
11.5 Scope: constraints, nonsmooth losses, variance reduction, and when not to sample	174
11.6 Summary and looking ahead	176

11.7 Exercises	176
Week 12 Large-Scale Optimisation and Decomposition	178
12.1 Structure as a source of scale	178
12.2 Coordinate descent	180
12.3 From coordinates to blocks, and proximal-coordinate steps	184
12.4 Coupling constraints and prices: dual decomposition	186
12.5 The augmented Lagrangian and ADMM	189
12.6 Summary and looking ahead	193
12.7 Exercises	194
Week 13 Optimisation with Complex Constraints	196
13.1 The bottleneck: when projection stops being cheap	196
13.2 Escape 1: a linear oracle and the Frank–Wolfe method	198
13.3 Escape 2: moving the constraints into the objective	203
13.4 Escape 3: the log barrier and interior-point methods	204
13.5 Choosing an escape, and the course in one map	207
13.6 Summary and looking ahead	210
13.7 Exercises	210
Appendix: Discrepancies between the notes and the lecture slides	212

Week 1 Introduction: Optimisation Models and Mathematical Foundations

Much of business analytics comes down to a single task: choosing the best decision from a set of allowed alternatives. A machine-learning engineer fits model parameters by minimising a training loss; a statistician estimates a parameter by maximising a likelihood; a portfolio manager chooses asset weights that balance risk against expected return; a marketing manager splits a budget across channels to maximise conversions. The vocabulary of these fields differs, but the mathematical structure is the same, and *mathematical optimisation* is the discipline that studies it. This book is about making better decisions with optimisation, and its first week asks the question that frames the rest: *when is a business decision ready to hand to a solver?*

The answer has two parts. A decision is ready, first, when it has been written down as an explicit *model*: what is chosen, what is known, what is wanted and what is not allowed. It is ready, second, when that model has a structure that a *solver*, the program that computes the best decision of a model, can exploit. The structure that matters most is *convexity*, and recognising, building and exploiting convex models is the thread that runs through all thirteen weeks. By the end of the book the reader should be able to formulate business decisions as optimisation models, explain the theory that makes them solvable, choose and implement methods that fit a problem's structure, and communicate the result as a recommendation.

Nothing is assumed beyond basic Python and NumPy, elementary linear algebra and single-variable calculus. [Section 1.1](#) shows how decisions become models, using three examples that recur all semester, and [Section 1.2](#) fixes the vocabulary that every later week uses. [Section 1.3](#) explains why not every model can be solved and why convexity is the dividing line. [Sections 1.4](#) and [1.5](#) review exactly the linear algebra and calculus on which later weeks rely, and [Section 1.6](#) builds and solves a first model in Python.

1.1 Decisions as optimisation models

1.1.1 The four elements of a model

Optimisation modelling is the translation of a real decision problem into mathematics. Every model, whatever its origin, is assembled from four elements, and each answers a plain question. The *decision variables* are the unknowns, written $x_1, x_2, x_3, \dots$: what is the decision to be made? The **data** or *parameters*, written with letters such as a, b, c , are the numbers the decision maker does not control: what information is available? The *objective function* is a function of the decision variables that is to be minimised or maximised: what is the goal? The *constraints* are equations or inequalities that the decision variables must satisfy: what limits the choice?

[Table 1.1](#) reads the four applications of the opening paragraph through this lens. Each row deliberately keeps the notation of its own field: in the machine-learning row (x_i, y_i) denotes the features and label of training record i , which are data, and the decision variable is the parameter vector θ . Everywhere else in this book x denotes the decision variable.

1.1.2 Three running examples

Three small models make the four elements concrete. Each returns several times in later weeks, and together they span the structural classes met most often: linear, quadratic and unconstrained.

Model 1.1 (Media-mix allocation). Weekly advertising spend on search, social and video:

$$\begin{aligned} \max \quad & 4x_1 + 3x_2 + 2x_3 \\ \text{s.t.} \quad & x_1 + x_2 + x_3 \leq 10000, \\ & x_1 \geq 2000, \quad x_2 \leq 4000, \\ & x_1, x_2, x_3 \geq 0. \end{aligned}$$

Table 1.1: Four decision problems described by the four elements of an optimisation model. Each row uses the customary notation of its field.

Application	Decision variables	Data and parameters	Objective	Constraints
Machine learning	model parameters θ	training data (x_i, y_i) , regularisation weight λ	minimise the loss	model class, bounds
Statistics	parameter θ	sample $y_1, \dots, y_N$	maximise the likelihood	$\theta \in \Theta$
Finance	portfolio weights w	expected returns μ , covariance Σ , a risk limit	maximise $\mu^\top w$	risk limit, budget, $w \geq 0$
Marketing	channel spend x	budget B , response curve	maximise conversions	budget, channel caps

The decision variables x_1, x_2, x_3 are the dollars spent on search, social and video. The objective coefficients 4, 3, 2 are data: the predicted number of conversions per dollar in each channel, so the objective is the predicted total number of conversions. The first constraint is the budget of \$10,000; the next two are policy rules (search receives at least \$2,000, social at most \$4,000); the last says that spend cannot be negative. Because the objective and every constraint are linear in x , this is a **linear optimisation** model (a linear programme, LP). The model has turned a sentence of business English into something a solver can work with.

Linearity is also the model's main assumption: each extra dollar in a channel is predicted to buy the same number of conversions as the first, with no diminishing returns and no interaction between channels. An LP of this size is solved instantly by any LP solver (Section 1.6 shows how), and here the solution can be found by inspection. A dollar is worth most in search and nothing but the budget caps search spend, so the whole budget goes there: $x^* = (10000, 0, 0)$ with 40,000 predicted conversions, and both policy rules hold with room to spare. Reading a solution means asking whether it is believable. An “all-in on one channel” recommendation is a direct consequence of constant returns; an analyst who finds it implausible has learned that the model needs a response curve with diminishing returns, not a different solver.

Model 1.2 (Long-only portfolio). An investor divides wealth among d assets:

$$\begin{aligned}
 \max \quad & \mu^\top w - \gamma w^\top \Sigma w \\
 \text{s.t.} \quad & \mathbf{1}^\top w = 1, \\
 & w \geq 0.
 \end{aligned}$$

The decision variable $w \in \mathbb{R}^d$ holds the asset weights: w_i is the fraction of wealth placed in asset i . The parameters $\mu \in \mathbb{R}^d$ and $\Sigma \in \mathbb{R}^{d \times d}$ are given data, estimated elsewhere: μ_i is the anticipated return per dollar invested in asset i , and Σ_{ij} , the covariance of the returns of assets i and j , measures how strongly the two move together. The portfolio then has anticipated return $\mu^\top w$ and return variance $w^\top \Sigma w$, the measure of risk. The objective trades one against the other, and the parameter $\gamma \geq 0$, the investor's degree of risk aversion, sets the exchange rate. The constraint $\mathbf{1}^\top w = 1$, where $\mathbf{1}$ is the vector of ones, says that all wealth is allocated, and $w \geq 0$ (every component nonnegative) rules out short positions, which is the meaning of *long-only*. The objective contains the quadratic term $w^\top \Sigma w$ and the constraints are linear, so this is a **quadratic optimisation** model. It assumes that μ and Σ are known, that variance is an adequate measure of risk, and that trading is free. A variant is handed to a solver in Week 4; the model itself is analysed through its optimality conditions in Week 5 and solved with our own algorithms from Week 8 onwards.

The solution is read through γ : with $\gamma = 0$ all wealth goes to the asset with the highest anticipated return, as γ grows the risk term spreads wealth across assets, and in the limit the

weights approach the portfolio of least variance. Diversification is not imposed; it emerges from the risk term.

Note on the lecture slides

The slides describe the finance problem in three ways: minimise risk subject to an expected-return target (frame 1.2); maximise $\mu^\top w$ subject to a risk constraint, with data “ μ, Σ, R ” in which R is never defined (frame 1.3); and the weighted trade-off above (frame 1.5). These are alternative ways of balancing the same two criteria. The notes adopt [Model 1.2](#) as the running model and read R as a risk limit; [Model 4.15](#) returns to that form.

Model 1.3 (Least squares). Fit a linear prediction rule to N data records:

$$\begin{aligned} \min \quad & \frac{1}{N} \sum_{i=1}^N (b_i - a_i^\top \beta)^2 \\ \text{s.t.} \quad & \beta \in \mathbb{R}^d. \end{aligned}$$

Here the data are the records (a_i, b_i) , $i = 1, \dots, N$, where $a_i \in \mathbb{R}^d$ holds the features of record i and $b_i \in \mathbb{R}$ its response; they are fixed numbers. The decision variable is the parameter vector $\beta \in \mathbb{R}^d$: the rule predicts $a_i^\top \beta$ for record i , the residual $b_i - a_i^\top \beta$ is its error, and the objective is the mean squared error over the data set. The “constraint” $\beta \in \mathbb{R}^d$ excludes nothing, so the problem is **unconstrained**. The modelling assumptions are that a linear rule is adequate and that squared error, which punishes large errors heavily, is the right loss. The example shows why optimisation matters for machine learning: *training a model means minimising a loss function*. Least squares is one of the few problems solved by a formula, derived in [Sections 1.4](#) and [1.5](#); for large N it becomes the test problem for gradient descent in [Week 6](#) and for stochastic gradient methods in [Week 11](#). The optimal β^* is the fitted model, and the optimal value is its training error.

1.1.3 The modelling workflow and why it pays

All three examples followed the same four-step pipeline, which we shall revisit throughout the book:

business question $\longrightarrow$ model $\longrightarrow$ solve $\longrightarrow$ recommendation.

A business question is turned into a model (variables, objective, constraints), the model is solved by a solver or an algorithm, and the solution is translated back into a recommendation with its trade-offs and limits. A good answer contains all four pieces: a clear question, an explicit model, an appropriate method and an interpretable recommendation.

The discipline rests on three pillars that feed each other. *Modelling* expresses objectives, decisions and constraints explicitly, and its difficulties motivate new theory. *Theory* (optimality conditions, convexity, duality) provides guarantees on feasibility, optimality and solution quality, and guides the design of algorithms. *Computation* finds optimal or near-optimal decisions at practical scale, and what can be solved guides modelling choices in turn. The analyst gains explicit assumptions that are open to challenge, the ability to handle many variables and constraints simultaneously, and transparent, reproducible decisions. Optimisation delivers the best feasible decision *for a stated model*, not merely a plausible guess, and the qualifier matters, as the media-mix solution has just shown. For the same reason AI assistants, useful for clarifying definitions, generating small examples and debugging code, support the workflow without replacing understanding: the analyst remains accountable for checking the formulation and for interpreting solver output critically.

1.2 The vocabulary of optimisation

1.2.1 The generic model and its feasible set

The three examples differ in their details but share one template, and the template is what the theory of later weeks is written for.

Definition 1.4 (Optimisation model). An **optimisation model** (or problem) has the form

$$\begin{array}{ll} \min / \max & f(x) \\ \text{s.t.} & x \in \mathcal{X}, \end{array} \quad (1.1)$$

where $x \in \mathbb{R}^d$ is the vector of **decision variables**, the quantities we choose; $f: \mathbb{R}^d \rightarrow \mathbb{R}$ is the **objective function**, which says what “better” means; and $\mathcal{X} \subseteq \mathbb{R}^d$ is the **feasible set**, the set of all allowed decisions. The feasible set is typically described by **constraints**,

$$\mathcal{X} = \{x \in D : g_i(x) \leq 0, i = 1, \dots, m, \quad h_j(x) = 0, j = 1, \dots, p\},$$

with m **inequality constraints** $g_i(x) \leq 0$, p **equality constraints** $h_j(x) = 0$ and a **domain restriction** $D \subseteq \mathbb{R}^d$. All other quantities that appear in f , g_i and h_j are **parameters** (data), given by the problem.

Writing every inequality as “function ≤ 0 ” and every equality as “function $= 0$ ” costs nothing. In [Model 1.1](#) the budget becomes $g_1(x) = x_1 + x_2 + x_3 - 10000 \leq 0$, the rule $x_1 \geq 2000$ becomes $g_2(x) = 2000 - x_1 \leq 0$, the cap on social becomes $g_3(x) = x_2 - 4000 \leq 0$, and nonnegativity is either three further inequalities $-x_k \leq 0$ or the domain restriction $D = \mathbb{R}_+^3$, the vectors with nonnegative components ([Section 1.4](#)); in [Model 1.2](#) there is one equality constraint, $h_1(w) = \mathbf{1}^\top w - 1 = 0$. Nor is the choice between min and max a real distinction: maximising f over $\mathcal{X}$ is the same problem as minimising $-f$, with the same optimal decisions and the optimal value negated. Business models are often naturally maximisations, and we write them that way, while theory and algorithms from [Week 3](#) onwards are stated for minimisation. [Weeks 4](#) and [5](#) will write f_0 for the objective and f_i for the inequality constraint functions, with the same meaning as f and g_i here.

1.2.2 Feasible and optimal solutions

With the template fixed, we can say what it means for a decision to be allowed and to be best.

Definition 1.5 (Solution concepts). Consider the model (1.1). A point x is a **feasible solution** if it satisfies all the constraints, that is, $x \in \mathcal{X}$; otherwise it is **infeasible**. An **optimal solution** is a best feasible solution: a point $x^* \in \mathcal{X}$ with $f(x^*) \leq f(x)$ for all $x \in \mathcal{X}$ in a minimisation problem ($f(x^*) \geq f(x)$ in a maximisation problem). The **optimal value** z^* is the objective value $f(x^*)$ at any optimal solution. ([Section 1.4.2](#) extends z^* to $\inf_{x \in \mathcal{X}} f(x)$, respectively $\sup_{x \in \mathcal{X}} f(x)$, so that it is defined for every problem, attained exactly when an optimal solution exists.)

A star always marks an optimal quantity. A problem may have many optimal solutions, but they all share one optimal value. Feasibility answers the question “is this decision allowed?”; optimality answers “is it the best among the allowed?”

Example 1.6 (A two-dimensional linear programme). Consider

$$\begin{array}{ll} z^* = \max & 2x_1 + 3x_2 \\ \text{s.t.} & x_1 \leq 3, \quad x_2 \leq 3, \quad x_1 + x_2 \leq 5, \quad x_1, x_2 \geq 0. \end{array}$$

The feasible set. Each constraint keeps the points on one side of a line, and the feasible set, where all five hold, is the shaded pentagon of [Figure 1.1](#) with corners $(0, 0)$, $(3, 0)$, $(3, 2)$, $(2, 3)$ and

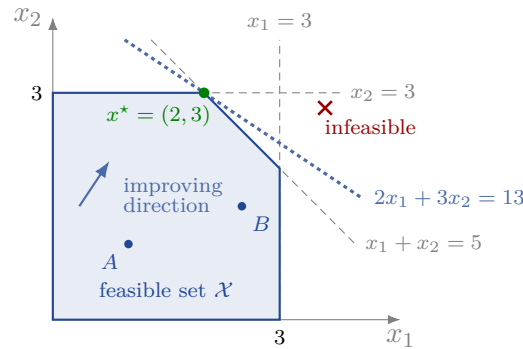


Figure 1.1: The linear programme of Example 1.6. The feasible set is the shaded pentagon; A and B are feasible, the cross is not. Pushing a contour of the objective in the improving direction $(2, 3)$, the last contour to touch the feasible set is $2x_1 + 3x_2 = 13$ (dotted), and it touches at the corner $x^* = (2, 3)$.

$(0, 3)$. The points $A = (1, 1)$ and $B = (2.5, 1.5)$ are feasible, with objective values 5 and 9.5; the point $(3.6, 2.8)$ is infeasible, since it violates $x_1 \leq 3$ (and $x_1 + x_2 \leq 5$), so its attractive objective value 15.6 is irrelevant.

The optimal solution. The points with objective value c form the line $2x_1 + 3x_2 = c$, called a *contour* (line of equal value) of the objective. As c grows, this line moves parallel to itself in the improving direction $(2, 3)$. The largest c for which the line still touches the pentagon is $c = 13$, where it passes through the corner $(2, 3)$. Evaluating the objective at the five corners, 0, 6, 12, 13 and 9, confirms that $(2, 3)$ is the best of them. Hence $x^* = (2, 3)$ and $z^* = 2(2) + 3(3) = 13$.

A certificate. Optimality can also be verified without a picture: every feasible x satisfies $2x_1 + 3x_2 = 2(x_1 + x_2) + x_2 \leq 2 \cdot 5 + 3 = 13$, by the constraints $x_1 + x_2 \leq 5$ and $x_2 \leq 3$, and equality holds only at $(2, 3)$, so the optimal solution is unique. Combining constraints to bound the objective is the germ of duality theory (Week 5).

1.2.3 Three solution statuses

Not every model has an optimal solution, and a solver reports which situation it met.

Definition 1.7 (Solution statuses). An optimisation problem is

- **optimal** (it “has status optimal”) if a best feasible point exists;
- **infeasible** if the feasible set is empty, $\mathcal{X} = \emptyset$;
- **unbounded** if it is feasible but the objective can be improved without limit over $\mathcal{X}$.

Three one-variable problems show the cases. The problem $\max x$ subject to $0 \leq x \leq 5$ has feasible set $\mathcal{X} = [0, 5]$, optimal solution $x^* = 5$ and optimal value $z^* = 5$. The problem $\max x$ subject to $x \leq 1$ and $x \geq 3$ has $\mathcal{X} = (-\infty, 1] \cap [3, \infty) = \emptyset$: the constraints conflict, and there is nothing to optimise. The problem $\max x$ subject to $x \geq 0$ has $\mathcal{X} = [0, \infty)$, and for every feasible point there is a better one, so the objective value grows without limit.

Each status is a message about the model and not only about the mathematics. *Optimal* means that a recommendation is available. *Infeasible* means that the rules as written conflict, and some requirement has to be relaxed before any decision can be recommended. *Unbounded* means that a cap is missing: no real business can earn unlimited profit, so a limit on some resource has been left out. The last two statuses are almost always modelling errors, and the status is the first thing to read after every solve.

Remark 1.8 (A fourth possibility). For linear programmes such as those of this week exactly one of the three statuses occurs. In general there is one more case: the objective may be bounded and yet no best point may exist. The problem $\max x$ subject to $0 \leq x < 5$ is feasible and no

feasible value exceeds 5, but every feasible point can be improved upon. [Section 1.4.2](#) supplies the language for this case.

1.3 The role of convexity

1.3.1 Can every model be solved?

Once a model is formulated the next question is computational. To *solve* a model means to find a global optimum or to certify that the problem is infeasible or unbounded; nothing less settles the decision. The bad news is that in general this cannot be done. In his *Lectures on Convex Optimization*, Yurii Nesterov calls it the main fact that anyone dealing with optimisation models should know: “in general, optimization problems are unsolvable”. Without useful structure, finding a best point among infinitely many candidates can take an amount of computation that explodes with the number of variables, and no clever software changes that.

The good news is that one class of problems, *convex optimisation*, is expressive enough to cover a great many practical models and can be solved reliably and efficiently: efficient algorithms are available, global optimality can often be certified, and infeasibility and unboundedness can often be detected. Informally, a set is *convex* if the straight segment between any two of its points stays inside it, and a function is convex if its graph never rises above the chord joining two of its points (every chord lies on or above the graph), so that it is bowl-shaped like the right-hand curve of [Figure 1.2](#); a convex optimisation problem minimises a convex function over a convex set, and [Weeks 2](#) and [3](#) make both notions precise. Linear functions, and feasible sets cut out by linear constraints, are convex; so are the squared error and the risk term $w^\top \Sigma w$ of a covariance matrix. All three models of [Section 1.1](#) are therefore convex problems once each maximisation is rewritten as the minimisation of $-f$ ([Section 1.2](#)). R. T. Rockafellar observed in 1993 that the great watershed in optimisation lies not between linearity and nonlinearity but between convexity and nonconvexity; Stephen Boyd likens solving a convex problem to solving a least-squares problem. The portfolio model is nonlinear and easy; a model with linear objective and constraints whose variables must be whole numbers (an integer programme) can be very hard, because the whole-number requirement destroys convexity. Formulating a model is therefore only half the job, and this book concentrates on convex models precisely because their solutions can be computed and trusted.

1.3.2 Why general problems are hard

Real problems may be nonlinear, nonsmooth (the objective has kinks, so that derivatives are not available everywhere), constrained and large, all at once. The deepest difficulty is nonconvexity. A nonconvex objective, like the left curve of [Figure 1.2](#), is a landscape with several valleys. An algorithm that improves its current point step by step, as practical algorithms do, slides into the nearest valley and stops there, and nothing visible from the bottom of that valley reveals that a deeper one exists. A convex bowl, on the right, has a single bottom, and walking downhill from anywhere reaches it. A solver can only exploit structure that it can recognise, so the book asks four questions of every model: *Is it convex? Is it smooth? Are the constraints simple? Which solver or algorithm fits?*

1.3.3 The central promise and the plan of the book

Call a feasible point a *local minimum* if no nearby feasible point is better, and a *global minimum* if no feasible point at all is better; [Definition 1.20](#) makes this precise once norms are available. The reliability of convex optimisation rests on one statement, which this week can only preview.

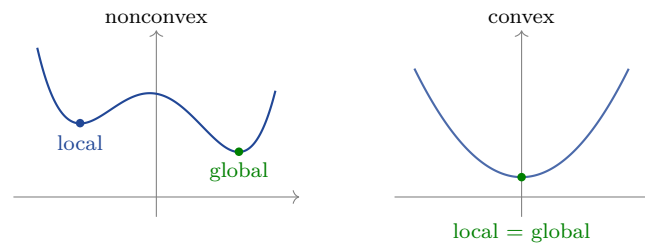


Figure 1.2: A nonconvex landscape (left) has a local minimum that is not the global one, and an algorithm that only ever moves downhill, started on the left of the hill, is trapped in it. A convex bowl (right) has one bottom: every local minimum is global.

Key idea

If a convex objective is minimised over a convex feasible set, then every local minimum is a global minimum.

The statement explains what a convex model buys: an algorithm that can only look around locally nevertheless finds the best decision overall. Making it precise and proving it takes the next three weeks, and the book is organised around it. [Weeks 1 to 3](#) lay the foundations: vocabulary, convex sets and convex functions. [Weeks 4 and 5](#) build solver-ready convex models in R and develop optimality conditions, duality and shadow prices, with which solutions are certified and interrogated. [Weeks 6 to 10](#) develop first-order algorithms that exploit convex structure (gradient descent, subgradient, projected-gradient and proximal methods, acceleration). [Weeks 11 and 12](#) address scale, in the data and in the model, and [Week 13](#) treats constraints too complicated for the earlier methods before drawing the course together.

1.4 Linear algebra review

Large models are written, analysed and coded with vectors and matrices. This subsection and the next collect exactly the linear algebra and calculus that later weeks use.

1.4.1 Scalars, vectors, matrices and norms

A **scalar** is a real number, $c \in \mathbb{R}$. We write $\mathbb{R}_+ = [0, \infty)$ and $\mathbb{R}_{++} = (0, \infty)$ for the nonnegative and the positive reals, so that $\mathbb{R}_+^d$ is the set of vectors with nonnegative components. A **vector** $x \in \mathbb{R}^d$ is a column of d real numbers with **components** $x_1, \dots, x_d$; throughout the book d denotes the dimension of the decision variable and N the number of data records. A **matrix** $A \in \mathbb{R}^{m \times d}$ has m rows and d columns; its j th row is written A_j ; and its k th column $A_{\cdot k}$. The **transpose** $x^\top$ turns the column x into a row, and $A^\top \in \mathbb{R}^{d \times m}$ exchanges the rows and columns of A . An inequality between vectors is read componentwise, so that $x \geq 0$ means $x_i \geq 0$ for every i . The product Ax is defined only when A has as many columns as x has components, and it has one component for each row of A :

$$x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \in \mathbb{R}^3, \quad A = \begin{bmatrix} 2 & 1 & 0 \\ 1 & 3 & 2 \end{bmatrix} \in \mathbb{R}^{2 \times 3}, \quad Ax = \begin{bmatrix} 2x_1 + x_2 \\ x_1 + 3x_2 + 2x_3 \end{bmatrix} \in \mathbb{R}^2.$$

Checking dimensions in this way is a habit worth forming, because dimension mistakes are among the most common modelling and coding errors. In a business model a vector x holds one decision or measurement per item (spend by channel, weight by asset), a matrix A arranges many coefficients systematically (resource usage by product, a data table of features), and a right-hand-side vector b collects limits or targets such as capacities or demands.

Table 1.2: Supremum and maximum of four sets. The supremum always exists; the maximum exists only when the supremum is attained.

Set S	$\sup S$	$\max S$	Comment
$[0, 5]$	5	5	the bound is attained
$[0, 5)$	5	none	5 is approached but never reached
$\{1/n : n = 1, 2, 3, \dots\}$	1	1	but $\inf S = 0$, and there is no minimum
$[0, \infty)$	$+\infty$	none	unbounded above

Definition 1.9 (ℓ_p -norms). For $p \geq 1$, the ℓ_p -**norm** of $x \in \mathbb{R}^d$ is $\|x\|_p = (\sum_{i=1}^d |x_i|^p)^{1/p}$; it measures the size of the vector. Three cases matter in this book: the **Euclidean norm** $\|x\|_2 = \sqrt{x^\top x}$, the ordinary length; the ℓ_1 -norm $\|x\|_1 = \sum_i |x_i|$, the sum of absolute values; and the ℓ_∞ -norm $\|x\|_\infty = \max_i |x_i|$, the largest component in magnitude (the limit of $\|x\|_p$ as $p \rightarrow \infty$). The **distance** between x and y is $\|x - y\|_2$. A norm written without a subscript, $\|x\|$, always means the Euclidean norm.

For $x = (3, -4)$, for instance, $\|x\|_1 = 7$, $\|x\|_2 = 5$ and $\|x\|_\infty = 4$. The Euclidean norm is the default measure of distance in every algorithm from [Week 6](#) onwards; the ℓ_1 -norm returns in [Week 9](#) as a regulariser.

1.4.2 Supremum and infimum

[Remark 1.8](#) met a problem whose objective is bounded although no best point exists. The language for such situations separates the best *bound* on a set of numbers from the question of whether that bound is *attained*.

Definition 1.10 (Supremum, infimum, maximum, minimum). Let $S \subseteq \mathbb{R}$. The **supremum** $\sup S$ is the least upper bound of S : the smallest u such that $s \leq u$ for all $s \in S$. The **infimum** $\inf S$ is the greatest lower bound. If S has no upper bound then $\sup S = +\infty$, and if it has no lower bound then $\inf S = -\infty$; for the empty set, $\sup \emptyset = -\infty$ and $\inf \emptyset = +\infty$. With these values allowed, $\sup S$ and $\inf S$ always exist. If $\sup S$ is a member of S it is called the **maximum**, $\max S$; if $\inf S$ belongs to S it is the **minimum**, $\min S$.

A maximum asks for more than a supremum: the bound must be attained by a member of the set. Whenever the maximum exists it equals the supremum, and likewise for the minimum and the infimum ([Table 1.2](#)).

Optimisation needs the distinction because it lets the optimal value be defined for *every* problem: formally,

$$z^* = \inf_{x \in \mathcal{X}} f(x) \quad \text{for a minimisation problem,} \quad z^* = \sup_{x \in \mathcal{X}} f(x) \quad \text{for a maximisation problem.}$$

When an optimal solution exists this agrees with [Definition 1.5](#), since the bound is attained at x^* . A maximisation problem is unbounded exactly when $z^* = +\infty$ and, by the convention for the empty set, infeasible exactly when $z^* = -\infty$ (for minimisation the signs are reversed); for $\max x$ subject to $0 \leq x < 5$, the value $z^* = 5$ is finite but not attained. Later weeks also use $\sup$ and $\inf$ inside models, notably in the dual function of [Week 5](#).

1.4.3 Inner products, linear constraints and the Cauchy–Schwarz inequality

Two expressions recur in almost every model. The first is the **inner product** of $a, x \in \mathbb{R}^d$, $a^\top x = \sum_{i=1}^d a_i x_i$, a weighted total: a score, a revenue or a cost. The objective of [Model 1.1](#) is $a^\top x$ with $a = (4, 3, 2)$, its budget constraint is $\mathbf{1}^\top x \leq 10000$, and the squared length of a vector is its inner product with itself, $\|x\|_2^2 = x^\top x$. The second expression is $Ax \leq b$ with $A \in \mathbb{R}^{m \times d}$

and $b \in \mathbb{R}^m$, which states m linear limits at once: row j reads $A_j x \leq b_j$, so the rows of A correspond to separate constraints. With $A = \begin{bmatrix} 2 & 1 \\ 1 & 3 \end{bmatrix}$ and $b = (100, 80)$, the statement $Ax \leq b$ means $2x_1 + x_2 \leq 100$ and $x_1 + 3x_2 \leq 80$, two capacity limits shared by two products. The inner product and the Euclidean norm are tied together by the most frequently used inequality of the book.

Proposition 1.11 (Cauchy–Schwarz inequality). *For all $a, x \in \mathbb{R}^d$,*

$$|a^\top x| \leq \|a\|_2 \|x\|_2,$$

with equality exactly when one of the two vectors is a scalar multiple of the other.

Proof. If $x = 0$ both sides vanish. Otherwise put $t = a^\top x / \|x\|_2^2$. Expanding the square gives $0 \leq \|a - tx\|_2^2 = \|a\|_2^2 - 2ta^\top x + t^2 \|x\|_2^2 = \|a\|_2^2 - (a^\top x)^2 / \|x\|_2^2$, which rearranges to $(a^\top x)^2 \leq \|a\|_2^2 \|x\|_2^2$. Equality requires $a = tx$, and multiples clearly attain it. $\square$

An inner product can never exceed the product of the two lengths. For $a = (2, 1)$ and $x = (1, 3)$ we find $a^\top x = 5$ while $\|a\|_2 \|x\|_2 = \sqrt{5}\sqrt{10} \approx 7.07$; for the parallel pair $a = (2, 1)$, $x = (4, 2)$ both sides equal 10. The inequality turns inner products into estimates of distance: it justifies the phrase “steepest ascent” in [Section 1.5](#), yields the triangle inequality of [Lemma 2.13](#), and carries the proofs in [Weeks 8](#) and [9](#) that projections and proximal operators never increase distances.

1.4.4 Linear independence and the normal equations

When does a linear system have exactly one solution? The answer turns on whether some columns of the matrix are redundant.

Definition 1.12 (Linear independence). Vectors $v_1, \dots, v_k$ are **linearly independent** if $c_1 v_1 + \dots + c_k v_k = 0$ forces $c_1 = \dots = c_k = 0$. They are **linearly dependent** otherwise, which happens exactly when one of them is a linear combination of the others.

A dependent vector adds no new direction: two vectors in the plane that are not multiples of one another are independent and their combinations fill the plane, whereas the combinations of v_1 and a multiple of v_1 fill only a line. For a matrix A the combination $c_1 A_{:,1} + \dots + c_d A_{:,d}$ of its columns is the product Ac , so the columns of A are independent precisely when $Az = 0$ implies $z = 0$.

Proposition 1.13 (Invertibility of $A^\top A$). *Let $A \in \mathbb{R}^{N \times d}$ have linearly independent columns, that is, $Az = 0$ only for $z = 0$. Then the $d \times d$ matrix $A^\top A$ is invertible, and for every $b \in \mathbb{R}^N$ the **normal equations***

$$A^\top A x = A^\top b \tag{1.2}$$

have the unique solution $x = (A^\top A)^{-1} A^\top b$.

Proof. Suppose $A^\top A z = 0$. Then $0 = z^\top A^\top A z = \|Az\|_2^2$, hence $Az = 0$ and, by independence of the columns, $z = 0$. A square matrix that sends only the zero vector to zero is invertible, and multiplying (1.2) by the inverse gives the formula. $\square$

The normal equations themselves are the equations that solve least squares. Stack the records of [Model 1.3](#) into a matrix $A \in \mathbb{R}^{N \times d}$ whose i th row is $a_i^\top$ and a vector $b \in \mathbb{R}^N$ of responses. Then $\frac{1}{N} \sum_{i=1}^N (b_i - a_i^\top \beta)^2 = \frac{1}{N} \|A\beta - b\|_2^2$. Later weeks write the unknown as x instead of β and the objective as $\frac{1}{2} \|Ax - b\|_2^2$; the positive factors $1/N$ and $1/2$ change the optimal value but not the minimiser, and [Section 1.5](#) shows that the minimiser must satisfy (1.2). Independent columns mean that no feature is a combination of the other features (which requires $N \geq d$), and the proposition then says that the best fit is unique, a fact [Week 5](#) uses when it certifies the

least-squares solution. If a feature is duplicated, $A^\top A$ is singular and infinitely many coefficient vectors fit equally well. (Applied to $A^\top$, the same proof shows that $AA^\top$ is invertible when the rows of A are independent, the form used in [Week 8](#).)

Example 1.14 (A line through three points). Fit the line $b \approx x_1 + x_2 t$ to the points $(t, b) = (0, 0), (1, 2), (2, 1)$. Each record has features $a_i = (1, t_i)$, so

$$A = \begin{bmatrix} 1 & 0 \\ 1 & 1 \\ 1 & 2 \end{bmatrix}, \quad b = \begin{bmatrix} 0 \\ 2 \\ 1 \end{bmatrix}, \quad A^\top A = \begin{bmatrix} 3 & 3 \\ 3 & 5 \end{bmatrix}, \quad A^\top b = \begin{bmatrix} 3 \\ 4 \end{bmatrix}.$$

The columns of A are not multiples of each other, so the normal equations $3x_1 + 3x_2 = 3$, $3x_1 + 5x_2 = 4$ have exactly one solution: subtracting the first from the second gives $x_2 = 0.5$, and then $x_1 = 0.5$. The fitted line $b = 0.5 + 0.5t$ leaves residuals $Ax - b = (0.5, -1, 0.5)$ and a mean squared error of $(0.25 + 1 + 0.25)/3 = 0.5$; no other line achieves a smaller value.

1.5 Calculus review

Linear algebra describes the data of a model; calculus describes how the objective responds to a change in the decision. The single idea of this subsection is that a differentiable function looks linear at close range, so that its slope says which way to move; every algorithm from [Week 6](#) onwards is built on it.

1.5.1 Slope and the tangent-line approximation

For a function f of one variable, the derivative $f'(x)$ measures the local slope at x . If $f'(x) > 0$ the function is increasing near x ; if $f'(x) < 0$ it is decreasing; and if $f'(x) = 0$ the tangent is flat, which makes x a *candidate* for a minimiser or maximiser. It is only a candidate: the nonconvex curve of [Figure 1.2](#) has three flat points, two valley bottoms and the hilltop between them. The slope also predicts nearby values, through the **tangent-line approximation**

$$f(x + \Delta) \approx f(x) + f'(x) \Delta, \quad (1.3)$$

in which Δ is a small displacement and $f'(x)\Delta$ is the predicted change ([Figure 1.3\(a\)](#)). For $f(x) = x^2$ at $x = 2$ we have $f'(2) = 4$, and with $\Delta = -0.1$ the prediction is $f(1.9) \approx 4 + 4(-0.1) = 3.6$ against the true value $1.9^2 = 3.61$. The error is exactly $\Delta^2 = 0.01$, because $(x + \Delta)^2 = x^2 + 2x\Delta + \Delta^2$. With $\Delta = -1$ the prediction is 0 and the truth is 1: the approximation is accurate for small Δ and drifts as Δ grows. This is the first optimisation idea: if $f'(x) < 0$ a small step to the right is predicted to decrease f , and if $f'(x) > 0$ a small step to the left. The local slope decides which way to move, and the quality of the approximation limits how far.

1.5.2 Several variables: contours and level sets

With several variables we write $f(x_1, \dots, x_d)$ or simply $f(x)$ with $x \in \mathbb{R}^d$. For $d = 2$ the graph of f is a surface, which is awkward to draw and to read. A **contour plot** instead shows the curves along which f takes equal values, as a topographic map shows lines of equal altitude. The set $\{x : f(x) = c\}$ of points with the same value c is a **level set** of f , and the set $\{x : f(x) \leq c\}$ of points with value at most c is a **sublevel set**. For $f(x) = x_1^2 + 2x_2^2$ the level sets are the ellipses $x_1^2 + 2x_2^2 = c$, nested around the minimiser at the origin as in [Figure 1.3\(b\)](#), and the sublevel sets are the same ellipses filled in; the contours of the linear objective of [Example 1.6](#) were parallel lines.

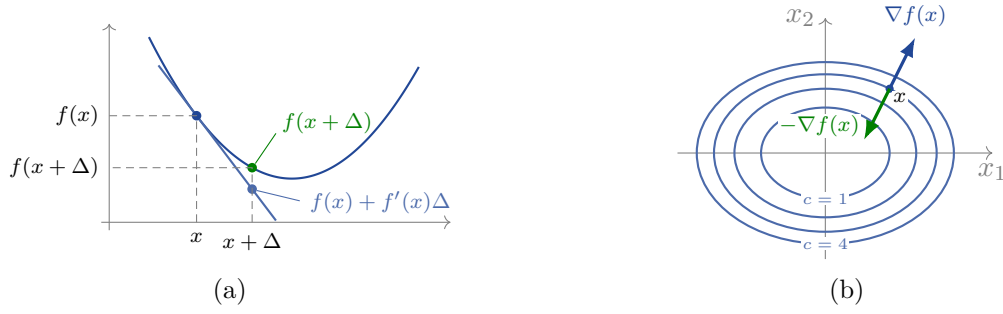


Figure 1.3: (a) The tangent at x , with slope $f'(x)$, predicts the value $f(x) + f'(x)\Delta$ at $x + \Delta$, slightly below the true value $f(x + \Delta)$; the gap grows with Δ . (b) The contours $x_1^2 + 2x_2^2 = c$, $c = 1, 2, 3, 4$, of the function of [Example 1.16](#), whose value increases outwards. The gradient $\nabla f(x) = (2, 4)$ at $x = (1, 1)$ is perpendicular to the contour $c = 3$ through x and points towards higher values; $-\nabla f(x)$ points downhill.

1.5.3 The gradient and the directional derivative

The one-variable slope must now become a vector, with one rate of change per coordinate.

Definition 1.15 (Gradient and directional derivative). Let $f : \mathbb{R}^d \rightarrow \mathbb{R}$ be differentiable. The **gradient** of f at x is the vector of partial derivatives

$$\nabla f(x) = \begin{bmatrix} \partial f / \partial x_1 \\ \vdots \\ \partial f / \partial x_d \end{bmatrix} \in \mathbb{R}^d,$$

where $\partial f / \partial x_i$ is the derivative of f with respect to x_i with the other components held fixed. The **directional derivative** of f at x along a direction $v \in \mathbb{R}^d$ is $Df(x; v) = \nabla f(x)^\top v$. It is the rate of change of f when moving from x along v , that is, the derivative of $t \mapsto f(x + tv)$ at $t = 0$.

The gradient is the several-variable slope, and it turns (1.3) into the **first-order approximation**

$$f(x + \Delta) \approx f(x) + \nabla f(x)^\top \Delta, \quad (1.4)$$

in which the displacement Δ is now a vector in $\mathbb{R}^d$ and the inner product $\nabla f(x)^\top \Delta$ predicts the local change. As in one variable, the approximation is accurate for small $\|\Delta\|_2$ and drifts as the displacement grows; [Week 6](#) bounds the gap. The formula supports the convexity test of [Week 3](#), the optimality conditions of [Week 5](#) and every gradient method thereafter.

Which direction increases f fastest? Among unit vectors v , the Cauchy–Schwarz inequality ([Proposition 1.11](#)) gives $Df(x; v) = \nabla f(x)^\top v \leq \|\nabla f(x)\|_2 \|v\|_2 = \|\nabla f(x)\|_2$, with equality (when $\nabla f(x) \neq 0$) exactly for $v = \nabla f(x) / \|\nabla f(x)\|_2$. Hence $\nabla f(x)$ points in the direction of steepest local increase, and $-\nabla f(x)$ is the direction of steepest decrease, the most natural direction in which to search for a minimiser. A direction with $\nabla f(x)^\top v = 0$ does not change f to first order: it runs along the contour through x . This is why the gradient is perpendicular to the contours in [Figure 1.3\(b\)](#).

Example 1.16 (Gradient, prediction and directions). Let $f(x_1, x_2) = x_1^2 + 2x_2^2$. The partial derivatives are $2x_1$ and $4x_2$, so $\nabla f(x) = (2x_1, 4x_2)$ and $\nabla f(1, 1) = (2, 4)$. For the displacement $\Delta = (-0.1, 0)$ the predicted change is $\nabla f(1, 1)^\top \Delta = 2(-0.1) + 4(0) = -0.2$, so (1.4) predicts $f(0.9, 1) \approx 3 - 0.2 = 2.8$; the true value is $0.81 + 2 = 2.81$. The steepest rate of increase at $(1, 1)$ is $\|(2, 4)\|_2 = \sqrt{20} \approx 4.47$. Along the unit vector $v = (2, -1)/\sqrt{5}$ the directional derivative is $(2 \cdot 2 - 4 \cdot 1)/\sqrt{5} = 0$: this direction is tangent to the ellipse $x_1^2 + 2x_2^2 = 3$ through $(1, 1)$.

1.5.4 Differentiation rules and the multivariate chain rule

Gradients are computed with the familiar rules for derivatives of one variable, applied one component at a time. For differentiable f, g and a constant α ,

$$\begin{array}{ll} \text{Sum} & (f + g)' = f' + g' \\ \text{Scaling} & (\alpha f)' = \alpha f' \\ \text{Chain} & (f(g(x)))' = f'(g(x)) g'(x) \end{array} \quad \begin{array}{ll} \text{Product} & (fg)' = f'g + fg' \\ \text{Quotient} & (f/g)' = (f'g - fg')/g^2 \end{array}$$

together with the common derivatives $(x^n)' = nx^{n-1}$, $(e^x)' = e^x$ and $(\log x)' = 1/x$. The chain rule does most of the work in this book. For the squared error of one record with one feature, $h(x) = (b - ax)^2$, it gives $h'(x) = 2(b - ax) \cdot (-a)$, the residual multiplied by the feature, and the multivariate chain rule says the same for a whole data set. It needs one new object: for a vector-valued map $r : \mathbb{R}^d \rightarrow \mathbb{R}^m$ with components $r_1, \dots, r_m$, the **Jacobian** $J_r(x) \in \mathbb{R}^{m \times d}$ is the matrix whose i th row is $\nabla r_i(x)^\top$.

Proposition 1.17 (Multivariate chain rule). *Let $r : \mathbb{R}^d \rightarrow \mathbb{R}^m$ and $f : \mathbb{R}^m \rightarrow \mathbb{R}$ be differentiable. The composite function $x \mapsto f(r(x))$ has gradient*

$$\nabla_x f(r(x)) = J_r(x)^\top \nabla f(r(x)),$$

where ∇_x indicates differentiation with respect to x . In particular, an affine map $r(x) = Ax + b$ (a linear map plus a constant vector) has $J_r(x) = A$, so $\nabla_x f(Ax + b) = A^\top \nabla f(Ax + b)$, and for least squares

$$\nabla_x \frac{1}{2} \|Ax - b\|_2^2 = A^\top (Ax - b). \quad (1.5)$$

Proof sketch. Fix k and move x by a small t along the k th coordinate, to $x + te_k$ with e_k the k th unit coordinate vector. The vector r moves by approximately $t \partial r(x) / \partial x_k$, the vector of the numbers $\partial r_i / \partial x_k$, and applying (1.4) to f at $r(x)$ with this displacement gives $f(r(x + te_k)) \approx f(r(x)) + t \nabla f(r(x))^\top \partial r(x) / \partial x_k$; hence $\partial f(r(x)) / \partial x_k = \sum_{i=1}^m (\partial f / \partial r_i) (\partial r_i / \partial x_k)$ (a full proof keeps track of the approximation errors). The numbers $\partial r_i / \partial x_k$, $i = 1, \dots, m$, form the k th column of $J_r(x)$, so the sum is the k th component of $J_r(x)^\top \nabla f(r(x))$. The i th component of $Ax + b$ is $A_{i:}x + b_i$ with gradient $A_{i:}^\top$, which gives $J_r = A$. For (1.5) take $r(x) = Ax - b$ and the outer function $f(r) = \frac{1}{2} \|r\|_2^2 = \frac{1}{2} \sum_i r_i^2$, whose gradient is $\nabla f(r) = r$. $\square$

The dimensions confirm the formula: $J_r(x)^\top$ is $d \times m$ and $\nabla f(r(x))$ has m components, so the product has one component per decision variable. Transposing the Jacobian pulls a gradient computed downstream back to the input x , as in the gradients of [Weeks 3, 7](#) and [9](#).

Example 1.18 (A Jacobian). Let $r : \mathbb{R}^2 \rightarrow \mathbb{R}^3$ be $r(x) = (x_1^2, x_1 x_2, x_2^2)$. Its rows of partial derivatives give

$$J_r(x) = \begin{pmatrix} 2x_1 & 0 \\ x_2 & x_1 \\ 0 & 2x_2 \end{pmatrix}, \quad J_r(1, 2) = \begin{pmatrix} 2 & 0 \\ 2 & 1 \\ 0 & 4 \end{pmatrix},$$

a 3×2 matrix, one row per output and one column per input.

Formula (1.5) closes the loop opened in [Section 1.4](#). At a minimiser of $\frac{1}{2} \|Ax - b\|_2^2$ no direction can decrease the function to first order, so the gradient must vanish: $A^\top (Ax - b) = 0$, which is precisely the normal equations (1.2). In [Example 1.14](#), indeed, $A^\top (Ax - b) = A^\top (0.5, -1, 0.5) = (0, 0)$ at $x = (0.5, 0.5)$. That a vanishing gradient is also *sufficient* for a minimiser here follows from convexity ([Theorem 3.10](#)). For most problems $\nabla f(x) = 0$ cannot be solved by a formula, and [Week 6](#) replaces the formula by an iteration that steps repeatedly along $-\nabla f(x)$.

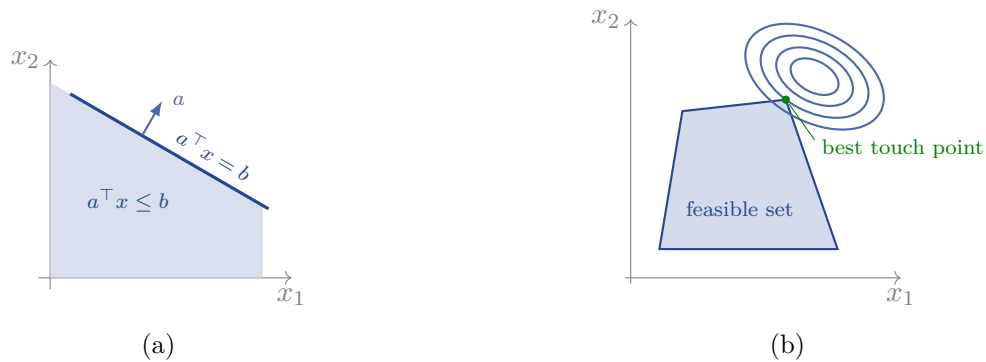


Figure 1.4: (a) The hyperplane $a^T x = b$ and the halfspace $a^T x \leq b$ (shaded); the normal vector a points out of the halfspace. (b) Contours of an objective whose unconstrained minimiser lies outside the feasible polygon. The lowest contour that still touches the feasible set meets it at a corner, which is the constrained optimum.

1.5.5 Affine functions, halfspaces and the geometry of constrained problems

Every constraint of this week's linear models has the same shape: a linear function of the decision compared with a constant.

Definition 1.19 (Affine function, hyperplane, halfspace). An **affine function** has the form $x \mapsto a^T x + b$ with $a \in \mathbb{R}^d$ and $b \in \mathbb{R}$: a linear function plus a constant. For $a \neq 0$ and $b \in \mathbb{R}$, the set $\{x : a^T x = b\}$ is a **hyperplane** and the set $\{x : a^T x \leq b\}$ is a **halfspace**. An intersection of finitely many halfspaces and hyperplanes is a **polyhedral set**.

In the plane a hyperplane is a line and a halfspace is one side of it (Figure 1.4(a)); in the language of Definition 1.4, the halfspace is the constraint $g(x) \leq 0$ for the affine function $g(x) = a^T x - b$. The vector a , the gradient of $a^T x$, is perpendicular to the hyperplane and points out of the halfspace. A linear equality constraint confines the decision to a hyperplane, a linear inequality to a halfspace, and a system $Ax \leq b$ to an intersection of m halfspaces: the pentagon of Example 1.6 is the intersection of five.

Drawing a feasible set on top of a contour plot shows what a constrained problem asks: we search only the feasible set, and geometrically we look for the *best contour that still touches the feasible region*. In Figure 1.4(b) the objective is smallest at the centre of the ellipses, which is infeasible; the two innermost contours miss the region altogether, and the first contour to reach it touches it at a single corner, the optimal solution. This is why boundary points matter: when the bottom of a bowl-shaped objective is infeasible, the constrained optimum lies on the boundary of the feasible set, where some constraints hold with equality. Week 5 turns this picture into optimality conditions.

1.5.6 Local versus global minima

With norms in hand, the two notions used informally in Section 1.3 can be made precise.

Definition 1.20 (Local and global minimum). Consider minimising f over a feasible set $\mathcal{X}$. A point $\bar{x} \in \mathcal{X}$ is a **global minimum** if $f(\bar{x}) \leq f(x)$ for all $x \in \mathcal{X}$; this is the same as an optimal solution. It is a **local minimum** if there is a radius $r > 0$ such that $f(\bar{x}) \leq f(x)$ for all $x \in \mathcal{X}$ with $\|x - \bar{x}\|_2 \leq r$, that is, if it is best within some neighbourhood. Local and global maxima are defined with the inequalities reversed. When f and $\mathcal{X}$ are convex the two notions coincide, every local minimum being global (Theorem 4.1); this is the central promise of Section 1.3.

Every global minimum is a local minimum, but not conversely. The nonconvex curve of Figure 1.2 has a local minimum near $x = -0.96$ (value 0.93) and its global minimum near

$x = 1.04$ (value 0.57). The derivative vanishes at both, so no test based on information at the point itself can tell them apart, and an analyst handed the left-hand point by an algorithm cannot know that a better decision exists. Convexity removes this ambiguity, and building it is the business of the coming weeks: [Week 2](#) begins with convex sets, [Week 3](#) continues with convex functions, and the promise itself is proved as [Theorem 4.1](#).

1.6 A first model in RSOME

1.6.1 The software stack

The book assumes basic Python fluency (functions and imports, notebooks, NumPy arrays and their shapes, slicing, broadcasting, matrix multiplication, simple plots) but no experience with optimisation software. All work is done inside one dedicated Python environment, here called `baopt`. NumPy is used for array computations and, from [Week 6](#), for the algorithms we implement ourselves. RSOME is the *modelling layer*: a Python package in which a model is typed almost as it is written on paper and which passes it to a solver. RSOME comes with a default solver built on SciPy; the commercial solvers Gurobi and MOSEK, free for academic use, handle larger problems and the richer model classes of [Week 4](#), and need a licence to be activated.¹ The environment is created once from a terminal:

```
conda create -n baopt python=3.12
conda activate baopt
pip install numpy scipy pandas matplotlib jupyterlab rsome
conda install -c gurobi gurobi          # Gurobi, or alternatively: pip install mosek
```

1.6.2 Building, solving and reading a model

The first model is a linear programme small enough to check by hand,

$$\begin{array}{ll} \max & 5x_1 + 4x_2 \\ \text{s.t.} & 6x_1 + 4x_2 \leq 24, \quad x_1 + 2x_2 \leq 6, \quad x_1, x_2 \geq 0, \end{array} \quad (1.6)$$

and the code of [Listing 1.1](#) mirrors it line by line.

Listing 1.1: A first RSOME model: build, solve, read the status and the solution.

```
from rsome import ro
from rsome import grb_solver as grb # needs Gurobi installed
m = ro.Model()                      # create an empty model
x = m.dvar(2)                       # two decision variables x[0], x[1]
m.max(5*x[0] + 4*x[1])              # objective
m.st(6*x[0] + 4*x[1] <= 24,         # constraints
     x[0] + 2*x[1] <= 6,
     x >= 0)
m.solve(grb)                        # solve with Gurobi
print(m.solution.status)            # solver status
print(m.get())                      # optimal value
print(x.get())                      # optimal solution
```

The call `ro.Model()` creates the model, `m.dvar(2)` declares a vector of two decision variables (indexed from zero), `m.max` or `m.min` sets the objective, and `m.st` adds any number of constraints; `x >= 0` applies to every component. After a short solver log, the three `print` statements produce

¹Official instructions: RSOME, https://xiongpengnus.github.io/rsome/get_start; Gurobi for Python, <https://support.gurobi.com/hc/en-us/articles/360044290292-How-do-I-install-Gurobi-for-Python>; MOSEK, <https://docs.mosek.com/latest/install/installation.html> and academic licences at <https://www.mosek.com/products/academic-licenses/>.

2, 21.0 and [3. 1.5]: the status (Gurobi’s code for optimal), the optimal value $z^* = 21$ and the optimal solution $x^* = (3, 1.5)$. The answer can be confirmed with the method of [Example 1.6](#): the feasible set has corners $(0, 0)$, $(4, 0)$, $(3, 1.5)$ and $(0, 3)$ with objective values 0, 20, 21 and 12, and at x^* both resource constraints hold with equality.

The status is the first thing to read, because it says which outcome of [Definition 1.7](#) occurred, but its encoding depends on the solver. For an optimal problem and for the infeasible and unbounded problems of [Section 1.2.3](#), the default solver reports 0, 2 and 3; Gurobi reports 2, 4 and 4 (code 4 means “infeasible or unbounded”); MOSEK reports ‘Optimal’, ‘Undefined’ and ‘Certificate’. When the status is not optimal there is no solution to read, and `m.get()` raises the error “No solution available”.

1.6.3 Same model, different solver

Switching solvers changes one line: `m.solve()` uses RSOME’s default solver, and importing `msk_solver` as `msk` and calling `m.solve(msk)` uses MOSEK. Import only the interfaces of solvers you have installed: importing `grb_solver` fails without Gurobi. All three solvers return $x^* = (3, 1.5)$ and $z^* = 21$. The formulation stays fixed while the solver is exchanged (later weeks check our own algorithms this way), and the habit for the rest of the book is set: formulate clearly, solve appropriately, read the status, and only then report the decision and its objective value. A first solve that fails is usually a setup error: the wrong environment, a different Python interpreter from the one where the packages were installed, or an unactivated licence.

1.7 Summary and looking ahead

A business decision is ready to hand to a solver when it has been written as a model with four elements (decision variables, data, objective and constraints), so that its assumptions are explicit, and when the model has structure that an algorithm can exploit. The three running models illustrated the first step, and the vocabulary of [Section 1.2](#) is the language in which every later result is stated. On the second step the news is mixed: general problems cannot be solved reliably, because a method that looks around locally can be trapped in a local minimum, whereas for convex problems every local minimum is global. The review subsections supplied the tools for making this precise: norms, suprema and infima, Cauchy–Schwarz and linear independence; the gradient, the first-order approximation and the chain rule; and contours touching a feasible set. A model typed in RSOME runs on any installed solver, and its status is read first. The missing ingredient is a precise account of convexity, and it starts with feasible sets: [Week 2](#) asks which feasible sets can be trusted and shows that the usual modelling operations preserve convexity.

1.8 Exercises

Exercise 1.1 (Vocabulary). A retailer allocates weekly online advertising spend x_1, x_2, x_3 across search, social and video to maximise predicted conversions; total spend cannot exceed \$10,000, search must receive at least \$2,000, social cannot exceed \$4,000, and all spends are nonnegative. Identify (a) the decision variables and (b) the parameters or data; (c) state a plausible objective; (d) write the constraints; (e) describe the feasible set in words.

Exercise 1.2 (Feasibility). Which of the candidates $(2000, 4000, 3000)$, $(1000, 4000, 3000)$ and $(5000, 4000, 3000)$ are feasible for [Model 1.1](#)? Name a violated constraint where there is one, and compute the predicted conversions of each feasible candidate.

Exercise 1.3 (Supremum and infimum). For each of the sets $[0, 5)$, $\{1/n : n = 1, 2, \dots\}$ and $[0, \infty)$, give the supremum and infimum, and state whether the maximum and minimum exist.

Exercise 1.4 (Status reasoning). For each problem, say which status you expect and give a one-sentence business interpretation (recommendation available, conflicting rules, or missing cap). Case A: $\max x_1 + x_2$ subject to $x_1 + x_2 \leq 10$, $x_1, x_2 \geq 0$. Case B: $\max x_1$ subject to $x_1 \leq 1$, $x_1 \geq 3$. Case C: $\max x_1$ subject to $x_1 \geq 0$. In Case A, is the optimal solution unique?

Exercise 1.5 (Derivatives and a first-order prediction). (a) Differentiate $x^2 e^x$, $\log(1 + e^x)$, $x/(1 + x^2)$ and $(3x - 2)^4$, naming the rule used. (b) Use (1.4) for $f(x_1, x_2) = x_1^2 + 2x_2^2$ at $(1, 1)$ to predict $f(0.9, 0.9)$, and compare with the exact value.

Exercise 1.6 (First solve and solver switch). Build [Model 1.1](#) in RSOME with the default solver; report the status, optimal value and decision; then re-solve it with Gurobi or MOSEK.

Exercise 1.7 (Contours over a feasible region). With NumPy and Matplotlib, draw the feasible set of [Example 1.6](#) with several contours of its objective, marking the best touching point.

Week 2 Convex Sets, Convex Cones, and Operations That Preserve Convexity

Week 1 ended with a promise: when a *convex* objective is minimised over a *convex* feasible set (a maximisation being rewritten as the minimisation of $-f$), a point that cannot be improved locally is a global optimum, so the answer returned by a solver can be trusted (Definition 1.20). The promise has two halves, one about sets and one about functions. This week takes the first half and asks: *which feasible sets can we safely optimise over?* The answer is the convex ones, and the work of the week is to make that word precise and, above all, easy to check.

We stay inside the workflow of Week 1 (business question, model, solve, recommendation) and reuse its vocabulary of feasible and optimal solutions and of infeasible and unbounded problems (Section 1.2), but we look only at the constraint side of a model: what shape do the constraints carve out of $\mathbb{R}^d$, and when is that shape convex? Three sets, each taken from an application of Week 1, accompany us. The long-only, fully invested portfolio weights of Model 1.2, $w \geq 0$ and $\mathbf{1}^\top w = 1$, where $\mathbf{1}$ is the vector of ones so that $\mathbf{1}^\top w$ is the sum of the weights, form a *simplex*. The budget and policy rules of the media-mix model (Model 1.1) are linear inequalities, and together they form a *polyhedron*. And a constraint $\|\beta\|_2 \leq 1$, which one could add to the least-squares problem of Model 1.3 to limit the size of the coefficients, is a *Euclidean ball*, a first glimpse of the regularisation studied in Week 9. For now these are only names; all three sets are defined in Section 2.2.

Section 2.1 builds the definitions from a single template, the weighted combination $\theta_1 x_1 + \dots + \theta_k x_k$, whose four weight rules produce subspaces, affine sets, convex cones and convex sets. Section 2.2 collects the standard convex sets that models use again and again. Section 2.3 gives the operations that are guaranteed to preserve convexity, so that a feasible region (the feasible set $\mathcal{X}$ of Definition 1.4; the two names are used interchangeably) assembled from standard pieces is convex *by construction*, and applies them to a media-mix model in which the budget is itself a decision. Section 2.4 lists the patterns that break convexity and condenses the week into a three-question checklist.

2.1 Convex combinations and convex sets

2.1.1 Line segments and the definition of a convex set

Take two points $x_1, x_2 \in \mathbb{R}^d$; think of two candidate plans, each a complete vector of decisions. The most natural way to produce a third plan from them is to blend them, taking a fraction θ of the first and the remaining fraction $1 - \theta$ of the second.

Definition 2.1 (Convex combination, convex set). A **convex combination** of two points $x_1, x_2 \in \mathbb{R}^d$ is a point of the form

$$\theta x_1 + (1 - \theta)x_2, \quad 0 \leq \theta \leq 1.$$

The set of all convex combinations of x_1 and x_2 is the **line segment** between them. A set $C \subseteq \mathbb{R}^d$ is **convex** if it contains every convex combination of its points:

$$x_1, x_2 \in C \text{ and } 0 \leq \theta \leq 1 \implies \theta x_1 + (1 - \theta)x_2 \in C.$$

Writing the combination as $x_2 + \theta(x_1 - x_2)$ shows why it traces a segment: we start at x_2 when $\theta = 0$ and move in the direction $x_1 - x_2$ until we reach x_1 at $\theta = 1$; the value $\theta = \frac{1}{2}$ gives the midpoint (Figure 2.1(a)). Coordinate by coordinate, a convex combination is a weighted average, which gives the definition its practical reading.

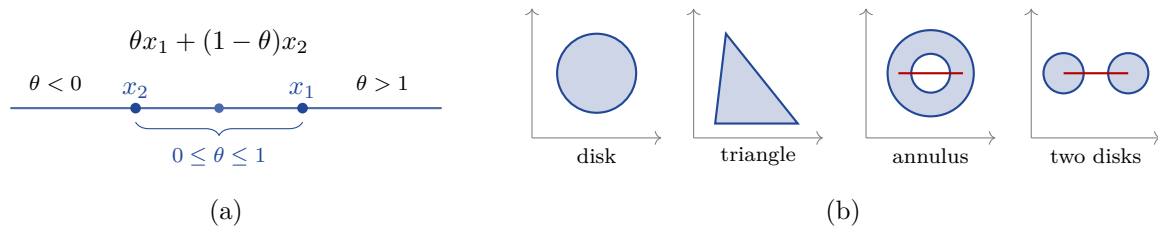


Figure 2.1: (a) The points $\theta x_1 + (1 - \theta)x_2$: weights $0 \leq \theta \leq 1$ give the segment between x_1 and x_2 (convex combinations); letting θ range over all of $\mathbb{R}$ gives the whole line (affine combinations, [Section 2.1.5](#)). (b) A disk and a triangle are convex; an annulus and a union of two disjoint disks are not, because the red segment joins two points of the set but leaves it.

Key idea

A feasible set is convex when *averaging two feasible plans always gives a feasible plan*: between any two of its points, the set contains the whole segment.

If two media plans both respect the budget and every policy rule, then so does the plan that spends, channel by channel, 30% of the first plus 70% of the second. A word on notation: in this definition, and in the other definitions of this subsection, $x_1, x_2, \dots$ are *points* of $\mathbb{R}^d$, not the coordinates of one point. In proofs we write x, y for two points, and whenever x_1, x_2 are the coordinates of a point x in the plane we say so.

Note on the lecture slides

The slides use x_1, x_2 in two ways: as two *points* of $\mathbb{R}^d$ in the definitions of convex, conic and affine combinations (frames 2.4, 2.7–2.10), and as the two *coordinates* of one point in the classification table, the RSOME model and the tutorial (frames 2.12, 2.26, 2.32–2.33). The affine-set figure on frame 2.10 even labels both its axes and its two marked points x_1, x_2 . The notes keep the slides' symbols but always say which reading is meant.

Example 2.2 (Convex and nonconvex sets). Intervals of the real line, disks, triangles and rectangles are convex: the segment between two of their points never leaves them ([Figure 2.1\(b\)](#)). The empty set, a single point and $\mathbb{R}^d$ itself are convex too. Four patterns are typically *not* convex. In an annulus, two points on opposite sides of the hole have their midpoint in the hole. In a union of two disjoint pieces, the segment between a point of each piece crosses the gap. An integer-only set such as $\{0, 1\} \subset \mathbb{R}$ contains 0 and 1 but not 0.5. An either-or rule such as “a channel is switched off or receives at least \$2000”, that is, $\{x \in \mathbb{R} \mid x = 0 \text{ or } x \geq 2000\}$, contains 0 and 2000 but not 1000. Convexity is a statement about *all* pairs of points: proving it needs an argument for arbitrary x, y and θ , whereas disproving it needs only one violating pair, which is the standard recipe for showing that a set is not convex.

2.1.2 Verifying convexity from the definition: the halfspace

Recall from [Week 1](#) that a linear inequality $a^\top z \leq b$, with a nonzero $a \in \mathbb{R}^d$ and $b \in \mathbb{R}$, defines a *halfspace*. It is the simplest constraint of all, and we verify its convexity directly from [Definition 2.1](#).

Proposition 2.3 (Halfspaces are convex). *The set $H = \{z \in \mathbb{R}^d \mid a^\top z \leq b\}$ is convex.*

Proof. Take any $x, y \in H$ and any $0 \leq \theta \leq 1$. Then

$$a^\top (\theta x + (1 - \theta)y) = \theta a^\top x + (1 - \theta) a^\top y \leq \theta b + (1 - \theta)b = b.$$

The equality is the linearity of the inner product. The inequality multiplies $a^\top x \leq b$ by $\theta \geq 0$ and $a^\top y \leq b$ by $1 - \theta \geq 0$ and adds the results. Hence $\theta x + (1 - \theta)y \in H$. $\square$

The proof shows where the restriction $0 \leq \theta \leq 1$ matters: for θ outside $[0, 1]$ one of the two weights is negative, the corresponding inequality flips, and the argument fails (a halfspace does not contain the whole line through two of its points; see [Section 2.1.5](#)). The same three lines work for $a^\top z \geq b$ and for the equation $a^\top z = b$. This is a useful proof to have done once; it is not how convexity is established in practice, where the skill (from [Section 2.2](#) onwards) is to recognise standard convex pieces and safe ways of combining them.

2.1.3 Convex combinations of many points and the convex hull

Definition 2.4 (Convex combination of k points, convex hull). A **convex combination** of points $x_1, \dots, x_k \in \mathbb{R}^d$ is a point of the form

$$\theta_1 x_1 + \dots + \theta_k x_k, \quad \theta_i \geq 0 \quad (i = 1, \dots, k), \quad \theta_1 + \dots + \theta_k = 1.$$

The **convex hull** of a set $S \subseteq \mathbb{R}^d$, written $\text{conv } S$, is the set of all convex combinations of (finitely many) points of S .

For $k = 2$ this is [Definition 2.1](#) with $\theta_1 = \theta$ and $\theta_2 = 1 - \theta$. The weights behave like probabilities: a convex combination is the mean of a random point that equals x_i with probability θ_i , and the next result says that the two-point definition already covers every such average.

Proposition 2.5. *A convex set C contains every convex combination of finitely many of its points.*

Proof. We use induction on k ; the case $k = 2$ is the definition. Let $x = \sum_{i=1}^k \theta_i x_i$ with all $x_i \in C$. If $\theta_k = 1$ then $x = x_k \in C$. Otherwise $y = \sum_{i=1}^{k-1} \frac{\theta_i}{1-\theta_k} x_i$ is a convex combination of $k-1$ points of C (its weights are nonnegative and sum to $(1-\theta_k)/(1-\theta_k) = 1$), so $y \in C$ by the induction hypothesis, and $x = (1-\theta_k)y + \theta_k x_k \in C$ by the two-point definition. $\square$

Consequently every convex set containing S contains $\text{conv } S$, which is itself convex: the convex hull is the *smallest* convex set containing S . Geometrically, we fill in the gaps until the set becomes convex: the hull of finitely many points in the plane is the polygon spanned by the outermost ones ([Figure 2.2\(a\)](#)), and the hull of the annulus in [Figure 2.1\(b\)](#) is the full disk.

2.1.4 Conic combinations and convex cones

Convex combinations keep the total weight fixed at one. In many models the total is not fixed: a plan may be scaled up or down. Dropping the sum condition, and keeping only the sign condition, gives the second kind of combination.

Definition 2.6 (Conic combination, convex cone). A **conic combination** of points $x_1, \dots, x_k \in \mathbb{R}^d$ is a point of the form $\theta_1 x_1 + \dots + \theta_k x_k$ with $\theta_i \geq 0$ for all i (the weights need not sum to one). A set $K \subseteq \mathbb{R}^d$ is a **convex cone** if it contains every conic combination of its points.

In [Figure 2.2\(b\)](#), $0.5u + 0.3v = (2, 0.55) + (0.39, 0.96) = (2.39, 1.51)$ is a conic combination of u and v but not a convex one, since the weights sum to 0.8. The definition bundles two operations, scaling and adding, and it is convenient to be able to test them separately.

Proposition 2.7 (Working tests for a convex cone). *For a nonempty set $K \subseteq \mathbb{R}^d$ the following statements are equivalent:*

- (a) K is a convex cone;
- (b) K is convex, and $\theta x \in K$ for all $x \in K$ and all $\theta \geq 0$;
- (c) $\theta x \in K$ and $x + y \in K$ for all $x, y \in K$ and all $\theta \geq 0$.

In each case $0 \in K$.

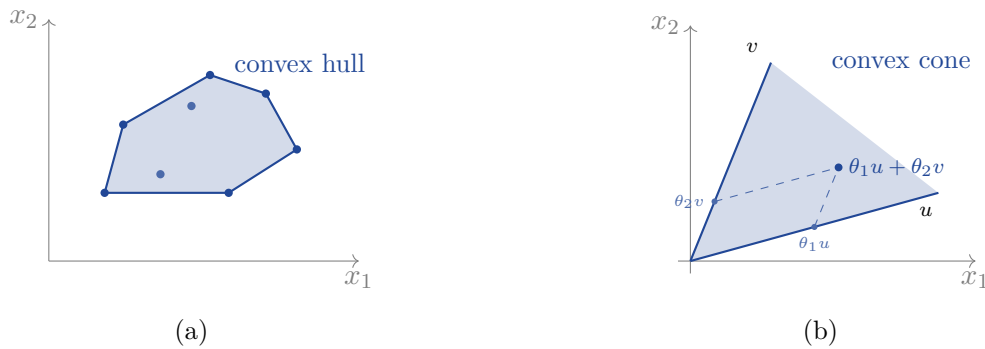


Figure 2.2: (a) The convex hull of eight points in the plane (axes are the coordinates x_1, x_2) is the polygon spanned by the six outer points; the two inner points are already convex combinations of the others. (b) The conic combinations (Definition 2.6) of $u = (4, 1.1)$ and $v = (1.3, 3.2)$ fill the wedge between the two rays, a convex cone that continues without bound; the marked point $0.5u + 0.3v = (2.39, 1.51)$ is obtained by the parallelogram rule.

Proof. (a) $\Rightarrow$ (b): a convex combination is a conic combination, and so is θx (take $k = 1$). (b) $\Rightarrow$ (c): $x + y = 2(\frac{1}{2}x + \frac{1}{2}y)$ is a nonnegative multiple of a midpoint. (c) $\Rightarrow$ (a): each $\theta_i x_i$ lies in K by scaling, and their sum lies in K by repeated addition. Finally $0 = 0 \cdot x \in K$. $\square$

Test (b) separates two ideas that are easy to confuse: *convexity is about averaging, conic structure is about nonnegative scaling*. Geometrically, a convex cone is a convex set that contains, with each point x , the whole ray $\{\theta x \mid \theta \geq 0\}$ from the origin through x ; in business terms, a feasible plan stays feasible when it is scaled up or down. The basic examples are the nonnegative orthant $\mathbb{R}_+^d = \{x \in \mathbb{R}^d \mid x \geq 0\}$ (scaling and adding nonnegative vectors keeps them nonnegative), a single ray from the origin, and the wedge of Figure 2.2(b); the norm cone joins them in Section 2.2.4.

Note on the lecture slides

Frame 2.12 and tutorial question 1 ask whether a set is a “cone”, but the slides define only a “convex cone” (frame 2.8). In this week “cone” always means convex cone, and Proposition 2.7(b) is the intended test: convex, and closed under nonnegative scaling. For every set in those questions the answer is the same whether one tests scaling alone or the full definition.

2.1.5 Affine combinations and affine sets

Now keep the rule that the weights sum to one but drop the sign condition.

Definition 2.8 (Affine combination, affine set). An **affine combination** of two points $x_1, x_2 \in \mathbb{R}^d$ is a point $\theta x_1 + (1 - \theta)x_2$ with $\theta \in \mathbb{R}$; more generally, an affine combination of $x_1, \dots, x_k$ is $\theta_1 x_1 + \dots + \theta_k x_k$ with $\theta_1 + \dots + \theta_k = 1$ and weights of any sign. A set $C \subseteq \mathbb{R}^d$ is **affine** if $\theta x_1 + (1 - \theta)x_2 \in C$ for every $x_1, x_2 \in C$ and every $\theta \in \mathbb{R}$.

The affine combinations of two distinct points form the whole *line* through them; for $\theta < 0$ or $\theta > 1$ we extrapolate beyond the segment (Figure 2.1(a)). An affine set thus contains the entire line through any two of its points, and since this includes the segment, *every affine set is convex*. Points, lines, planes and $\mathbb{R}^d$ are affine sets. The example that matters for modelling is the solution set of a system of linear equations, $\{x \in \mathbb{R}^d \mid Ax = b\}$: if $Ax_1 = b$ and $Ax_2 = b$, then for every $\theta \in \mathbb{R}$

$$A(\theta x_1 + (1 - \theta)x_2) = \theta Ax_1 + (1 - \theta)Ax_2 = \theta b + (1 - \theta)b = b.$$

Only the fact that the weights sum to one was used; no sign was needed, in contrast to the halfspace proof. In a model, such equalities are accounting identities, conservation equations (what flows in equals what flows out) and balance constraints such as “fully invested”, $\mathbf{1}^\top w = 1$. They survive extrapolation, whereas an inequality such as $w \geq 0$ does not, which is why the simplex (in two or more dimensions) will turn out to be convex but not affine.

2.1.6 Four kinds of combinations, and a first classification exercise

All the definitions of this subsection use the same template $\theta_1 x_1 + \cdots + \theta_k x_k$ and differ only in the rule imposed on the weights; Table 2.1 lists the four possibilities, the first being the linear combinations of Week 1. An affine set is likewise closed under affine combinations of any number of its points (the induction of Proposition 2.5 goes through after splitting off a weight $\theta_i \neq 1$, which exists because the weights sum to one).

Table 2.1: One template, four weight rules. Each rule defines a kind of combination and the kind of set that is closed under it.

Combination	Restriction on the weights	Set closed under it
Linear	none	subspace (span)
Affine	$\sum_i \theta_i = 1$	affine set (point, line, plane)
Conic	$\theta_i \geq 0$	convex cone
Convex	$\theta_i \geq 0$ and $\sum_i \theta_i = 1$	convex set (segment, convex hull)

Convex combinations obey both restrictions at once: they are exactly the combinations that are both affine and conic. A set that is closed under a larger family of combinations is in particular closed under a smaller one, so every affine set and every convex cone is convex. In the first row, a *subspace* is a set closed under all linear combinations, such as a line or plane through the origin, and the *span* of some vectors is the set of all their linear combinations, the smallest subspace containing them; every subspace is both an affine set and a convex cone. The converse fails: a disk is convex but is neither affine nor a cone.

Example 2.9 (Convex, cone, both, or neither?). In this example $x = (x_1, x_2)$ is a point of the plane with coordinates x_1, x_2 .

- (i) $\{x \in \mathbb{R}^2 \mid x \geq 0, \mathbf{1}^\top x \leq 1\}$, the triangle with vertices $(0, 0)$, $(1, 0)$, $(0, 1)$, is convex: if x, y satisfy the inequalities then $\theta x + (1 - \theta)y \geq 0$ and $\mathbf{1}^\top(\theta x + (1 - \theta)y) \leq \theta + (1 - \theta) = 1$. It is not a cone: $(1, 0)$ belongs to it but $2 \cdot (1, 0) = (2, 0)$ does not.
- (ii) $\{x \in \mathbb{R}^2 \mid x \geq 0\} = \mathbb{R}_+^2$ is convex and closed under nonnegative scaling, hence a convex cone.
- (iii) The disk $\{x \in \mathbb{R}^2 \mid \|x\|_2 \leq 1\}$ is convex (Proposition 2.16) but not a cone, since $(1, 0)$ belongs to it and $(2, 0)$ does not: a ball has a fixed size.
- (iv) $\{x \in \mathbb{R}^2 \mid x_1 \in \{0, 1\}, 0 \leq x_2 \leq 1\}$ consists of two parallel vertical segments. It contains $(0, 0)$ and $(1, 0)$ but not their midpoint $(0.5, 0)$, so it is not convex and therefore not a convex cone either (also $2 \cdot (1, 1) = (2, 2)$ leaves the set). The discrete restriction on x_1 is what breaks convexity.

The four answers show that “convex” and “cone” are different properties: a budget-capped region allows averaging but not unlimited scaling.

2.2 A catalogue of standard convex sets

Feasible regions of real models are not checked point pair by point pair. They are assembled from a short list of standard pieces, and this subsection collects the list, in each case with the kind of business rule the piece expresses. All the pieces are convex. For some the proof is immediate

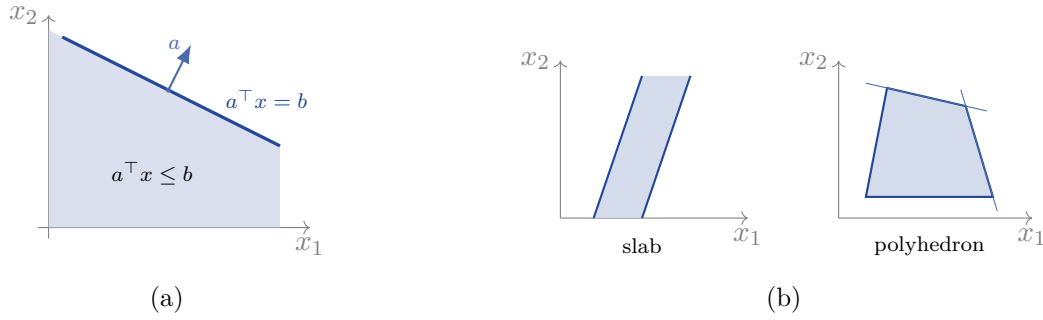


Figure 2.3: (a) The hyperplane $a^\top x = b$ (here, in coordinates x_1, x_2 , the line $x_1 + 2x_2 = 7.25$, so a is a multiple of $(1, 2)$) and the halfspace $a^\top x \leq b$; the normal vector a is perpendicular to the hyperplane and points away from the halfspace. (b) A slab is the band between two parallel hyperplanes; a polyhedron is cut out by finitely many halfspaces, two of whose boundary lines are extended here.

from [Section 2.1](#); for those that are built by stacking several constraints it follows from the intersection rule proved in [Section 2.3](#).

2.2.1 Hyperplanes and halfspaces

We restate the two basic sets of [Week 1](#) so that the catalogue is complete.

Definition 2.10 (Hyperplane, halfspace). Let $a \in \mathbb{R}^d$ with $a \neq 0$, and let $b \in \mathbb{R}$. The set $\{x \in \mathbb{R}^d \mid a^\top x = b\}$ is a *hyperplane*, and the set $\{x \in \mathbb{R}^d \mid a^\top x \leq b\}$ is a *halfspace*.

These are the basic equality and inequality templates of every optimisation model. A hyperplane is a line in $\mathbb{R}^2$ and a plane in $\mathbb{R}^3$, and it cuts $\mathbb{R}^d$ into the two halfspaces $a^\top x \leq b$ and $a^\top x \geq b$. As in [Week 1](#), the vector a is the hyperplane's *normal*: $a^\top(x - y) = 0$ for any two points x, y of the hyperplane, and a points out of the halfspace $a^\top x \leq b$ ([Figure 2.3\(a\)](#)). The requirement $a \neq 0$ excludes the degenerate cases in which the two sets would be all of $\mathbb{R}^d$ or empty.

How do these two sets fit into [Table 2.1](#)? A hyperplane is the solution set of a linear equation, so it is affine ([Section 2.1.5](#)) and therefore convex. A halfspace is convex ([Proposition 2.3](#)) but not affine: the line through one of its points in the direction a eventually violates $a^\top x \leq b$. Whether either set is a convex cone depends on b alone. If $b = 0$, scaling by $\theta \geq 0$ preserves both $a^\top x = 0$ and $a^\top x \leq 0$, so by [Proposition 2.7\(b\)](#) both sets are convex cones. If $b \neq 0$, neither is: the hyperplane misses the origin; the halfspace misses the origin when $b < 0$; and when $b > 0$ the point $x = (b/\|a\|_2^2)a$ satisfies $a^\top x = b$ while $a^\top(2x) = 2b > b$. In modelling terms, a homogeneous rule such as “search receives at least as much as social” (in coordinates, $x_2 - x_1 \leq 0$) survives a rescaling of the whole plan, whereas a budget cap ($\mathbf{1}^\top x \leq 10\,000$) does not.

2.2.2 Slabs and polyhedra

A two-sided limit such as “social receives between 10% and 40% of the \$10 000 budget”, that is, $1000 \leq x_2 \leq 4000$ for the social coordinate x_2 , bounds one linear expression from both sides, and a complete model imposes many linear rules at once.

Definition 2.11 (Slab, polyhedron). A **slab** is a set $\{x \in \mathbb{R}^d \mid \alpha \leq a^\top x \leq \beta\}$ with $a \neq 0$ and scalars $\alpha \leq \beta$ (two numbers, unrelated to the coefficient vector β of the introduction). A **polyhedron** is the intersection of finitely many halfspaces and hyperplanes,

$$\{x \in \mathbb{R}^d \mid Ax \leq b, Cx = e\},$$

where $A \in \mathbb{R}^{m \times d}$, $b \in \mathbb{R}^m$, $C \in \mathbb{R}^{p \times d}$ and $e \in \mathbb{R}^p$.

(Week 1 called such sets polyhedral; the definition here adds the matrix notation.)

A slab is the intersection of the two halfspaces $a^\top x \leq \beta$ and $-a^\top x \leq -\alpha$: a “tolerance band” or “allowed range” (Figure 2.3(b)). In the polyhedron, the vector inequality $Ax \leq b$ is read row by row, as in Week 1: if $a_i^\top$ denotes the i th row of A , it says $a_i^\top x \leq b_i$ for $i = 1, \dots, m$, one halfspace per row, and each row of $Cx = e$ is one hyperplane. In this formula C is a *matrix*, not the generic convex set C of the other definitions; the context always tells which is meant. A polyhedron need not be bounded: a single halfspace and the nonnegative orthant are polyhedra. The feasible set of the media-mix model (Model 1.1) is a polyhedron in $\mathbb{R}^3$ with six inequality rows and no equality: the budget row $(1, 1, 1)$ with right-hand side 10 000, the policy rules written as $-x_1 \leq -2000$ and $x_2 \leq 4000$, and the three sign rows $-x_j \leq 0$. Since halfspaces and hyperplanes are convex and intersections of convex sets are convex (Proposition 2.17), *every polyhedron is convex*. Linear equalities and inequalities therefore always define convex feasible regions, which is why linear programming (Model 4.6) is fundamentally convex.

2.2.3 Boxes and the simplex

Two polyhedra are used so often that they have names of their own.

Definition 2.12 (Box, simplex). Given $\ell, u \in \mathbb{R}^d$ with $\ell \leq u$, the **box** with lower bounds ℓ and upper bounds u is $\{x \in \mathbb{R}^d \mid \ell \leq x \leq u\}$, the inequalities holding in every coordinate. The **simplex** in $\mathbb{R}^d$ is

$$\{x \in \mathbb{R}^d \mid x \geq 0, \mathbf{1}^\top x = 1\}.$$

A box is the intersection of $2d$ halfspaces, $\ell_j \leq x_j$ and $x_j \leq u_j$ for each coordinate j ; it models capacity limits, policy bounds and variable domains. The simplex is the intersection of the nonnegative orthant with one hyperplane and captures *nonnegative shares with a fixed total*: portfolio weights, budget shares, probabilities. In $\mathbb{R}^2$ it is the segment joining $(1, 0)$ and $(0, 1)$ (Figure 2.4(a)); in $\mathbb{R}^3$ it is the triangle with vertices the unit vectors $e_1 = (1, 0, 0)$, $e_2 = (0, 1, 0)$, $e_3 = (0, 0, 1)$. In general, every point of the simplex is $x = x_1 e_1 + \dots + x_d e_d$, a convex combination of the unit vectors with its own coordinates $x_j \geq 0$ (summing to one) as weights, so the simplex is the convex hull $\text{conv}\{e_1, \dots, e_d\}$ (Definition 2.4): the portfolios obtained by mixing the d single-asset portfolios. For $d \geq 2$ it is convex but neither affine nor a cone, because $x \geq 0$ does not survive extrapolation and $\mathbf{1}^\top x = 1$ does not survive scaling.

2.2.4 Euclidean balls and the norm cone

The remaining two pieces are curved, and they rest on the Euclidean norm $\|z\|_2 = \sqrt{z^\top z}$ of Week 1, the length of the vector z . Two properties of the norm are all we need, and the second follows from the Cauchy–Schwarz inequality $|x^\top y| \leq \|x\|_2 \|y\|_2$ (Proposition 1.11).

Lemma 2.13 (Two facts about the Euclidean norm). *For all $x, y \in \mathbb{R}^d$ and $\theta \in \mathbb{R}$,*

$$\|\theta x\|_2 = |\theta| \|x\|_2 \quad \text{and} \quad \|x + y\|_2 \leq \|x\|_2 + \|y\|_2 \quad (\text{triangle inequality}).$$

Proof. The first fact is $\|\theta x\|_2^2 = \theta^2 x^\top x$. For the second, expand the square and bound the middle term by Cauchy–Schwarz: $\|x + y\|_2^2 = \|x\|_2^2 + 2x^\top y + \|y\|_2^2 \leq \|x\|_2^2 + 2\|x\|_2 \|y\|_2 + \|y\|_2^2 = (\|x\|_2 + \|y\|_2)^2$; then take square roots. $\square$

Definition 2.14 (Euclidean ball). The **Euclidean ball** with centre $x_c \in \mathbb{R}^d$ and radius $r \geq 0$ is $\{x \in \mathbb{R}^d \mid \|x - x_c\|_2 \leq r\}$.

Definition 2.15 (Norm cone). The **norm cone** in $\mathbb{R}^{d+1}$ is $\{(x, t) \mid x \in \mathbb{R}^d, t \in \mathbb{R}, \|x\|_2 \leq t\}$.

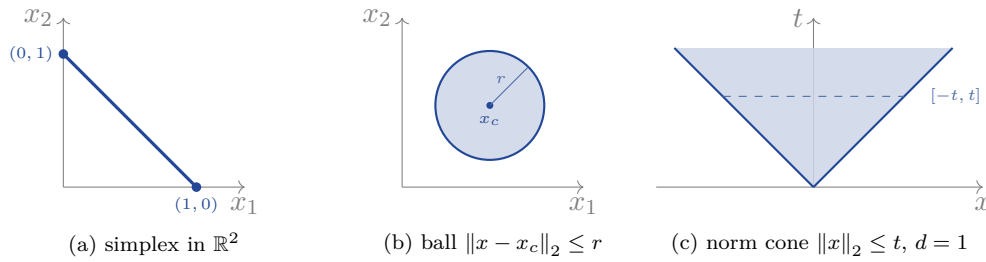


Figure 2.4: (a) The simplex in $\mathbb{R}^2$ is the segment joining $(1, 0)$ and $(0, 1)$. (b) A Euclidean ball with centre x_c and radius r . (c) The norm cone for $d = 1$, drawn in the (x, t) plane: the slice at height t is the ball $[-t, t]$ of radius t , and the cone continues upwards without bound.

Balls express distance limits (“stay within r of the current plan x_c ”), trust regions (a limit on how far a single step of an algorithm may move) and limits on the size of a residual; the constraint $\|\beta\|_2 \leq 1$ of the introduction is the unit ball centred at the origin, and a stretched and rotated ball, the ellipsoid $\{x_c + Au \mid \|u\|_2 \leq 1\}$, is treated in [Example 2.19](#). The norm cone has one more coordinate than the ball: a point is a pair (x, t) , and it belongs to the cone when the length of x is at most t . For $t \geq 0$ its slice at height t is the ball of radius t (for $t < 0$ the slice is empty), so the norm cone is “a ball whose radius is itself a variable”; for $d = 1$ it is the wedge $|x| \leq t$ of [Figure 2.4\(c\)](#).

Proposition 2.16. *Every Euclidean ball is convex, and the norm cone is a convex cone.*

Proof. Let x, y belong to the ball and $0 \leq \theta \leq 1$. Writing $x_c = \theta x_c + (1 - \theta)x_c$ and using both parts of [Lemma 2.13](#),

$$\|\theta x + (1 - \theta)y - x_c\|_2 = \|\theta(x - x_c) + (1 - \theta)(y - x_c)\|_2 \leq \theta \|x - x_c\|_2 + (1 - \theta) \|y - x_c\|_2 \leq r.$$

For the norm cone we use [Proposition 2.7\(c\)](#). Let (x, t) and (y, s) belong to it and $\theta \geq 0$. Scaling preserves the inequality, $\|\theta x\|_2 = \theta \|x\|_2 \leq \theta t$, and sums stay inside because $\|x + y\|_2 \leq \|x\|_2 + \|y\|_2 \leq t + s$. $\square$

Note on the lecture slides

Frame 2.16 justifies the norm cone by “ $\|x_1 + x_2\|_2 \leq \|x_1\|_2 + \|x_2\|_2 \leq t_1 + t_2$ (triangle inequality)”, but the triangle inequality is introduced neither there nor in Week 1, and the convexity of balls, asserted on frame 2.12, listed among the examples of convex sets on frame 2.16 and relied on in frames 2.20 and 2.21, is nowhere argued. The notes state the triangle inequality, with the homogeneity of the norm, as [Lemma 2.13](#), derive it from Week 1’s Cauchy–Schwarz inequality, and prove both claims; on frame 2.16 x_1, x_2 are two points, written x, y here.

The norm cone returns in [Week 4](#) under the name *second-order cone*, as one of the constraint types that conic solvers such as Gurobi and MOSEK accept directly.

2.3 Operations that preserve convexity

2.3.1 Why the definition does not scale

The line-segment definition is the foundation, and the two-point proof worked nicely for a single halfspace. But a realistic media-mix region stacks a budget cap, lower and upper bounds on each channel, several policy rules and a ball that limits risk; checking the definition would mean a separate proof for every constraint, then an argument for all of them together, redone after every revision of the model. This does not scale. The faster route is to recognise the standard

convex pieces of [Section 2.2](#) and to combine them only by operations that are *guaranteed* to preserve convexity: intersect, map affinely, project, lift. Each guarantee is a two-line consequence of [Definition 2.1](#), proved once below.

Key idea

Convexity is justified by structure. Standard convex pieces, combined by convexity-preserving operations, give a feasible region that is convex *by construction*; no new proof is required when the model changes.

2.3.2 Intersection

Proposition 2.17 (Intersections preserve convexity). *If $C_1, \dots, C_m \subseteq \mathbb{R}^d$ are convex, then their intersection $\bigcap_{i=1}^m C_i$ is convex.*

Proof. Let $x, y \in \bigcap_i C_i$ and $0 \leq \theta \leq 1$. For each i , both x and y lie in C_i , which is convex, so $\theta x + (1 - \theta)y \in C_i$. As this holds for every i , the point $\theta x + (1 - \theta)y$ lies in the intersection. $\square$

This is the rule that makes modelling work. A point is feasible when it satisfies *all* the constraints, so the feasible set of a model is the intersection of the sets defined by its individual constraints, and the proposition says that *stacking convex constraints is safe* ([Figure 2.5\(a\)](#)). It completes the catalogue: slabs, boxes, polyhedra and the simplex are intersections of halfspaces and hyperplanes, hence convex, and so is a long-only portfolio region with an additional cap on each weight, a simplex intersected with a box. The proof never uses the number of sets, so the rule holds for infinitely many sets as well. Two cautions: the intersection may be empty, in which case the model is infeasible ([Definition 1.7](#); the empty set is convex, so convexity promises nothing about feasibility), and the rule is about “and”, not “or”: the *union* of two convex sets is in general not convex, as the two disks of [Figure 2.1\(b\)](#) show.

2.3.3 Affine images and inverse images

An **affine map** from $\mathbb{R}^d$ to $\mathbb{R}^m$, already met in the chain rule of [Proposition 1.17](#), has the form $x \mapsto Ax + b$ with $A \in \mathbb{R}^{m \times d}$ and $b \in \mathbb{R}^m$, where m , the number of rows of A , may differ from d (in this subsection A and b denote a generic such map, not the data of a polyhedron). Translations ($A = I$), scalings, rotations and general linear reparameterisations ($b = 0$) are all affine maps, and A need not be square or invertible. The property that matters is that an affine map commutes with convex combinations: for all $x, y \in \mathbb{R}^d$ and $0 \leq \theta \leq 1$,

$$A(\theta x + (1 - \theta)y) + b = \theta(Ax + b) + (1 - \theta)(Ay + b), \quad (2.1)$$

because $\theta b + (1 - \theta)b = b$. Read from left to right and from right to left, this identity gives two closure rules.

Proposition 2.18 (Affine images and inverse images). *Let $A \in \mathbb{R}^{m \times d}$ and $b \in \mathbb{R}^m$.*

- (a) *If $C \subseteq \mathbb{R}^d$ is convex, then its **affine image** $\{Ax + b \mid x \in C\} \subseteq \mathbb{R}^m$ is convex.*
- (b) *If $C \subseteq \mathbb{R}^m$ is convex, then its **affine inverse image** $\{x \in \mathbb{R}^d \mid Ax + b \in C\}$ is convex.*

Proof. (a) Two points of the image have the form $Ax + b$ and $Ay + b$ with $x, y \in C$. By (2.1) their convex combination is the image of $\theta x + (1 - \theta)y$, which lies in C . (b) If $Ax + b \in C$ and $Ay + b \in C$, then the right-hand side of (2.1) lies in C by convexity, so the left-hand side says that $\theta x + (1 - \theta)y$ belongs to the inverse image. $\square$

Rule (a) says that an affine change of variables, such as passing from budget shares u to dollars $x = 10\,000u$, does not destroy convexity. Rule (b) is the one used most in modelling: a constraint “some affine expression of x must lie in a standard convex set” is convex in x .

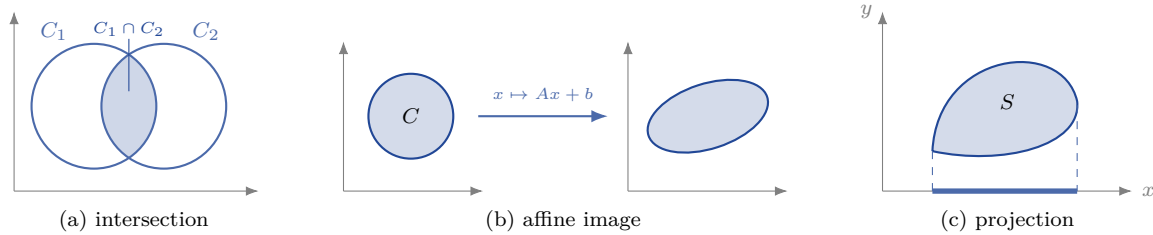


Figure 2.5: Three operations that preserve convexity. (a) The shaded lens is the intersection of two convex sets. (b) An affine map sends a disk to an ellipse, shifted, stretched and rotated but still convex. (c) The projection of a convex set S of points (x, y) onto the x -axis is its shadow, the thick interval.

Example 2.19 (An ellipsoid and a residual ball). An **ellipsoid** is a set $\{x_c + Au \mid \|u\|_2 \leq 1\}$ with centre $x_c \in \mathbb{R}^d$ and a matrix $A \in \mathbb{R}^{d \times d}$. It is the image of the unit ball under the affine map $u \mapsto Au + x_c$, so it is convex “for free” (Figure 2.5(b)); $A = rI$ gives back the ball of radius r . For statistics and machine learning the important example goes the other way. Stack the records of the least-squares model (Model 1.3) into a data matrix $A \in \mathbb{R}^{N \times d}$, whose i th row is $a_i^\top$, and a response vector $b \in \mathbb{R}^N$, and write the coefficient vector as x rather than β , as Week 1 announced. The set

$$\{x \in \mathbb{R}^d \mid \|Ax - b\|_2 \leq \varepsilon\}$$

contains the coefficient vectors whose residual $Ax - b$ has length at most ε . It is the inverse image of the ball $\{z \in \mathbb{R}^N \mid \|z\|_2 \leq \varepsilon\}$ under the affine map $x \mapsto Ax - b$ (the rule is applied with offset $-b$), hence convex by Proposition 2.18(b) and Proposition 2.16; no computation with the entries of A was needed.

2.3.4 Projection

Models often contain auxiliary variables that are needed to write the constraints but are not themselves of interest. Write a point of $\mathbb{R}^{d+d'}$ as a pair (x, y) with $x \in \mathbb{R}^d$ the variables of interest and $y \in \mathbb{R}^{d'}$ the auxiliary ones.

Proposition 2.20 (Projection preserves convexity). *If $S \subseteq \mathbb{R}^{d+d'}$ is convex, then its **projection** onto the x -coordinates,*

$$\{x \in \mathbb{R}^d \mid \text{there exists } y \in \mathbb{R}^{d'} \text{ such that } (x, y) \in S\},$$

is convex.

Proof. Let x and $\tilde{x}$ belong to the projection, with witnesses y and $\tilde{y}$, so that $(x, y) \in S$ and $(\tilde{x}, \tilde{y}) \in S$. Since S is convex it contains $(\theta x + (1 - \theta)\tilde{x}, \theta y + (1 - \theta)\tilde{y})$, and therefore $\theta y + (1 - \theta)\tilde{y}$ is a witness for $\theta x + (1 - \theta)\tilde{x}$. (Alternatively, the projection is the image of S under the linear map $(x, y) \mapsto x$, and Proposition 2.18(a) applies.) $\square$

Geometrically, the projection is the shadow of S on the x -space (Figure 2.5(c)). The practical message is that *hidden or auxiliary variables do not break convexity*: if the constraints are convex in all the variables jointly, the set of achievable x is convex too. A warning about vocabulary: here “projection” means dropping coordinates. From Week 8 onwards the same word denotes a different operation, the map that sends a point to the nearest point of a convex set.

2.3.5 The perspective cone

The simplex fixes the total of the shares at one. What if the total is itself a decision? We need a construction that turns a set of fixed size into a family of scaled copies of it, one for each size t , and does so without losing convexity.

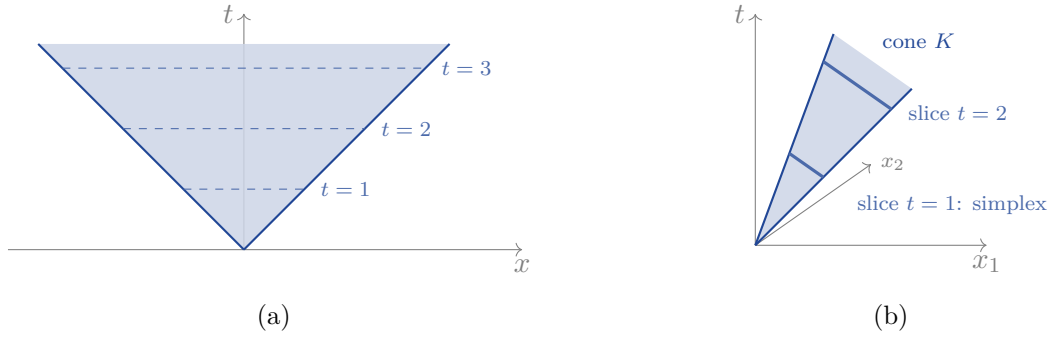


Figure 2.6: (a) The perspective cone of $C = [-1, 1]$ is the wedge $|x| \leq t$; its slices at $t = 1, 2, 3$ are the scaled copies $[-t, t]$ of C . (b) The perspective cone of the simplex in $\mathbb{R}^2$, drawn in (x_1, x_2, t) space: every slice is a scaled simplex (total spend t), and the edges are the rays through $(1, 0, 1)$ and $(0, 1, 1)$.

Definition 2.21 (Perspective cone). The **perspective cone** of a convex set $C \subseteq \mathbb{R}^d$ is

$$K = \{(x, t) \in \mathbb{R}^{d+1} \mid t > 0, x/t \in C\} \cup \{(0, 0)\}.$$

For $t > 0$ the condition $x/t \in C$ says $x \in tC = \{tz \mid z \in C\}$, so the slice of K at height t is the copy of C scaled by t : the original C sits at $t = 1$, a copy twice as large at $t = 2$, and for a bounded set C the copies shrink towards the single point $(0, 0)$ as $t \rightarrow 0$, which is why that point is adjoined. For the interval $C = [-1, 1]$ the slices are $[-t, t]$ and $K = \{(x, t) \mid |x| \leq t\}$ (Figure 2.6(a)). In the same way the unit ball gives $\{(x, t) \mid \|x\|_2 \leq t\}$: the norm cone is the perspective cone of the unit ball.

Proposition 2.22. If $C \subseteq \mathbb{R}^d$ is convex, then its perspective cone K is a convex cone.

Proof. We verify Proposition 2.7(c). *Scaling.* Let $(x, t) \in K$ with $t > 0$. For $\theta > 0$ we have $(\theta x)/(\theta t) = x/t \in C$, so $(\theta x, \theta t) \in K$, and $\theta = 0$ gives $(0, 0) \in K$; also $\theta(0, 0) = (0, 0) \in K$. *Sums.* Let $(x, t), (y, s) \in K$ with $t, s > 0$. Then

$$\frac{x+y}{t+s} = \frac{t}{t+s} \cdot \frac{x}{t} + \frac{s}{t+s} \cdot \frac{y}{s}$$

is a convex combination of the points x/t and y/s of C , with weights $t/(t+s)$ and $s/(t+s)$, so it lies in C and $(x+y, t+s) \in K$: a sum of two plans of sizes t and s is a plan of size $t+s$ whose “shape” is the size-weighted average of the two shapes, and convexity of C keeps that average admissible. Adding $(0, 0)$ to a point of K changes nothing. $\square$

Example 2.23 (From the simplex to a variable-budget cone). Start from the fixed-budget simplex $C = \{u \in \mathbb{R}^d \mid u \geq 0, \mathbf{1}^\top u = 1\}$, whose points are vectors of budget *shares*. For $t > 0$ we have $x/t \geq 0$ exactly when $x \geq 0$, and $\mathbf{1}^\top(x/t) = 1$ exactly when $\mathbf{1}^\top x = t$. At $t = 0$ the conditions $x \geq 0$ and $\mathbf{1}^\top x = 0$ force $x = 0$, which is precisely the point $(0, 0)$ adjoined in Definition 2.21. Hence the perspective cone of the simplex is

$$K = \{(x, t) \mid x \geq 0, \mathbf{1}^\top x = t, t \geq 0\},$$

where $t \geq 0$ is implied by the other two conditions and kept for readability. The interpretation is $x = tu$: the vector u holds shares, while (x, t) holds the actual spend x together with the total budget t . Shares $u = (0.5, 0.3, 0.2)$ with a budget $t = 8000$ give the spend $x = (4000, 2400, 1600)$. For $d = 2$, Figure 2.6(b) shows the cone, whose two edges are the rays through $(1, 0, 1)$ and $(0, 1, 1)$, “everything in channel 1” and “everything in channel 2”. The fixed-size feasible set has become a variable-size conic family, described by linear constraints only.

2.3.6 Lifting in modelling: the variable-budget model in RSOME

The norm cone and the perspective cone are both obtained by **lifting**: a variable t is added, and the set is described in the larger space of points (x, t) . Combined with projection, this is a standard modelling technique: a set that is awkward to describe in its own variables is described more cleanly in a higher-dimensional space, as an **extended formulation**, and the auxiliary variables are projected out at the end. By [Proposition 2.20](#) the visible set is then convex whenever the lifted description is, so more variables do not make a model harder when the lifted description is structured and convex. The following small model puts the variable-budget cone to work.

Model 2.24 (Variable-budget media mix). A retailer chooses the spend on search, social and video *and* the total budget, which costs 3.5 per dollar raised:

$$\begin{aligned} \max_{x,t} \quad & 4x_1 + 3x_2 + 2x_3 - 3.5t \\ \text{s.t.} \quad & \mathbf{1}^\top x = t, \quad x \geq 0, \quad 0 \leq t \leq 10\,000. \end{aligned} \tag{2.2}$$

Here x_1, x_2, x_3 are the coordinates of the spend vector $x \in \mathbb{R}^3$, in dollars per channel, as in [Model 1.1](#); the new decision variable $t \in \mathbb{R}$ is the total budget, which [Week 1](#) treated as the fixed number 10 000. The objective is the predicted number of conversions, with the same returns $r = (4, 3, 2)$ per dollar, minus a charge of 3.5 per budget dollar, measured in the same units as the returns (conversions per dollar) so that the objective is a net number of conversions: a dollar of budget is worth raising only if it buys more than 3.5 conversions. The constraint $\mathbf{1}^\top x = t$ says that the budget is fully spent, $x \geq 0$ that spend is nonnegative, and $0 \leq t \leq B$ with $B = 10\,000$ that at most \$10 000 can be raised. The policy rules $x_1 \geq 2000$ and $x_2 \leq 4000$ of [Week 1](#) are dropped in order to isolate the budget decision. The feasible set

$$\{(x, t) \mid x \geq 0, \mathbf{1}^\top x = t, 0 \leq t \leq 10\,000\}$$

is the variable-budget cone K of [Example 2.23](#) sliced by the box $0 \leq t \leq 10\,000$ in the t -coordinate (since $t \geq 0$ already holds in K , only the halfspace $t \leq 10\,000$ cuts anything): a standard cone sliced by a standard piece, hence convex by construction ([Proposition 2.17](#)), and in fact a polyhedron. With a linear objective the model is a linear programme, which any LP solver handles instantly, and it inherits the assumption of constant returns per dollar. Its projection onto the x -coordinates is $\{x \mid x \geq 0, \mathbf{1}^\top x \leq 10\,000\}$, exactly the budget constraint of [Week 1](#): the lifted model describes the same spending plans and additionally records what each of them costs.

Predicting the solution. Before running any code, substitute $t = \mathbf{1}^\top x$ into the objective:

$$4x_1 + 3x_2 + 2x_3 - 3.5(x_1 + x_2 + x_3) = 0.5x_1 - 0.5x_2 - 1.5x_3.$$

Only search earns more than the budget costs; a dollar given to social or video loses money. So $x_2 = x_3 = 0$, every dollar in search nets 0.5, and we want as many such dollars as the cap allows: $x^* = (10\,000, 0, 0)$, $t^* = 10\,000$, with optimal value $0.5 \times 10\,000 = 5000$.

Listing 2.1: The variable-budget model in RSOME.

```

from rsome import ro
from rsome import grb_solver as grb

m = ro.Model()
x = m.dvar(3)    # spend per channel
t = m.dvar()     # total budget: a decision!

m.max(4*x[0] + 3*x[1] + 2*x[2] - 3.5*t)
m.st(x.sum() == t,
      x >= 0,
      t >= 0, t <= 10000)

m.solve(grb)
print(m.solution.status) # 2 = optimal
print(m.get(), x.get(), t.get())

```

Listing 2.1 follows the RSOME pattern of Section 1.6: create the model, declare a vector of three variables and one scalar variable, state the objective with `m.max`, pass all constraints to `m.st`, and solve. With Gurobi it prints the status 2, followed by 5000.0 [10000. 0. 0.] 10000.0: the optimal value, x^* and t^* , as predicted.

Reading the solution. In the cone K the optimum sits on the edge ray through $(1, 0, 0, 1)$, “everything in search”, and it has moved out along that ray until the cap $t \leq B$ stopped it. The cap is active because each additional dollar still nets $0.5 > 0$; without it the objective could be increased for ever along the ray and the problem would be *unbounded* (Definition 1.7); Gurobi then reports code 4, “infeasible or unbounded”, and the default solver code 3, as in Section 1.6. Unboundedness is typical of a cone: scaling a feasible point is always allowed, so something else must limit the scale. The same reasoning covers other prices: if the budget cost exceeded 4, no channel would beat it and the optimum would be $t^* = 0$, spending nothing. As in Week 1, the all-or-nothing answer reflects constant returns.

Note on the lecture slides

The comment `# 2 = optimal` on frame 2.26 is specific to Gurobi, and the import of `grb_solver` fails unless Gurobi is installed. Week 1 recommends RSOME’s default solver first (Section 1.6): with `m.solve()` the model prints status 0, with `msk_solver` it prints `Optimal`, and all three return the same solution.

2.4 Recognising convex feasible regions

2.4.1 What does not preserve convexity

The toolkit is as useful for what it excludes. Table 2.2 sets the safe constructions of Section 2.3 against the patterns that usually break convexity, and for each pattern in the right-hand column two feasible points with an infeasible average are easy to find. An *either-or* rule is a union, as with the channel of Example 2.2 that is switched off or receives at least \$2000. A *binary or integer* restriction leaves gaps: $x_1 \in \{0, 1\}$ admits 0 and 1 but not 0.5, as in Example 2.9(iv). A *product of decision variables* bends the boundary the wrong way: the set $\{x \in \mathbb{R}_+^2 \mid x_1 x_2 \leq 1\}$, with coordinates x_1, x_2 , contains $(4, \frac{1}{4})$ and $(\frac{1}{4}, 4)$, but their midpoint $(2.125, 2.125)$ has product $4.52 > 1$. A *disconnected* region fails for the reason shown in Figure 2.1(b).

The word “usually” matters in both columns. The left column is safe only when the ingredients are convex. The right column lists warning signs, not theorems: the union of two nested convex sets, for instance, is convex. Nor are the nonconvex patterns “bad”. Fixed costs, indivisible units and logical conditions are real features of business problems; they simply require a different

Table 2.2: Constructions that preserve convexity and patterns that usually break it.

Usually safe in convex modelling	Usually breaks convexity
intersection of convex sets	union of sets, either-or logic
affine image or affine inverse image	binary and integer restrictions
projection of a convex lifted set	products of decision variables such as x_1x_2
perspective cone construction	disconnected feasible regions

toolkit, often mixed-integer or nonconvex optimisation, which cannot rely on the guarantee that a local solution is a global one.

2.4.2 A recognition checklist

For any future model the week condenses into three questions.

1. Is each primitive constraint a standard convex set, or the affine inverse image of one?
2. Is the full feasible region built from convex pieces by intersections, affine images, projections or perspective-type lifting?
3. Is there any discrete, either-or or bilinear (product of two decision variables) structure that breaks convexity?

If the first two answers are yes and the third is no, the feasible region is convex and the model is a candidate for the convex toolkit.

Example 2.25 (Applying the checklist). Consider the media-mix region described at the start of [Section 2.3](#): spends $x \in \mathbb{R}^3$ with budget cap $\mathbf{1}^\top x \leq 10\,000$, channel limits $0 \leq x \leq u$ (u the vector of caps, as in the box of [Definition 2.12](#)), the policy rules $x_1 \geq 2000$ and $x_2 \leq 4000$, and a risk limit $\|Fx\|_2 \leq \rho$ with a given matrix F and tolerance $\rho > 0$. Question 1: the cap and the policy rules are halfspaces, the bounds form a box, and the risk limit is the inverse image of the ball of radius ρ under the linear map $x \mapsto Fx$. Question 2: the region is the intersection of these pieces. Question 3: there is no integer variable, no “or” and no product of variables. The region is convex, and the verdict survives any edit that adds further constraints of the same kinds. Now add the rule “video is off or receives at least \$2000”, and question 3 fails: the region becomes the union of two convex pieces, one with $x_3 = 0$ and one with $x_3 \geq 2000$, and the model leaves the convex toolkit.

2.5 Summary and looking ahead

All the sets of this week come from one template, $\theta_1x_1 + \dots + \theta_kx_k$: affine combinations take weights of any sign that sum to one and produce lines and the solution sets of linear equations, convex combinations restrict the weights to averages and produce segments and convex hulls, and conic combinations drop the sum rule and add nonnegative scaling. A set is convex when averaging feasible points keeps them feasible. Hyperplanes, halfspaces, slabs, polyhedra, boxes, the simplex, Euclidean balls and the norm cone are the modelling templates; intersections, affine images, affine inverse images, projections and perspective cones are the operations that keep a region convex.

The closure-rule method deserves a verdict of its own. *What it does*: it certifies that a feasible region is convex by reading its structure. *Why it works*: each rule is a short consequence of the two-point definition, proved once. *When it applies*: whenever the region is assembled from standard pieces by the five operations, which covers every linear model and many nonlinear ones. *Its limitations*: the rules are sufficient, not necessary, so a set that fails the checklist may still be convex and then needs a direct argument; discrete, either-or and bilinear structure falls outside; and nothing has yet been said about the objective.

Later weeks keep drawing on the catalogue: the norm cone becomes the second-order cone of [Week 4](#), boxes, balls and the simplex are the “simple” sets of [Week 8](#), and the fact that a convex combination of feasible points is feasible keeps the iterates of the Frank–Wolfe method of [Week 13](#) feasible. The objective is the task of [Week 3](#), which adds convex *functions* and their sublevel sets and proves the objective’s half of the promise of [Week 1](#): a local minimiser of a convex function is a global one. [Week 4](#) then joins the two halves: with a convex objective minimised over a convex feasible set, local and global minima coincide ([Theorem 4.1](#)). The link is a lifting of the kind used above: a function is turned into a set, its epigraph ([Definition 3.1](#)), so that this week’s rules become rules for recognising convex functions.

2.6 Exercises

In these exercises x_1, x_2 denote the coordinates of a point $x \in \mathbb{R}^2$, and “cone” means convex cone.

Exercise 2.1 (Convex, cone, both or neither). Classify each set as convex, cone, both, or neither. Parts (a) and (d) were classified from the definition in [Example 2.9](#); here justify every positive answer by a closure rule or the cone test of [Proposition 2.7](#) instead, and every negative answer by a counterexample (a violating pair of points, or a point whose nonnegative multiple leaves the set): (a) $\{x \in \mathbb{R}^2 \mid x \geq 0, \mathbf{1}^\top x \leq 1\}$; (b) $\{x \in \mathbb{R}^2 \mid a^\top x = 0\}$ for a fixed $a \neq 0$; (c) $\{x \in \mathbb{R}^2 \mid 1 \leq \|x\|_2 \leq 2\}$; (d) $\{x \in \mathbb{R}^2 \mid x_1 \in \{0, 1\}, 0 \leq x_2 \leq 1\}$.

Exercise 2.2 (Convex and conic combinations). Let $u = (1, 0)$ and $v = (0, 2)$. Describe all convex combinations and all conic combinations of u and v .

Exercise 2.3 (Feasible sets from business rules). Write the feasible set for each of the following, and name the standard sets involved: (a) a budget split across three channels with nonnegative shares summing to one; (b) a long-only portfolio with total investment one and weights at most 0.5; (c) a two-product resource-allocation model with linear capacity limits.

Exercise 2.4 (Closure rules and coordinate projection). Name the closure rules used, as in [Example 2.19](#), to show that $\{x \in \mathbb{R}^d \mid \|Ax - b\|_2 \leq 1\} \cap \{x \in \mathbb{R}^d \mid x \geq 0\}$ is convex, and write the projection of $\{(x_1, x_2) \in \mathbb{R}^2 \mid x_1 \geq 0, x_2 \geq 0, x_1 + x_2 = 1\}$ onto the x_1 -coordinate.

Exercise 2.5 (Sketch, plot and check). Sketch the feasible region $\{x \in \mathbb{R}^2 \mid x_1 \geq 0, x_2 \geq 0, x_1 + x_2 \leq 4, x_1 \leq 3\}$. Then plot it in Python, pick two feasible points, and check numerically that several of their convex combinations are feasible.

Exercise 2.6 (RSOME mini-lab). With $r = (4, 3, 2)$, a budget cost of 3.5 per dollar and $B = 10\,000$, consider

$$\begin{aligned} \max \quad & r^\top x - 3.5t \\ \text{s.t.} \quad & \mathbf{1}^\top x = t, \quad x \geq 0, \quad 0 \leq t \leq B. \end{aligned}$$

[Model 2.24](#) solved this model with a budget cost of 3.5. Solve it in RSOME and confirm the solution; then repeat with costs 4.5 and 2.5, predicting each answer first and saying which channels beat the budget cost, where the optimum sits in the cone, and whether the budget bound is active.

Week 3 Convex Functions and How to Recognise Them

[Week 2](#) answered the question of which *feasible sets* can be optimised over safely. This week asks the companion question about the *objective*: how do we recognise a convex objective, and why does a convex objective make minimisation reliable? Two models can share the same feasible set and still behave very differently. If the objective has a single basin, any procedure that keeps moving downhill ends at the global minimum x^* ; if it has two basins, like the two-well function in the right panel of [Figure 3.1](#), the procedure stops at the bottom of the basin it starts in, and nothing it observes locally reveals that a better point exists elsewhere. [Week 1](#) distinguished local from global minima ([Definition 1.20](#)) and promised that for convex problems the distinction disappears. This week proves the objective's half of that promise.

The organising device of the week is the *epigraph*. Lifting a function f to the set of points on or above its graph turns a question about a function into a question about a set, and for sets we already have the toolkit of [Section 2.3](#). We shall see that f is convex exactly when its epigraph is a convex set, that the tests of calculus (chords, tangents, second derivatives) are readings of that one fact, and that the rules for building convex functions from simpler ones are, for the most part, the set operations of [Week 2](#) applied to epigraphs.

The section has five parts. [Section 3.1](#) introduces the epigraph and the definition of a convex function. [Section 3.2](#) develops the one-variable tests, and [Section 3.3](#) carries them to $\mathbb{R}^d$, proves that stationary points of convex functions are global minimisers, settles convex quadratics, and shows that convexity survives kinks. [Section 3.4](#) gives the rules that preserve convexity, and [Section 3.5](#) turns the epigraph into a modelling move solved in RSOME. The running examples are least squares, convex quadratics, $|x|$ and the ℓ_1 norm, maxima of affine functions, the two-well function $(|x| - 1)^2$, and $x \log x$. This week t denotes the epigraph (height) variable unless stated otherwise; g and z are reused for auxiliary functions and variables, as announced locally.

3.1 From convex sets to convex functions

3.1.1 The epigraph

Throughout, f is a real-valued function of $x \in \mathbb{R}^d$ whose *domain* $\text{dom } f \subseteq \mathbb{R}^d$ is the set of points at which it is defined and finite; for instance $\text{dom}(-\log) = \mathbb{R}_{++}$. Equivalently, f may be regarded as defined on all of $\mathbb{R}^d$ with the value $+\infty$ outside $\text{dom } f$: such a point never satisfies $f(x) \leq t$ for a real t , so both conventions give the same set in the next definition.

Definition 3.1 (Epigraph). The **epigraph** of a function $f: \text{dom } f \rightarrow \mathbb{R}$ with $\text{dom } f \subseteq \mathbb{R}^d$ is the set

$$\text{epi } f = \{(x, t) \mid x \in \text{dom } f, f(x) \leq t\} \subseteq \mathbb{R}^{d+1}.$$

The epigraph is the region on or above the graph of f (“epi” means “above”); its lower boundary is the graph itself, where $t = f(x)$. The left panel of [Figure 3.1](#) shades it for a function of one variable. The epigraph also gives a second way of writing a minimisation problem. For any feasible set $\mathcal{X} \subseteq \text{dom } f$,

$$\min_{x \in \mathcal{X}} f(x) \iff \min_{x \in \mathcal{X}, t \in \mathbb{R}} \{t \mid f(x) \leq t\}. \quad (3.1)$$

The two problems have the same optimal value and the same optimal x : a feasible (x, t) on the right has $t \geq f(x)$, so the right cannot beat the left, and for every $x \in \mathcal{X}$ the pair $(x, f(x))$ is feasible on the right with objective $f(x)$; at an optimum the constraint is tight, $t^* = f(x^*)$. Geometrically, minimising f means sliding downwards in the t direction for as long as we stay in $\text{epi } f$: the optimal value is the height of the lowest point of the epigraph above $\mathcal{X}$. The

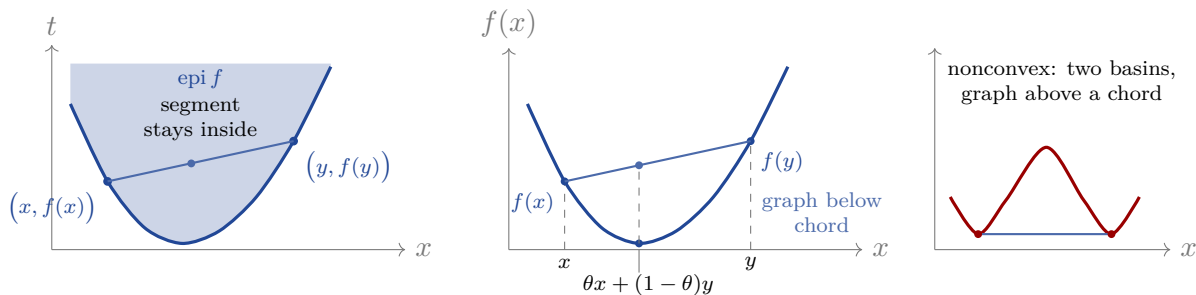


Figure 3.1: Left: the epigraph of a convex function is a convex set. Middle: on the lower boundary this is the chord inequality (3.2): the value at an averaged input (lower dot) lies below the average of the values (upper dot). Right: a two-well function such as $(|x| - 1)^2$ violates it, and a local search can be trapped in either basin.

right-hand problem is the **epigraph form**. Its objective is linear and all the nonlinearity sits in the constraint $f(x) \leq t$; this is the modelling move of Section 3.5 and of Week 4.

3.1.2 Convex functions and convex epigraphs

Recall from Definition 2.1 that a set is convex if it contains the segment $\theta x + (1 - \theta)y$, $0 \leq \theta \leq 1$, between any two of its points. The definition of a convex function compares the function along such a segment with the straight line joining its end values; the domain must be convex so that f can be evaluated at every point in between.

Definition 3.2 (Convex function). A function f is **convex** if $\text{dom } f$ is a convex set and

$$f(\theta x + (1 - \theta)y) \leq \theta f(x) + (1 - \theta)f(y) \quad \text{for all } x, y \in \text{dom } f \text{ and all } 0 \leq \theta \leq 1. \quad (3.2)$$

It is **strictly convex** if the inequality is strict whenever $x \neq y$ and $0 < \theta < 1$, and **concave** if $-f$ is convex. An affine function $a^\top x + b$ (Week 1) satisfies (3.2) with equality and is therefore both convex and concave.

Theorem 3.3 (Convex function $\Leftrightarrow$ convex epigraph). A function f is convex if and only if $\text{epi } f$ is a convex set.

Proof. Suppose f is convex and take $(x, s), (y, u) \in \text{epi } f$, so $f(x) \leq s$ and $f(y) \leq u$. For $0 \leq \theta \leq 1$ the point $\theta x + (1 - \theta)y$ lies in $\text{dom } f$ and

$$f(\theta x + (1 - \theta)y) \leq \theta f(x) + (1 - \theta)f(y) \leq \theta s + (1 - \theta)u,$$

which says that $\theta(x, s) + (1 - \theta)(y, u) \in \text{epi } f$. Conversely, suppose $\text{epi } f$ is convex. Then $\text{dom } f$, the coordinate projection of $\text{epi } f$ onto x , is convex (Proposition 2.20). For $x, y \in \text{dom } f$ the points $(x, f(x))$ and $(y, f(y))$ lie in $\text{epi } f$, hence so does $(\theta x + (1 - \theta)y, \theta f(x) + (1 - \theta)f(y))$, and membership of $\text{epi } f$ is exactly the inequality (3.2). $\square$

The theorem is the bridge between this week and the last. In one direction it explains the definition: on $\mathbb{R}$ the lower boundary of $\text{epi } f$ is the graph, so “the segment between two lifted points stays in the epigraph” is the visible rule that *the graph lies below every chord* (Figure 3.1, middle). In the other direction it proves a function convex by showing that a set is convex. A first instance is already in hand: the norm cone $\{(x, t) \mid \|x\|_2 \leq t\}$ of Definition 2.15 is precisely $\text{epi } \|\cdot\|_2$ and was shown in Week 2 to be a convex cone, using the triangle inequality of Lemma 2.13, so the Euclidean norm is a convex function.

3.2 Convex functions of one variable

3.2.1 The chord inequality and its relatives

For $d = 1$ the convex subsets of $\mathbb{R}$ are the intervals, so a convex function of one variable is a function on an interval that satisfies (3.2). The inequality compares *the value at an average input* with *the average of the values*: if $f(x)$ is the cost of operating at level x , then running a blend of two plans never costs more than the same blend of their costs.

Three relatives of convexity were named in Definition 3.2. Concavity is the mirror image: the graph of a concave function lies *above* every chord, and maximising a concave function is the same as minimising a convex one; this is why the maximisation in Model 1.2 belongs to the convex world. Affine functions are exactly the functions that are both convex and concave. Strict convexity rules out flat pieces in the graph, and it makes minimisers *unique*: if $x \neq y$ were two minimisers of a strictly convex f over a convex set $\mathcal{X}$, with common value f^* , the midpoint $\frac{1}{2}x + \frac{1}{2}y \in \mathcal{X}$ would have a value strictly below $\frac{1}{2}f(x) + \frac{1}{2}f(y) = f^*$, which is impossible. Hence a strictly convex function has at most one minimiser (it may have none: e^x on $\mathbb{R}$).

Note on the lecture slides

Frame 3.8 says that strict convexity makes minimisers unique, “an argument we run in Week 5”. It is the midpoint argument just given; the course runs it in Week 4 (Proposition 4.2), and Week 5 only states the conclusion.

3.2.2 The tangent-line test

Checking (3.2) for all pairs of points is rarely practical; for differentiable functions convexity can be read from the derivative instead. Recall the first-order approximation of Equation (1.4): near x , $f(y) \approx f(x) + f'(x)(y - x)$, the tangent line at x . For a general function the approximation is only local; for a convex function it is a *global* lower bound. The interval below is taken open so that f' is an ordinary two-sided derivative at every point (at an endpoint one uses the one-sided derivative, and nothing changes).

Proposition 3.4 (Tangent-line test). *Let f be differentiable on an open interval I . The following are equivalent:*

- (i) f is convex on I ;
- (ii) $f(y) \geq f(x) + f'(x)(y - x)$ for all $x, y \in I$ (every tangent line is a global under-estimator);
- (iii) f' is nondecreasing on I .

Proof. (i) $\Rightarrow$ (ii). Write the chord inequality with weight $\theta \in (0, 1]$ on y : $f(x + \theta(y - x)) \leq (1 - \theta)f(x) + \theta f(y)$. Rearranging, $[f(x + \theta(y - x)) - f(x)]/\theta \leq f(y) - f(x)$, and as $\theta \downarrow 0$ the left-hand side tends to $f'(x)(y - x)$. (ii) $\Rightarrow$ (i). Put $z = \theta x + (1 - \theta)y$ and apply (ii) twice with base point z : $f(x) \geq f(z) + f'(z)(x - z)$ and $f(y) \geq f(z) + f'(z)(y - z)$. Multiply the first by θ , the second by $1 - \theta$ and add; the derivative terms cancel because $\theta(x - z) + (1 - \theta)(y - z) = 0$, leaving $\theta f(x) + (1 - \theta)f(y) \geq f(z)$. (ii) $\Rightarrow$ (iii). Add (ii) to the same inequality with x and y exchanged to get $(f'(y) - f'(x))(y - x) \geq 0$, so $y > x$ implies $f'(y) \geq f'(x)$. (iii) $\Rightarrow$ (ii). By the mean value theorem $f(y) - f(x) = f'(\xi)(y - x)$ for some ξ between x and y . If $y > x$ then $f'(\xi) \geq f'(x)$; if $y < x$ then $f'(\xi) \leq f'(x)$ and $y - x < 0$. In both cases $f'(\xi)(y - x) \geq f'(x)(y - x)$. $\square$

The left panel of Figure 3.2 shows statement (ii): *local* information (the value and slope at one point) yields a *global* bound on f , the source of the week’s payoff, Theorem 3.10.

3.2.3 The second-derivative test

For a twice-differentiable function, statement (iii) becomes the most convenient one-variable test.

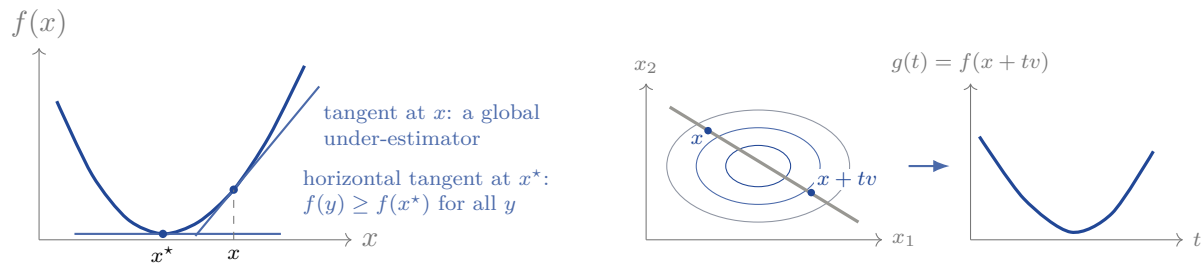


Figure 3.2: Left: every tangent of a convex function lies below its graph; at a stationary point x^* the tangent is horizontal, so $f(x^*)$ is a global lower bound (Theorem 3.10). Right: restriction to a line (Proposition 3.7): along the line through x with direction v , a convex function of two variables (drawn by its contours) is a convex function of t .

Proposition 3.5 (Second-derivative test). *Let f be twice differentiable on an open interval I . Then f is convex on I if and only if $f''(x) \geq 0$ for all $x \in I$. If $f''(x) > 0$ for all $x \in I$, then f is strictly convex.*

Proof. f' is nondecreasing on I if and only if its derivative f'' is nonnegative on I , so the first claim is Proposition 3.4(iii). If $f'' > 0$ then f' is strictly increasing, the mean-value step above gives (ii) with strict inequality for $y \neq x$, and the step (ii) $\Rightarrow$ (i) then gives the strict chord inequality for $x \neq y$ and $0 < \theta < 1$. $\square$

Two limitations should be kept in mind: $f'' > 0$ is sufficient but not necessary for strict convexity (x^4 is strictly convex although $f''(0) = 0$), and the test needs two derivatives, so it is silent about kinks; for large multivariate models the main recognition tool is instead the set of rules in Section 3.4.

Example 3.6 (A gallery of one-variable functions). *Smooth convex functions.* x^2 has $f'' = 2$ and e^x has $f'' = e^x > 0$, so both are strictly convex on $\mathbb{R}$. On $\mathbb{R}_{++}$, $-\log x$ has $f'' = 1/x^2 > 0$, and $x \log x$ has $f' = 1 + \log x$ and $f'' = 1/x > 0$; both are strictly convex there (they return as the log barrier of Week 13 and in Example 4.22).

Convex functions with a kink. $|x|$ and $\max\{0, x\}$ are not differentiable at 0, so the derivative tests do not apply there, but the definition does. For the absolute value, $|\theta x + (1 - \theta)y| \leq \theta|x| + (1 - \theta)|y|$ is the triangle inequality. For $\max\{0, x\}$, both 0 and $\theta x + (1 - \theta)y$ are at most $\theta \max\{0, x\} + (1 - \theta) \max\{0, y\}$, hence so is their maximum.

A non-example. x^3 has $f'' = 6x < 0$ for $x < 0$, so it is not convex on $\mathbb{R}$. The chord inequality fails at $x = -2$, $y = 0$, $\theta = \frac{1}{2}$: $f(-1) = -1$, whereas $\frac{1}{2}f(-2) + \frac{1}{2}f(0) = -4$. On the smaller domain $\mathbb{R}_+$ the same formula *is* convex: convexity belongs to a function together with its domain.

3.3 Convex functions of several variables

3.3.1 Definition and restriction to a line

On $\mathbb{R}^d$ the definition is word for word the same (Definition 3.2 with x and y now vectors in a convex domain); only the geometry has changed: the graph is a surface in $\mathbb{R}^{d+1}$ lying below each of its chords. The cleanest bridge from one variable to many is to look at f along lines.

Proposition 3.7 (Restriction to a line). *A function f on $\mathbb{R}^d$ is convex if and only if, for every $x \in \text{dom } f$ and every direction $v \in \mathbb{R}^d$, the one-variable slice*

$$g(t) = f(x + tv), \quad \text{dom } g = \{t \in \mathbb{R} \mid x + tv \in \text{dom } f\},$$

is convex in the scalar t . (Here t is the position along the line, not an epigraph variable.)

Proof. If f is convex, $\text{dom } g$ is the intersection of a line with a convex set, hence an interval, and $g(\theta t_1 + (1 - \theta)t_2) = f(\theta(x + t_1v) + (1 - \theta)(x + t_2v)) \leq \theta g(t_1) + (1 - \theta)g(t_2)$. Conversely, given $x, y \in \text{dom } f$ take $v = y - x$, so that $g(0) = f(x)$ and $g(1) = f(y)$. Convexity of the slice gives $[0, 1] \subseteq \text{dom } g$, so the whole segment lies in $\text{dom } f$, and $f(\theta x + (1 - \theta)y) = g(1 - \theta) \leq \theta g(0) + (1 - \theta)g(1)$, which is (3.2). $\square$

Picture walking through the domain along a straight line and recording the height of f (Figure 3.2, right): f is convex exactly when every such profile is. With the chain rule this turns each test of Section 3.2 into a test on $\mathbb{R}^d$. In the results below $\mathcal{X}$ denotes the domain of f (here it plays the role of $\text{dom } f$, not of a feasible set as in Definition 1.4), a convex set, taken open (for example $\mathbb{R}^d$ or $\mathbb{R}_{++}^d$) whenever derivatives are involved, so that every slice is differentiable on an open interval. The first-order results remain true, with the same proof, when f is differentiable on an open set containing a convex $\mathcal{X}$; only the Hessian test needs $\mathcal{X}$ itself to be open.

3.3.2 The first-order inequality and monotone gradients

For differentiable f the slice has derivative $g'(t) = \nabla f(x + tv)^\top v$, by the chain rule of Proposition 1.17; at $t = 0$ this is the directional derivative of Definition 1.15.

Theorem 3.8 (First-order inequality). *Let f be differentiable on an open convex domain $\mathcal{X} \subseteq \mathbb{R}^d$. Then f is convex if and only if*

$$f(y) \geq f(x) + \nabla f(x)^\top (y - x) \quad \text{for all } x, y \in \mathcal{X}. \quad (3.3)$$

Proposition 3.9 (Monotone gradient). *Let f be differentiable on an open convex domain $\mathcal{X} \subseteq \mathbb{R}^d$. Then f is convex if and only if its gradient is **monotone**:*

$$(\nabla f(x) - \nabla f(y))^\top (x - y) \geq 0 \quad \text{for all } x, y \in \mathcal{X}. \quad (3.4)$$

Proof of both results. Let $u = x + t_1v$ and $w = x + t_2v$ be two points on a line in $\mathcal{X}$, and let g be the slice along that line. Since $g'(t) = \nabla f(x + tv)^\top v$ and $w - u = (t_2 - t_1)v$,

$$\begin{aligned} g(t_1) + g'(t_1)(t_2 - t_1) &= f(u) + \nabla f(u)^\top (w - u), \\ (g'(t_2) - g'(t_1))(t_2 - t_1) &= (\nabla f(w) - \nabla f(u))^\top (w - u). \end{aligned}$$

Hence (3.3) holds for all pairs of points if and only if every slice satisfies the tangent-line inequality, and (3.4) holds for all pairs if and only if every slice has a nondecreasing derivative. By Proposition 3.4 each of these holds exactly when every slice is convex, that is (Proposition 3.7), exactly when f is convex. $\square$

The right-hand side of (3.3) is the first-order approximation of Equation (1.4), the tangent plane at x : the best affine predictor of f near x , which under convexity *never over-estimates, anywhere*. Inequality (3.4) generalises “ f' is nondecreasing”: the change in the gradient makes a nonnegative inner product with the displacement. The first-order inequality is the most used inequality of the course: it certifies optimality in Week 5 and drives the convergence proofs from Week 6 onwards.

3.3.3 Stationary points are global minimisers

A point x^* with $\nabla f(x^*) = 0$ is called a **stationary point**. For a general function stationarity says very little: the point could be a local minimum, a local maximum or a saddle. For a convex function it settles everything.

Theorem 3.10 (Stationary means global; local means global). *Let f be convex.*

- (a) If $\text{dom } f$ is open, f is differentiable on it and $\nabla f(x^*) = 0$, then $f(y) \geq f(x^*)$ for all $y \in \text{dom } f$: x^* is a global minimiser.
- (b) Every local minimiser of f is a global minimiser, whether or not f is differentiable.

Proof. (a) Take $x = x^*$ in (3.3), which is legitimate because the inequality holds for *all* pairs of points. The correction term is the inner product of a zero gradient with $y - x^*$, so it vanishes and $f(y) \geq f(x^*)$ remains. (b) Let x^* be a local minimiser, so $f(z) \geq f(x^*)$ whenever $z \in \text{dom } f$ and $\|z - x^*\|_2 \leq \rho$, for some radius $\rho > 0$. Suppose some y had $f(y) < f(x^*)$. The points $z_\theta = x^* + \theta(y - x^*)$, $0 < \theta \leq 1$, satisfy $f(z_\theta) \leq (1 - \theta)f(x^*) + \theta f(y) < f(x^*)$, and for small θ they lie within distance ρ of x^* . This contradicts local optimality. $\square$

Key idea

For a convex function a local certificate is a global one: the horizontal tangent plane at a stationary point is a lower bound for the whole function (Figure 3.2, left), and a better point anywhere would drag the function down along the segment towards it. Convex objectives have no spurious local minima.

This is the promise made in Definition 1.20; note its limits. The theorem does not say that a minimiser exists, nor that it is unique (that needs strict convexity). Part (a) concerns unconstrained minimisation, where $\nabla f(x^*) = 0$ is also necessary for optimality (Theorem 5.1). With constraints a minimiser may sit on the boundary of the feasible set with a nonzero gradient, and the certificate changes; that is the subject of Section 5.2. Part (b) survives unchanged on a convex feasible set, with the same proof (Theorem 4.1).

3.3.4 The Hessian test

The **Hessian** of a twice-differentiable f at x is the symmetric $d \times d$ matrix $\nabla^2 f(x)$ with entries $[\nabla^2 f(x)]_{ij} = \partial^2 f / \partial x_i \partial x_j$. It plays the role of f'' , and the condition $f'' \geq 0$ becomes a sign condition on a matrix.

Definition 3.11 (Positive semidefinite and positive definite matrices). A symmetric matrix $Q \in \mathbb{R}^{d \times d}$ is **positive semidefinite**, written $Q \succeq 0$, if $z^\top Q z \geq 0$ for all $z \in \mathbb{R}^d$. It is **positive definite**, written $Q \succ 0$, if $z^\top Q z > 0$ for all $z \neq 0$. We write $A \succeq B$ when $A - B \succeq 0$.

Theorem 3.12 (Hessian test). *Let f be twice differentiable on an open convex domain $\mathcal{X} \subseteq \mathbb{R}^d$. Then f is convex if and only if $\nabla^2 f(x) \succeq 0$ for all $x \in \mathcal{X}$. If $\nabla^2 f(x) \succ 0$ for all $x \in \mathcal{X}$, then f is strictly convex.*

Proof. Differentiating $g'(t) = \nabla f(x + tv)^\top v$ once more gives $g''(t) = v^\top \nabla^2 f(x + tv) v$: the second derivative of a slice is the curvature of f in the direction v . By Propositions 3.5 and 3.7, f is convex if and only if $g'' \geq 0$ for every slice, that is, $v^\top \nabla^2 f(u) v \geq 0$ for every point $u \in \mathcal{X}$ and every direction v . If the Hessian is positive definite, every slice with $v \neq 0$ has $g'' > 0$ and is strictly convex, which gives strict convexity of f . $\square$

The test generalises $f''(x) \geq 0$: curvature is nonnegative in *every* direction z . Positive semidefiniteness is checked through the numbers $z^\top Q z$. Week 10 introduces eigenvalues; the smallest and largest are the minimum and maximum of $z^\top Q z$ over unit vectors z , and they measure how unevenly a quadratic curves.

Example 3.13 (Least squares is convex). Let $f(x) = \frac{1}{2} \|Ax - b\|_2^2$ with data $A \in \mathbb{R}^{N \times d}$ and $b \in \mathbb{R}^N$, the matrix form of Model 1.3. By Proposition 1.17, $\nabla f(x) = A^\top (Ax - b) = A^\top Ax - A^\top b$. The gradient is affine in x , and differentiating once more gives the constant Hessian $\nabla^2 f(x) = A^\top A$. For every z ,

$$z^\top A^\top A z = (Az)^\top (Az) = \|Az\|_2^2 \geq 0,$$

so $\nabla^2 f \succeq 0$ and least squares is convex *for any data* A, b . Consequently, by [Theorem 3.10](#), every solution of the normal equations $A^\top Ax = A^\top b$ is a global minimiser. Moreover $\|Az\|_2^2 > 0$ for all $z \neq 0$ exactly when the columns of A are linearly independent; then $A^\top A \succ 0$, f is strictly convex and the minimiser is unique (compare [Proposition 1.13](#)). Other scalings, such as $\frac{1}{N} \|Ax - b\|_2^2$, multiply the gradient and Hessian by a positive constant and change nothing.

Note on the lecture slides

Frame 3.16 states the Hessian test “on a convex domain $\mathcal{X}$ ” and defines only $Q \succeq 0$, although later frames use $Q \succ 0$ and strict convexity; the notes add both. The implication “convex $\Rightarrow \nabla^2 f \succeq 0$ ” needs an *open* domain: $x_1^2 - x_2^2$ restricted to the line $\{x \mid x_2 = 0\}$ is the convex function x_1^2 , yet its Hessian $\text{diag}(2, -2)$ is not positive semidefinite.

3.3.5 Convex quadratics

Quadratic objectives are everywhere in analytics: the least-squares loss, the portfolio risk $w^\top \Sigma w$ of [Model 1.2](#), the ridge penalty $\lambda \|x\|_2^2$, and the local quadratic models that algorithms build from [Week 6](#) onwards. Their convexity is settled completely by the matrix of the quadratic part.

Proposition 3.14 (Convex quadratics). *Let $f(x) = \frac{1}{2}x^\top Qx + q^\top x + r$ with $Q = Q^\top \in \mathbb{R}^{d \times d}$, $q \in \mathbb{R}^d$ and $r \in \mathbb{R}$.*

- (a) *f is convex if and only if $Q \succeq 0$, and strictly convex if and only if $Q \succ 0$. The linear and constant terms play no role.*
- (b) (Factorisation test.) *If $Q = A^\top A$ for some matrix A , then $Q \succeq 0$. If $Q = A^\top A + \mu I$ with $\mu > 0$, then $Q \succ 0$.*
- (c) (Two-dimensional test.) $\begin{bmatrix} a & b \\ b & c \end{bmatrix} \succeq 0$ if and only if $a \geq 0$, $c \geq 0$ and $ac - b^2 \geq 0$.

Proof. (a) The Hessian of f is Q at every point, so the first claim is [Theorem 3.12](#). A direct expansion shows more: for all x, y and θ ,

$$\theta f(x) + (1 - \theta)f(y) - f(\theta x + (1 - \theta)y) = \frac{1}{2} \theta(1 - \theta) (x - y)^\top Q (x - y),$$

the affine terms cancelling. The right-hand side is nonnegative for all choices exactly when $Q \succeq 0$, and strictly positive whenever $x \neq y$ and $0 < \theta < 1$ exactly when $Q \succ 0$. (b) $z^\top (A^\top A + \mu I)z = \|Az\|_2^2 + \mu \|z\|_2^2$, which is nonnegative when $\mu = 0$ and strictly positive for $z \neq 0$ when $\mu > 0$. (c) Write $z^\top Qz = az_1^2 + 2bz_1z_2 + cz_2^2$. The axis directions $z = (1, 0)$ and $z = (0, 1)$ force $a \geq 0$ and $c \geq 0$. If $a > 0$, completing the square gives

$$z^\top Qz = a \left(z_1 + \frac{b}{a} z_2 \right)^2 + \frac{ac - b^2}{a} z_2^2,$$

which is nonnegative for all z exactly when $ac - b^2 \geq 0$ (choose $z_2 = 1$, $z_1 = -b/a$ for the necessity): the cross term b must not overpower the axis curvatures a and c . If $a = 0$, the condition $ac - b^2 \geq 0$ reads $-b^2 \geq 0$, that is $b = 0$, and then $z^\top Qz = cz_2^2 \geq 0$; conversely, if $a = 0$ and $Q \succeq 0$, the directions $z = (s, 1)$ give $2bs + c \geq 0$ for all $s \in \mathbb{R}$, forcing $b = 0$. $\square$

Part (b) is the practical test: a quadratic that is visibly a sum of squares, $\|Ax\|_2^2$, is convex, and adding $\mu \|x\|_2^2$ with $\mu > 0$ makes it strictly convex. Least squares ($Q = A^\top A$) and ridge regression ($Q = A^\top A + 2\lambda I$ for $\frac{1}{2} \|Ax - b\|_2^2 + \lambda \|x\|_2^2$) are the canonical cases, and μ is the prototype of the strong-convexity constant of [Definition 6.12](#) (this scalar μ is unrelated to the vector of expected returns μ of [Model 1.2](#)). Portfolio risk is covered too: Σ is the covariance matrix of the asset returns, so $z^\top \Sigma z$ is the variance of the return of a portfolio with holdings z . A variance is never negative, so $\Sigma \succeq 0$ and $w^\top \Sigma w$ is convex.

Example 3.15 (The two-dimensional test). Consider $2x_1^2 + 2x_1x_2 + x_2^2 - 4x_1$. The linear term is irrelevant; matching the quadratic part with $ax_1^2 + 2bx_1x_2 + cx_2^2$ gives $a = 2$, $b = 1$, $c = 1$. The quadratic part is $x^\top Qx$ with $Q = \begin{bmatrix} a & b \\ b & c \end{bmatrix}$ (no factor $\frac{1}{2}$), so the Hessian is $2Q$, which is positive semidefinite exactly when Q is; part (c) decides: $a, c \geq 0$ and $ac - b^2 = 1 \geq 0$, so the function is convex. In fact $z^\top Qz = 2(z_1 + \frac{1}{2}z_2)^2 + \frac{1}{2}z_2^2 > 0$ for $z \neq 0$, so $Q \succ 0$ and it is strictly convex. Now consider $x_1^2 + 4x_1x_2 + x_2^2$, with $a = c = 1$ and $b = 2$. Both axis curvatures are positive, but $ac - b^2 = -3 < 0$, and the direction $z = (1, -1)$ exposes the problem: $z^\top Qz = 1 - 4 + 1 = -2$, and along that line the function is $-2t^2$, which curves downwards. The function is not convex, and checking the coordinate directions alone would have missed it.

Note on the lecture slides

Frame 3.19 writes the two-dimensional quadratic as $ax_1^2 + 2bx_1x_2 + cx_2^2 + \dots$ with $Q = \begin{bmatrix} a & b \\ b & c \end{bmatrix}$, that is $x^\top Qx$ *without* the factor $\frac{1}{2}$ of frames 3.17–3.18 and of [Proposition 3.14](#); its Hessian is $2Q$, not Q . The sign test is unaffected, but the factor matters when curvature constants are read from a Hessian, as in [Week 6](#).

3.3.6 Sublevel sets and contour plots

For a level $\alpha \in \mathbb{R}$ the *sublevel set* of f ([Week 1](#)) is $\{x \in \text{dom } f \mid f(x) \leq \alpha\}$: the region enclosed by the contour $f(x) = \alpha$ in the contour plots of that week.

Proposition 3.16 (Sublevel sets). *If f is convex, every sublevel set $\{x \mid f(x) \leq \alpha\}$ is convex.*

Proof. If $f(x) \leq \alpha$ and $f(y) \leq \alpha$, then $f(\theta x + (1 - \theta)y) \leq \theta f(x) + (1 - \theta)f(y) \leq \alpha$. $\square$

This is how convex functions produce convex feasible sets: a constraint $f(x) \leq \alpha$ with f convex carves out a convex region, and [Definition 4.3](#) builds its standard model from such constraints. It is the region $\{f \leq \alpha\}$ that is convex, not the contour $\{f = \alpha\}$. For a convex quadratic with $Q \succ 0$ the sublevel sets are nested ellipses ($d = 2$) or ellipsoids; when Q is singular they can be unbounded. The converse is false: every sublevel set of x^3 is a half-line, yet x^3 is not convex, so convex sublevel sets are a consequence of convexity, not a test for it.

3.3.7 Nonsmooth convex functions and subgradients

Convexity is a geometric property of the graph and the epigraph; differentiability is optional. Alongside the smooth x^2 , e^x and $\frac{1}{2}\|Ax - b\|_2^2$ stand the kinked $|x|$, $\|x\|_1$ and $\max_i(a_i^\top x + b_i)$, and kinks matter because many penalties and robust losses are piecewise linear or built from absolute values. At a kink there is no tangent, but affine global under-estimators survive.

Definition 3.17 (Subgradient and subdifferential). Let f be convex and $x \in \text{dom } f$. A vector $g \in \mathbb{R}^d$ is a **subgradient** of f at x if

$$f(y) \geq f(x) + g^\top(y - x) \quad \text{for all } y \in \text{dom } f.$$

The set of all subgradients of f at x is the **subdifferential** $\partial f(x)$.

Here g is a vector, not a function. The affine function on the right is a global lower bound that touches f at x ; in the language of epigraphs its graph is a **supporting hyperplane** of $\text{epi } f$, a hyperplane through $(x, f(x))$ with $\text{epi } f$ entirely on one side. If f is differentiable at an interior point x of $\text{dom } f$, then $\nabla f(x)$ is a subgradient (the limit step (i) $\Rightarrow$ (ii) in the proof of [Proposition 3.4](#), run along the segment from x to y , uses differentiability at x only), and it is the only one (divide $f(x + \theta v) - f(x) \geq \theta g^\top v$ by $\theta > 0$ and let $\theta \downarrow 0$, for v and for $-v$). Thus $\partial f(x) = \{\nabla f(x)\}$ at a smooth point, a single slope; at a kink there can be many.

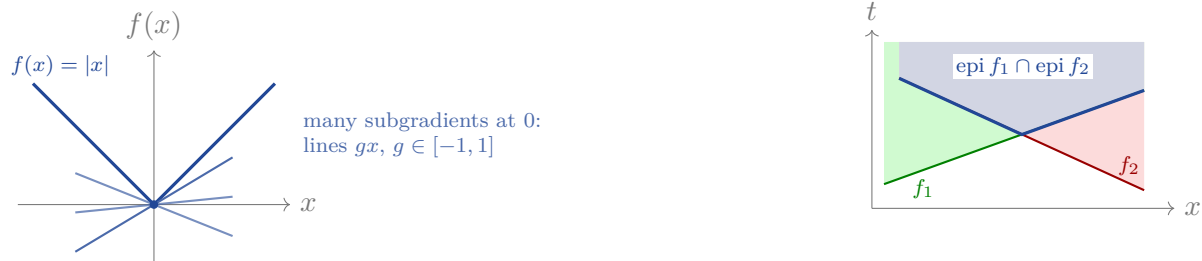


Figure 3.3: Two kinked convex functions. Left: at the kink of $|x|$ every line through the origin with slope in $[-1, 1]$ supports the graph from below. Right: the epigraph of the maximum of two affine functions (dark region) is the intersection of their epigraphs (Proposition 3.19(b)).

Example 3.18 (The subdifferential of the absolute value). For $x > 0$ the function $|x|$ is differentiable with slope 1, and for $x < 0$ with slope -1 . At $x = 0$ the definition asks for all g with $|y| \geq gy$ for every y . Taking $y = 1$ and $y = -1$ shows that $-1 \leq g \leq 1$ is necessary, and it is sufficient because $gy \leq |g| |y| \leq |y|$. Hence

$$\partial|x| = \begin{cases} \{1\}, & x > 0, \\ [-1, 1], & x = 0, \\ \{-1\}, & x < 0. \end{cases}$$

At the kink a whole interval of slopes is valid: every line $y \mapsto gy$ with $g \in [-1, 1]$ passes through the origin and stays below the graph (Figure 3.3, left). In particular the horizontal line is a supporting line, another way of saying that $x = 0$ is the global minimiser.

The last remark is general. Directly from the definition, $0 \in \partial f(x^*)$ means $f(y) \geq f(x^*)$ for all y , the nonsmooth version of Theorem 3.10(a). This is only a first look: Theorem 5.2 uses the condition as an optimality certificate, and Week 7 develops the calculus of subgradients (Section 7.2) and an algorithm built on them.

3.4 Operations that preserve convexity

In practice convexity is rarely rediscovered from scratch by derivative tests. Objectives are *built* from a few primitive convex pieces (affine functions, quadratics, norms, exponentials, absolute values) by operations known to preserve convexity, and recognising a convex objective means tracking its construction, as Section 2.3 did for feasible sets, often through the epigraph. The same constructions return in Proposition 7.4, where sums, affine maps and maxima become rules for computing subgradients.

3.4.1 The basic rules

Proposition 3.19 (Operations that preserve convexity). *Each of the following functions is convex on the domain where its defining expression makes sense (for instance $\{(x, t) \mid t > 0, x/t \in \text{dom } f\}$ in (c)).*

- (a) Nonnegative weighted sum: $\sum_{i=1}^m \alpha_i f_i$, where $f_1, \dots, f_m$ are convex and the weights satisfy $\alpha_1, \dots, \alpha_m \geq 0$ (unrelated to the level α of a sublevel set).
- (b) Pointwise maximum: $f(x) = \max_i f_i(x)$, where $f_1, \dots, f_m$ are convex; likewise the pointwise supremum of an infinite family of convex functions.
- (c) Perspective: the **perspective** $(x, t) \mapsto t f(x/t)$ of a convex f , convex jointly in (x, t) , that is, as a function of the stacked vector (x, t) , for $t > 0$ (here t is a scaling variable, not an epigraph height).

- (d) Affine precomposition: $x \mapsto f(Ax + b)$, where f is convex.
- (e) Partial minimisation: $f(x) = \inf_z g(x, z)$, where z is the variable minimised out (not a test direction), g is jointly convex in (x, z) and the infimum is greater than $-\infty$ for every x .
- (f) Monotone composition: $h \circ g$, where g is convex and $h: \mathbb{R} \rightarrow \mathbb{R}$ is convex and nondecreasing.

Proof. (a) Multiply the chord inequality (3.2) for f_i by $\alpha_i \geq 0$ and add over i . (b) A pair (x, t) satisfies $\max_i f_i(x) \leq t$ exactly when $f_i(x) \leq t$ for every i , so

$$\text{epi}\left(\max_i f_i\right) = \bigcap_i \text{epi } f_i. \quad (3.5)$$

An intersection of convex sets is convex (Proposition 2.17; its proof works verbatim for an arbitrary family of sets), and Theorem 3.3 converts this back into convexity of the maximum.

(c) Take $(x_1, t_1), (x_2, t_2)$ with $t_1, t_2 > 0$, and put $x = \theta x_1 + (1 - \theta)x_2$, $t = \theta t_1 + (1 - \theta)t_2$ and $\lambda = \theta t_1/t \in [0, 1]$. Then $x/t = \lambda(x_1/t_1) + (1 - \lambda)(x_2/t_2)$, so convexity of f gives $tf(x/t) \leq t\lambda f(x_1/t_1) + t(1 - \lambda)f(x_2/t_2) = \theta t_1 f(x_1/t_1) + (1 - \theta)t_2 f(x_2/t_2)$.

(d) With $T(x) = Ax + b$, $\text{epi}(f \circ T) = \{(x, t) \mid (T(x), t) \in \text{epi } f\}$ is the inverse image of $\text{epi } f$ under the affine map $(x, t) \mapsto (Ax + b, t)$, which is convex by Proposition 2.18(b).

(e) Given x_1, x_2 and $\varepsilon > 0$, choose z_1, z_2 with $g(x_i, z_i) \leq f(x_i) + \varepsilon$. Joint convexity of g gives $f(\theta x_1 + (1 - \theta)x_2) \leq g(\theta(x_1, z_1) + (1 - \theta)(x_2, z_2)) \leq \theta f(x_1) + (1 - \theta)f(x_2) + \varepsilon$, and ε was arbitrary. Geometrically, when the infimum is attained $\text{epi } f$ is the projection of $\text{epi } g$ onto (x, t) , convex by Proposition 2.20; otherwise the two sets differ only on the lower boundary.

(f) Convexity of g gives $g(\theta x + (1 - \theta)y) \leq \theta g(x) + (1 - \theta)g(y)$. Applying the nondecreasing h preserves the inequality, and convexity of h then bounds the right-hand side by $\theta h(g(x)) + (1 - \theta)h(g(y))$. $\square$

Rules (b)–(e) are the set operations of Section 2.3 read on epigraphs. Each rule has a characteristic use. *Sums* explain why “loss plus penalty” objectives are convex. The template of statistical learning is $f(x) = \frac{1}{N} \sum_{i=1}^N \ell_i(x) + \lambda r(x)$, where ℓ_i is the loss on record i , r is a regulariser and $\lambda \geq 0$ its weight. With squared losses $\ell_i(x) = (a_i^\top x - b_i)^2$ and $r(x) = \|x\|_1 = \sum_j |x_j|$ the objective is $\frac{1}{N} \|Ax - b\|_2^2 + \lambda \|x\|_1$. Each ℓ_i is the convex function u^2 of an affine function of x (rule (d)), each $|x_j|$ is the absolute value of a coordinate, and all weights are nonnegative, so the objective is convex although it is not differentiable. Up to scaling it is the lasso of Model 9.3; an average of convex losses is the objective of Model 11.1.

Maxima are the robust, worst-case pattern: if $f_i(x)$ is the cost of decision x under scenario i , then $\max_i f_i(x)$ is its worst-case cost, and it is convex whenever each scenario cost is. The right panel of Figure 3.3 shows (3.5) for two affine pieces. For instance $\|x\|_\infty = \max_j |x_j| = \max\{x_1, -x_1, \dots, x_d, -x_d\}$ is a pointwise maximum of $2d$ linear functions, hence convex.

The *perspective* rescales a function: if $f(u)$ is the cost of one unit run at level u , then $tf(x/t)$ is the cost of t identical units sharing the total x . It is the function-side twin of the perspective cone of Definition 2.21 (its epigraph is the perspective cone of $\text{epi } f$ without the adjoined origin), and Week 4 uses it as a modelling pattern. The perspective of $f(u) = \|u\|_2^2$ is the **quadratic-over-linear** function $\|x\|_2^2/t$, jointly convex on $t > 0$. It gives a second, rule-based route to the convexity of the Euclidean norm, in which the epigraph height t of the norm doubles as the scale of the perspective. For $t > 0$, squaring shows that $\|x\|_2 \leq t$ is equivalent to $\|x\|_2^2/t - t \leq 0$, so the part of $\text{epi } \|\cdot\|_2$ with $t > 0$ is a sublevel set of a convex function (a perspective plus an affine term), convex by Proposition 3.16. The edge $t = 0$ adds only the point $(0, 0)$, which keeps the set convex because $\theta(x, t) + (1 - \theta)(0, 0) = (\theta x, \theta t)$ still satisfies $\|\theta x\|_2 \leq \theta t$. The triangle inequality remains the everyday argument.

Affine precomposition explains residual-based losses: $\|Ax - b\|_2^2$, $\|Ax - b\|_2$ and $\|Ax - b\|_1$ are convex functions of the residual $Ax - b$, which is affine in x , so they are convex whatever the data. *Partial minimisation* says that minimising a jointly convex function over some of



Figure 3.4: Left: the Huber function (solid), the infimal convolution of $|x|$ and $\frac{1}{2}x^2$ (dashed). Right: the convex hull (solid) of two parabolas follows f_1 left of -1 and f_2 right of 1 , and is flat in between; with the dashed arcs, their pointwise minimum is the two-well function $(|x| - 1)^2$.

its variables leaves a convex function of the others. It is the engine behind *profiling*, in which nuisance variables are minimised over and no longer reported, behind decomposition schemes that minimise over one block of variables first and then over the rest, and behind the next construction. (The dual functions of [Weeks 5](#) and [12](#) are concave for a different reason: each is a pointwise infimum of functions affine in the multipliers.)

3.4.2 Infimal convolution and the Huber function

The **infimal convolution** of two functions f and g on $\mathbb{R}^d$ is

$$(f \square g)(x) = \inf_z \{f(z) + g(x - z)\} :$$

split x into two parts z and $x - z$, charge them with f and g , and choose the cheapest split. If f and g are convex, then $(x, z) \mapsto f(z) + g(x - z)$ is jointly convex (rules (d) and (a)), so $f \square g$ is convex by rule (e) whenever it is finite.

Example 3.20 (The Huber function). Take $f = |\cdot|$ and $g = \frac{1}{2}(\cdot)^2$ on $\mathbb{R}$, so that $(f \square g)(x) = \inf_z \phi(z)$ with $\phi(z) = |z| + \frac{1}{2}(x - z)^2$, a convex function of z for each fixed x . If $|x| \leq 1$, then $\phi(z) - \phi(0) = |z| - xz + \frac{1}{2}z^2 \geq (1 - |x|)|z| + \frac{1}{2}z^2 \geq 0$, so $\hat{z} = 0$ and the value is $\phi(0) = \frac{1}{2}x^2$. If $x > 1$, then on the open half-line $z > 0$ the function $\phi(z) = z + \frac{1}{2}(x - z)^2$ is differentiable with $\phi'(z) = 1 - (x - z)$, which vanishes at $\hat{z} = x - 1 > 0$; by [Theorem 3.10\(a\)](#) the point $\hat{z}$ minimises ϕ over $z > 0$, so it is a local minimiser of ϕ on $\mathbb{R}$ and, by part (b), a global one, with $\phi(\hat{z}) = (x - 1) + \frac{1}{2} = x - \frac{1}{2}$. The case $x < -1$ is symmetric, with $\hat{z} = x + 1$. Altogether

$$(|\cdot| \square \tfrac{1}{2}(\cdot)^2)(x) = \begin{cases} \frac{1}{2}x^2, & |x| \leq 1, \\ |x| - \frac{1}{2}, & |x| > 1, \end{cases} \quad \hat{z} = \text{sign}(x) \max\{|x| - 1, 0\}.$$

For instance $x = 2.5$ gives $\hat{z} = 1.5$ and the value $1.5 + \frac{1}{2}(1)^2 = 2 = |2.5| - \frac{1}{2}$. This is the **Huber function** ([Figure 3.4](#), left): quadratic near 0, linear in the tails. At $|x| = 1$ the two pieces agree in value ($\frac{1}{2}$) and in slope (± 1), so the function is differentiable everywhere: convolving a kinked function with a smooth one smooths the kink. The inner minimiser $\hat{z}$ shrinks x towards zero by one unit and sets it to zero when $|x| \leq 1$; it is the soft-thresholding operator of [Proposition 9.11](#).

3.4.3 Convex envelopes and convex hulls of functions

When a function is not convex, the epigraph shows how to repair it: the **convex envelope** of f is the largest convex function that lies below f , obtained by replacing $\text{epi } f$ with its convex hull ([Definition 2.4](#)) and reading off the new lower boundary, a principled way to convexify a nonconvex objective. For a family of convex functions $f_1, \dots, f_N$ on $\mathbb{R}^d$ (here N counts the functions, not data records), the same construction applied to the pointwise minimum $\min_i f_i$ (which is not convex in general: the minimum is not among the rules) gives the **convex hull**

of the family, written $\text{conv}\{f_1, \dots, f_N\}$ (the symbol conv of Definition 2.4, now applied to functions),

$$f(x) = \inf \left\{ \sum_{i=1}^N t_i f_i(x_i/t_i) \mid \sum_{i=1}^N x_i = x, \ t_i \geq 0, \ \sum_{i=1}^N t_i = 1 \right\},$$

the largest convex function below $\min_i f_i$. Here the $x_i \in \mathbb{R}^d$ are vectors that split x (not coordinates of x), the t_i are weights, and a term with $t_i = 0$ is read as 0 and requires $x_i = 0$. Writing $y_i = x_i/t_i$ shows what is computed: the smallest weighted average $\sum_i t_i f_i(y_i)$ over all ways of writing x as the weighted average $\sum_i t_i y_i$. Convexity follows from the toolkit: each term is a perspective (rule (c)), the sum is convex (rule (a)), and the infimum over the split, under affine conditions, is a partial minimisation (rule (e)).

Example 3.21 (The hull of two parabolas). Let $f_1(x) = (x+1)^2$ and $f_2(x) = (x-1)^2$ on $\mathbb{R}$, with weights $t_1 = t$ and $t_2 = 1-t$. For fixed t and any y_1, y_2 with $ty_1 + (1-t)y_2 = x$, convexity of the square gives

$$t(y_1+1)^2 + (1-t)(y_2-1)^2 \geq (t(y_1+1) + (1-t)(y_2-1))^2 = (x-1+2t)^2,$$

with equality when $y_1+1 = y_2-1$. It remains to minimise $(x-1+2t)^2$ over $t \in [0, 1]$, where $x-1+2t$ ranges over $[x-1, x+1]$. If $|x| \leq 1$ this interval contains 0 and the minimum is 0 (at $t = (1-x)/2$); if $x > 1$ the minimum is $(x-1)^2$ at $t = 0$; if $x < -1$ it is $(x+1)^2$ at $t = 1$. Hence $\text{conv}\{f_1, f_2\}(x) = (\max\{|x|-1, 0\})^2$: the hull follows each parabola outside $[-1, 1]$ and is flat at 0 in between (Figure 3.4, right). At $x = 1.7$ it equals $f_2(1.7) = 0.49$, whereas at $x = 0.3$ it equals 0 although $\min\{f_1(0.3), f_2(0.3)\} = 0.49$.

3.4.4 Safe and risky patterns, and a worked recognition

Table 3.1 collects the constructions that can be trusted and, opposite them, the patterns that should raise suspicion; the right-hand column mirrors the set operations that break convexity in Section 2.4. “Risky” means that no rule applies and the function must be examined. The first and fourth risky entries are the same pattern, a product of two decision variables, which Section 2.4 called bilinear.

Table 3.1: Common patterns that preserve or break convexity of a function.

Usually safe	Usually risky, or not convex in general
nonnegative weighted sums	products of decision variables such as $x_1 x_2$
maximum of convex functions	ratio of two arbitrary affine functions
affine precomposition	differences of convex functions in general
partial minimisation (profiling)	unrestricted bilinear terms
convex envelope, epigraph reformulation	arbitrary composition of convex with convex
monotone composition: h convex and	dropping monotonicity: $h(u) = u^2$ and
nondecreasing, g convex (for example $e^{g(x)}$)	$g(x) = x - 1$ give the two-well function $(x - 1)^2$

The last row deserves a closer look, because “a convex function of a convex function” sounds safe and is not. Both $h(u) = u^2$ and $g(x) = |x| - 1$ are convex, but h is decreasing for $u < 0$ and g takes negative values. The composition $f(x) = (|x| - 1)^2$ has $f(-1) = f(1) = 0$ and $f(0) = 1$, so $f(0) > \frac{1}{2}f(-1) + \frac{1}{2}f(1)$: the chord inequality fails, and f is the two-well function of Figure 3.1. Replacing h by the convex *nondecreasing* function $(\max\{u, 0\})^2$ restores rule (f) and yields $(\max\{|x| - 1, 0\})^2$, which is exactly the convex hull found in Example 3.21.

Example 3.22 (Log-sum-exp). Is $f(x) = \log \sum_{i=1}^N \exp(a_i^\top x + b_i)$ convex? (Here N counts the exponential terms.) A Hessian computation is messy; the epigraph turns the question into one about a convex set, as in Week 2, in four steps. *Step 1.* Consider first the **log-sum-exp** function

$g(z) = \log \sum_{i=1}^N e^{z_i}$ on $\mathbb{R}^N$. Exponentiating $g(z) \leq t$ (the exponential is increasing) and dividing by $e^t > 0$ gives

$$\text{epi } g = \left\{ (z, t) \mid \log \sum_{i=1}^N e^{z_i} \leq t \right\} = \left\{ (z, t) \mid \sum_{i=1}^N e^{z_i - t} \leq 1 \right\}.$$

Step 2. Each $e^{z_i - t}$ is the exponential, convex by [Example 3.6](#), of an affine function of (z, t) , hence convex by rule (d); the sum is convex by rule (a). *Step 3.* Thus $\text{epi } g$ is a sublevel set of a convex function of (z, t) , a convex set by [Proposition 3.16](#), and g is convex by [Theorem 3.3](#). *Step 4.* Finally $f(x) = g(Ax + b)$, where A has rows $a_i^\top$, is convex by rule (d). With two terms and $z = (0, u)$ the function is $\log(1 + e^u)$, the *softplus* function of [Week 4](#); applied, one term per record, to an affine function of the model parameters, it gives the logistic loss. Log-sum-exp is thus the backbone of logistic regression ([Model 4.18](#) and [Example 6.17](#)) and of its multiclass (softmax) extension.

3.4.5 The convex conjugate

Definition 3.23 (Convex conjugate). The **convex conjugate** of a function f is

$$f^*(y) = \sup_{x \in \text{dom } f} (y^\top x - f(x)),$$

for $y \in \mathbb{R}^d$; the supremum may be $+\infty$, and $\text{dom } f^*$ is the set of y at which it is finite.

In this definition y is a *slope*, not a second point of the domain. An affine function $x \mapsto y^\top x - c$ with slope y lies below f exactly when $c \geq y^\top x - f(x)$ for all x , that is, when $c \geq f^*(y)$. So for $y \in \text{dom } f^*$ the function $y^\top x - f^*(y)$ is the *best affine lower bound of f with slope y* : among all parallel lines under the graph it is the highest one, and when the supremum is attained it touches the graph. The conjugate is convex for *every* f , convex or not. For each fixed x the function $y \mapsto y^\top x - f(x)$ is affine in y , and f^* is the pointwise supremum of this (infinite) family, so rule (b) applies.

Example 3.24 (The conjugate of $x \log x$). Let $f(x) = x \log x$ on $\mathbb{R}_{++}$, so that $f^*(y) = \sup_{x>0} (yx - x \log x)$. The function being maximised is concave in x (its second derivative is $-1/x < 0$), so by [Theorem 3.10](#), applied to its negative, a stationary point is a global maximiser. Setting the derivative $y - (1 + \log x)$ to zero gives $x = e^{y-1}$, and substituting back,

$$f^*(y) = y e^{y-1} - e^{y-1}(y - 1) = e^{y-1} \quad \text{for every } y \in \mathbb{R}.$$

The conjugate of $x \log x$ is an exponential: entropy-type terms in a model become exponential terms in the slope variable y .

Note on the lecture slides

Two forward references in the deck point to the wrong place. Frame 3.30 says that “robust losses in Week 7 use exactly this idea”; Week 7 keeps the absolute value kinked and uses subgradients, and the Huber computation links instead to the soft-thresholding of [Week 9](#). Frame 3.35 announces conjugacy as a forward reference to duality and regularisation, but no later week uses conjugates.

3.5 Recognising convex functions in practice

3.5.1 Epigraph reformulation of a minimax problem

The epigraph is also a modelling move. Consider choosing $x \in \mathbb{R}^d$ to minimise the largest of m affine functions, $\min_x \max_{i=1, \dots, m} (a_i^\top x + b_i)$, for example the worst of m scenario costs. The

Listing 3.1: The minimax model (3.6) in RSOME for $d = 2$ and $m = 3$.

```

import numpy as np
from rsome import ro
from rsome import grb_solver as grb
mod = ro.Model()
x = mod.dvar(2)                # the decision x
t = mod.dvar()                 # the epigraph variable t
A = np.array([[1., -1.], [-2., 1.], [0.5, 0.5]])
b = np.array([1., 2., 0.3])
for i in range(3):
    mod.st(A[i] @ x + b[i] <= t)    # one constraint per affine piece
mod.min(t)
mod.solve(grb)
print(mod.solution.status, mod.get(), x.get())

```

objective is convex by [Proposition 3.19\(b\)](#), but it is kinked, so there is no gradient equation to solve. The epigraph form (3.1) together with (3.5) (the maximum is at most t exactly when every piece is) removes the kinks.

Model 3.25 (Minimax of affine functions in epigraph form).

$$\begin{aligned}
 &\min_{x,t} && t \\
 \text{s.t.} &&& a_i^\top x + b_i \leq t, \quad i = 1, \dots, m.
 \end{aligned} \tag{3.6}$$

The decision variables are the original decision $x \in \mathbb{R}^d$ and one new scalar, the epigraph variable $t \in \mathbb{R}$, an upper bound on every piece. The objective t is linear and the m constraints are linear inequalities in (x, t) , so the feasible set, $\text{epi}(\max_i(a_i^\top x + b_i))$, is a polyhedron ([Definition 2.11](#)) and the model is a linear programme for any LP solver; [Proposition 4.8](#) turns the same move into a family of standard patterns. The only assumption is that the pieces are affine (with convex pieces $f_i(x) \leq t$ the model stays convex but is no longer linear). By (3.1) the optimal x^* solves the original problem and $t^* = \max_i(a_i^\top x^* + b_i)$ is the minimax value; the constraints that hold with equality identify the pieces that attain the maximum at x^* .

[Listing 3.1](#) follows the RSOME skeleton of [Section 1.6](#) and prints the status, the optimal value and x^* . After Gurobi's log lines the output is

```
2 1.0857142857142859 [0.82857143 0.74285714]
```

The status 2 is Gurobi's code for "optimal"; other solvers report the same solution under their own status codes ([Section 1.6](#)). In exact arithmetic $x^* = (29/35, 26/35)$, and all three pieces equal $t^* = 38/35$ there: the optimum sits at a kink where the three pieces meet, which is typical of piecewise-linear objectives. A final caution: the solver found an optimal solution of the model it was given and never checked convexity; the modeller did, by recognising a pointwise maximum of affine functions.

3.5.2 Where the patterns appear, and a recognition checklist

Many objectives in business analytics are assembled from the primitives and rules of this week, which is why recognising a pattern matters more than memorising formulas. Sums and compositions produce loss functions and maximum-likelihood objectives; maxima and epigraph variables produce robust, worst-case models; partial minimisation (profiling) removes latent or nuisance variables; conjugates belong to duality and penalties. Faced with a new objective, the following questions, asked in order, settle most cases.

1. Identify the domain. Is it convex?

2. Look for primitive convex pieces: quadratics, norms, exponentials, absolute values, maxima.
3. Check whether the construction uses only safe operations: sum, maximum, affine map, monotone composition (h convex and nondecreasing with g convex), perspective, projection (partial minimisation).
4. If the function is differentiable, ask whether the first-order inequality, the Hessian test or a one-variable derivative test applies.
5. If the function is nonsmooth, ask for a supporting-hyperplane or subgradient description.
6. If the function looks nonconvex, ask whether an envelope, a reformulation or an approximation is more appropriate.

Structure (steps 1–3) comes before calculus (step 4): it scales to large models and works equally for kinked functions.

Key idea

Convexity is recognised by construction: primitive convex pieces combined by convexity-preserving operations, most of them convex-set operations on epigraphs.

3.6 Summary and looking ahead

A function is convex exactly when its epigraph is a convex set, and minimising f is the same as minimising the height t over $\text{epi } f$. On $\mathbb{R}$ this gives the chord definition, the tangent-line test and the test $f'' \geq 0$; restriction to a line carries them to $\mathbb{R}^d$ as the first-order inequality, the monotone gradient and the Hessian test $\nabla^2 f(x) \succeq 0$. The first-order inequality delivers the payoff promised in [Week 1](#): a stationary point of a convex function is a global minimiser, and there are no spurious local minima. Convex quadratics are settled completely ($\frac{1}{2}x^\top Qx + q^\top x + r$ is convex exactly when $Q \succeq 0$, with $Q = A^\top A$ the canonical case), and convexity survives kinks, with subgradients taking over the role of tangents. In practice convexity is recognised by construction: sums, maxima, affine precomposition, perspectives and partial minimisation are the main rules, and infimal convolutions, envelopes, hulls and conjugates extend the toolbox.

None of this yet handles constraints. [Week 4](#) turns recognition into practice: it combines the convex feasible sets of [Week 2](#) with this week's convex objectives into a standard convex model, shows that local and global minima coincide there too, and writes such models in solver-ready form, where epigraph variables, norm constraints and convex quadratics return as patterns.

3.7 Exercises

Exercise 3.1 (Convex or not?). Decide whether each function is convex on the stated domain and justify your answer briefly: x^2 , e^x , $|x|$ and x^3 on $\mathbb{R}$, and $x \log x$ on $\mathbb{R}_{++}$. Then sketch the epigraph of a convex function on $\mathbb{R}$ and explain why the epigraph must be a convex set.

Exercise 3.2 (Monotone derivatives and the Hessian test). (a) Show that a differentiable convex function on an interval has a nondecreasing derivative. (b) Let $f(x) = \frac{1}{2} \|Ax - b\|_2^2 + \lambda \|x\|_2^2$ with $\lambda \geq 0$. Use the Hessian test to explain why f is convex, and determine when it is strictly convex. (c) For $f(x) = \frac{1}{2} \|Ax - b\|_2^2$, verify the monotonicity of the gradient directly: $(\nabla f(x) - \nabla f(y))^\top (x - y) = \|A(x - y)\|_2^2 \geq 0$.

Exercise 3.3 (Convexity of two-variable quadratics). For $q(x_1, x_2) = ax_1^2 + 2bx_1x_2 + cx_2^2 + d_1x_1 + d_2x_2 + r$, state the two-dimensional convexity test. Apply it to $3x_1^2 - 2x_1x_2 + x_2^2 + 5x_2$ and to $x_1^2 + 6x_1x_2 + 4x_2^2$; for any function that is not convex, give a direction z along which it curves downwards.

Exercise 3.4 (Subdifferentials and a maximum of affine functions). Compute $\partial|x|$ at $x = 0$ and at $x = 2$. Then prove that $f(x) = \max\{2x + 1, -x + 4\}$ is convex using the epigraph-intersection argument, find the point at which f is not differentiable, and determine the subdifferential of f there.

Exercise 3.5 (Log-sum-exp and partial minimisation). (a) Show that $f(x) = \log(e^{a_1^\top x + b_1} + e^{a_2^\top x + b_2})$ is convex by rewriting the epigraph of $g(z) = \log(e^{z_1} + e^{z_2})$ as $\{(z, t) \mid e^{z_1 - t} + e^{z_2 - t} \leq 1\}$ and then applying affine precomposition. (b) Let $g(x, z) = \frac{1}{2} \|x - z\|_2^2 + \|z\|_1$ and $f(x) = \inf_z g(x, z)$. Explain why f is convex even if you do not compute the infimum explicitly.

Exercise 3.6 (Conjugate, infimal convolution and convex hull). (a) Compute the conjugate of $f(x) = e^x$ on $\mathbb{R}$ and compare it with [Example 3.24](#). (b) For $\lambda > 0$, compute and sketch the infimal convolution of $\lambda|x|$ with $\frac{1}{2}x^2$. (c) Compute and sketch the convex hull of $f_1(x) = x^2$ and $f_2(x) = (x - 3)^2$. (Parts (b) and (c) follow the computations of [Examples 3.20](#) and [3.21](#).)

Exercise 3.7 (RSOME mini-lab). Model and solve a problem of the form $\min_x \max_i (a_i^\top x + b_i)$, with data of your choice, by introducing an epigraph variable t as in [Listing 3.1](#); switch between the installed solvers and compare statuses and solutions. What happens when $m = 1$ and $a_1 \neq 0$?

Week 4 Convex Optimisation Modelling and the RSOME Workflow

How do we turn a business problem into a solvable model? The first three weeks prepared the ingredients: [Week 1](#) fixed the vocabulary of optimisation and three running examples (media mix, portfolio, least squares), [Week 2](#) told us which feasible sets are convex, and [Week 3](#) which objectives are convex. This week assembles those pieces into a *modelling language*. Two questions recur throughout: *is the model convex?* and *is the model solver-ready?* The first is a mathematical question about the formulation; the second asks whether the formulation is written in a form that a solver such as Gurobi or MOSEK accepts.

Four problems motivate the week: a diet or media-mix allocation, least squares under side constraints, a risk–return portfolio, and logistic regression. They look unrelated, yet one disciplined workflow handles all four, and we shall classify each into its *model family* before coding it.

[Section 4.1](#) states the workflow, proves why convexity is worth the effort (over a convex feasible set, local minima are global), and fixes the standard convex form and its epigraph reformulation. [Section 4.2](#) climbs the ladder of linear, quadratic and second-order-cone models and their reformulation patterns. [Section 4.3](#) uses logistic regression, geometric programming and a pricing model to show that convexity is a property of the *formulation*, not of the business problem. [Section 4.4](#) views cones as solver-facing primitives and lists common traps, and [Section 4.5](#) codes everything in RSOME.

4.1 From a business problem to a standard convex model

4.1.1 The modelling workflow and its vocabulary

Good modelling is not symbolic guesswork; it is systematic assembly. From a verbal problem statement we (i) separate the *decisions*, which the organisation can choose, from the *data*, which it takes as given; (ii) write the *objective*, which encodes the trade-offs, and the *constraints*, which encode business rules, resources and risk limits; (iii) audit convexity, recognising each term as a known convex block; (iv) reformulate if necessary into a solver-ready form; and (v) call the solver and interpret its status and values, keeping units, signs and domains explicit throughout.

The vocabulary is that of [Definitions 1.4](#) and [1.5](#): decision variable $x \in \mathbb{R}^d$, objective, constraints, feasible point, feasible set $\mathcal{X}$, optimal value and optimiser x^* . Recall also the three statuses of [Definition 1.7](#). A problem is *optimal* when the feasible set is non-empty and a best value is attained, as in $\max\{x_1 : 0 \leq x_1 \leq 1\}$ with $x_1^* = 1$; *infeasible* when the rules contradict one another, as in $\min\{x_1 : x_1 \geq 1, x_1 \leq 0\}$; and *unbounded* when the objective improves forever along a feasible direction, as in $\max\{x_1 : x_1 \geq 0\}$. The status is part of the modelling answer, not an implementation footnote ([Section 4.5.4](#)).

4.1.2 Why convexity is the modelling advantage

[Definition 1.20](#) distinguished a *local* minimiser of f over $\mathcal{X}$ — a point $x^* \in \mathcal{X}$ with $f(x^*) \leq f(x)$ for all $x \in \mathcal{X}$ within some radius $\rho > 0$ of x^* (called r there and ρ in [Week 3](#); this week r has other jobs) — from a *global* one. A solver can only ever inspect a neighbourhood of its current point, so in general it can certify local optimality at best. [Theorem 3.10](#) showed that for a convex *function* this is enough: a stationary point of a differentiable convex function is a global minimiser (part (a)), and every local minimiser of a convex function, smooth or not, is global (part (b)). The following theorem carries part (b) over to a convex *feasible set*, with the same segment argument, and it is the reason we insist on convex models.



Figure 4.1: Left: a convex objective over a convex feasible set; the lowest contour that touches the set does so at the global minimiser x^* . Right: the proof of [Theorem 4.1](#): a better feasible point y would make the points z_θ feasible, better than x^* and arbitrarily close to it.

Theorem 4.1 (Local minima are global). *Let $\mathcal{X} \subseteq \mathbb{R}^d$ be a convex set and let f be convex on $\mathcal{X}$. If x^* is a local minimiser of f over $\mathcal{X}$, then $f(x^*) \leq f(y)$ for every $y \in \mathcal{X}$.*

Proof. Suppose, to the contrary, that some feasible y has $f(y) < f(x^*)$. For $\theta \in [0, 1]$ put $z_\theta = (1 - \theta)x^* + \theta y$. Because $\mathcal{X}$ is convex, the whole segment stays feasible: $z_\theta \in \mathcal{X}$. Because f is convex,

$$f(z_\theta) \leq (1 - \theta)f(x^*) + \theta f(y) < f(x^*) \quad \text{for every } \theta \in (0, 1].$$

Finally $\|z_\theta - x^*\|_2 = \theta \|y - x^*\|_2$, so for $\theta \leq \rho / \|y - x^*\|_2$ the point z_θ lies within radius ρ of x^* , is feasible, and is strictly better — contradicting local optimality. $\square$

The proof uses both halves of the hypothesis ([Figure 4.1](#)): convexity of the set keeps the segment feasible, and convexity of the function makes its points better than x^* . This is why the convexity audit covers the objective *and* every constraint, and why a point that a solver certifies as locally optimal for a convex model is the best decision available.

Convexity rules out spurious local minima but does not make the minimiser unique: a linear objective may be minimised along a whole edge of a polyhedron. A guarantee of uniqueness needs *strict* convexity ([Definition 3.2](#): the convexity inequality is strict for $x \neq y$ and $0 < \theta < 1$). [Week 3](#) gave the midpoint argument informally; we record it as a proposition for later reference.

Proposition 4.2 (Strict convexity gives uniqueness). *If f is strictly convex on the convex set $\mathcal{X}$, then f has at most one minimiser over $\mathcal{X}$.*

Proof. Suppose $x_1 \neq x_2$ both attain the minimum value f^* . The midpoint $\frac{1}{2}(x_1 + x_2)$ is feasible because $\mathcal{X}$ is convex, and strict convexity gives $f(\frac{1}{2}x_1 + \frac{1}{2}x_2) < \frac{1}{2}f(x_1) + \frac{1}{2}f(x_2) = f^*$, a feasible point with a value below the minimum — a contradiction. $\square$

In particular a convex quadratic $\frac{1}{2}x^\top Qx + q^\top x$ with $Q \succ 0$, for instance $Q = A^\top A + \mu I$ with $\mu > 0$ ([Proposition 3.14](#)), has exactly one minimiser over $\mathbb{R}^d$, and over any non-empty closed convex feasible set (closed: it contains all its boundary points, as does every set $\{x : f_i(x) \leq 0, i = 1, \dots, m\}$ with each f_i continuous on $\mathbb{R}^d$), so the recommendation does not depend on the solver or on its starting point. As noted in [Week 3](#), the proposition says only “at most one” (e^x on $\mathbb{R}$ has no minimiser).

4.1.3 The standard convex optimisation form

To make “the model is convex” checkable we agree on one canonical way of writing a model.

Definition 4.3 (Standard convex optimisation problem). A **convex optimisation problem in standard form** is

$$\begin{aligned} \min \quad & f_0(x) \\ \text{s.t.} \quad & f_i(x) \leq 0, \quad i = 1, \dots, m, \\ & a_j^\top x = b_j, \quad j = 1, \dots, p, \end{aligned} \tag{4.1}$$

where the objective f_0 and the inequality-constraint functions $f_1, \dots, f_m$ are convex, and the equality constraints are affine. Domain restrictions (the set on which all the f_i are defined, such as $x > 0$ for a model containing $\log x$) are part of the model and must be stated explicitly.

Definition 1.4 wrote constraints generically as $g_i(x) \leq 0$ and $h_j(x) = 0$. Here and in [Week 5](#) the inequality functions are f_i , numbered together with the objective f_0 ; the equalities are written $a_j^\top x = b_j$ since they must be affine, and [Week 5](#) stacks the p of them into one matrix equation $Ax = b$. The form is less restrictive than it looks: $g(x) \leq c$ is $g(x) - c \leq 0$; a constraint $g(x) \geq c$ with g concave is $c - g(x) \leq 0$; and maximising a concave objective is minimising its convex negative. The form forbids what typically breaks convexity: minimising a concave function, a constraint $g(x) \geq c$ with g convex, and nonlinear equalities.

Proposition 4.4 (The feasible set is convex). *The feasible set $\mathcal{X}$ of a problem in the standard form (4.1) is convex.*

Proof. Each set $\{x : f_i(x) \leq 0\}$ is a sublevel set of a convex function and is therefore convex: if $f_i(x) \leq 0$ and $f_i(y) \leq 0$, then $f_i(\theta x + (1 - \theta)y) \leq \theta f_i(x) + (1 - \theta)f_i(y) \leq 0$ ([Week 3](#)). Each set $\{x : a_j^\top x = b_j\}$ is a hyperplane, which is convex ([Definition 2.10](#)). The feasible set is the intersection of these $m + p$ sets and the common domain of the f_i , which is convex because the domain of a convex function is ([Definition 3.2](#)); intersections preserve convexity ([Proposition 2.17](#)). $\square$

Combined with [Theorem 4.1](#), every model in standard form has the local-equals-global property. Equalities must be affine because the argument above does not extend to them: for a convex h , the set $\{x : h(x) = 0\}$ is the *boundary* of a sublevel set and is usually not convex. The circle $\{x : x_1^2 + x_2^2 = 1\}$ contains $(1, 0)$ and $(-1, 0)$ but not their midpoint.

4.1.4 The epigraph form

Solvers are built around a small number of constraint types, and they are most transparent when the objective is linear. The device that moves any convex objective into the constraints was previewed in [Equation \(3.1\)](#); it is the single most reusable modelling idea of the unit.

Proposition 4.5 (Epigraph form). *Let $\mathcal{X} \subseteq \mathbb{R}^d$ and $f_0 : \mathcal{X} \rightarrow \mathbb{R}$. The problems*

$$\begin{array}{ll} \min_x & f_0(x) \\ \text{s.t.} & x \in \mathcal{X} \end{array} \quad \Longleftrightarrow \quad \begin{array}{ll} \min_{x,t} & t \\ \text{s.t.} & f_0(x) \leq t, \quad x \in \mathcal{X} \end{array} \quad (4.2)$$

have the same optimal value, and x^* solves the first if and only if $(x^*, f_0(x^*))$ solves the second. At every optimal solution (x^*, t^*) of the second problem the new constraint is tight, $t^* = f_0(x^*)$.

Proof. Fix a feasible x . The values of t allowed by $f_0(x) \leq t$ are $[f_0(x), \infty)$, and the smallest is $t = f_0(x)$. Minimising over t first therefore turns the second problem into the first. If $t^* > f_0(x^*)$ at a claimed optimum, lowering t to $f_0(x^*)$ would stay feasible and improve the objective. $\square$

The scalar t is called an **epigraph variable**: the pairs (x, t) with $f_0(x) \leq t$ form precisely $\text{epi } f_0$ ([Definition 3.1](#)), a convex set exactly when f_0 is a convex function ([Theorem 3.3](#)). The reformulated problem is therefore again convex, now with the *affine* objective t and one more convex constraint (left panel of [Figure 4.2](#)). Nothing has been gained mathematically, but the nonlinearity now sits in a constraint, where it can be rewritten further until every piece is of a type the solver recognises.

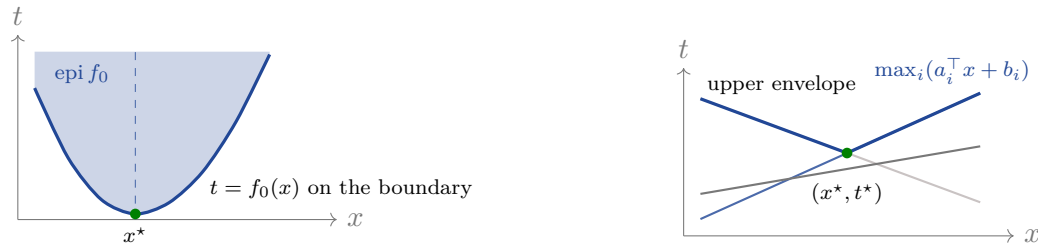


Figure 4.2: Left: the epigraph form. The feasible pairs (x, t) form $\text{epi } f_0$ (shaded); minimising t slides down to its lowest point, on the graph above the minimiser x^* . Right: three affine functions and their pointwise maximum (thick). The region above the envelope is an intersection of halfspaces $a_i^\top x + b_i \leq t$, so minimising the maximum needs only linear constraints.

Key idea

Minimising $f_0(x)$ is the same as pushing a ceiling t down onto the graph of f_0 : minimise t subject to $f_0(x) \leq t$. The RSOME code of this unit always works this way, with an affine objective and every nonlinear convex piece in a constraint.

4.2 Model families and reformulation patterns

Many practical models reduce to a small number of reusable *families*, and classifying the family first tells us how to code the model and which solver to call. The families form a ladder: *linear* models (which include maxima and the ℓ_1 and ℓ_∞ norms); *quadratic and second-order-cone* models (least squares, risk, ℓ_2 norms); and *exponential-cone* models (softplus and logistic losses), which after a change of variables also accommodate *geometric* programmes built from products and powers. This subsection covers the first two rungs and [Section 4.3](#) the rest.

4.2.1 The linear programme

Model 4.6 (Linear programme, LP).

$$\begin{aligned} \min_{x \in \mathbb{R}^d} \quad & c^\top x \\ \text{s.t.} \quad & Ax \leq b, \\ & Cx = e. \end{aligned} \tag{4.3}$$

The decision $x \in \mathbb{R}^d$ lists the levels of d activities (dollars spent, units shipped); the data are the cost vector c and the pairs (A, b) and (C, e) , the equality right-hand side being called e because d is the dimension. The objective is affine, c_k being the cost per unit of activity k ; each row of $Ax \leq b$ is one business rule (a budget, a capacity, a minimum commitment) and each row of $Cx = e$ a balance that must hold exactly. The feasible set is a polyhedron ([Definition 2.11](#)), so an LP is the standard form (4.1) with every f_i affine. The formulation assumes *proportionality* (cost and resource use scale linearly with each activity), *additivity* (no interactions) and *divisibility* (fractional levels are meaningful). LPs with millions of variables are solved routinely by Gurobi or MOSEK. At an optimal plan x^* , the constraints that hold with equality (the *active* constraints) are the candidates for the rules that limit performance; [Section 5.5](#) attaches a price to each resource constraint, and the rules that genuinely limit performance are the active ones with a positive price.

Example 4.7 (The media-mix model, classified). In [Model 1.1](#) a marketer splits \$10 000 between search, social and video advertising: $\max 4x_1 + 3x_2 + 2x_3$ subject to $x_1 + x_2 + x_3 \leq 10\,000$, $x_1 \geq 2000$, $x_2 \leq 4000$, $x \geq 0$, the coefficients being conversions per dollar. Maximising $c^\top x$ is minimising $(-c)^\top x$, and $x_1 \geq 2000$ is $-x_1 \leq -2000$, so the model is (4.3) with $c = -(4, 3, 2)$,

no equalities, and six rows in $Ax \leq b$: an affine objective over a polyhedron, hence an LP, and any LP solver will do. Its solution is $x^* = (10\,000, 0, 0)$ with 40\,000 conversions: search has the highest return per dollar and no cap, so it takes the whole budget; only the budget and the bounds $x_2, x_3 \geq 0$ are active.

4.2.2 Reformulations that stay linear

A *reformulation* replaces a raw objective or constraint by an equivalent one that makes the convex structure visible to the solver, usually by introducing an auxiliary variable with a clear interpretation (a ceiling, a worst case, an absolute deviation). The epigraph form of [Proposition 4.5](#) is the template. Applied to maxima, absolute values and the ℓ_1 and ℓ_∞ norms it yields models that look nonlinear but are linear programmes. Throughout, $A \in \mathbb{R}^{N \times d}$ and $b \in \mathbb{R}^N$ are data and $r = Ax - b$ is the vector of N residuals; m counts the affine pieces in (i).

Proposition 4.8 (LP-representable patterns). (i) Maximum of affine functions. $\max_i v_i \leq t$ if and only if $v_i \leq t$ for every i . Hence

$$\min_x \max_{i=1,\dots,m} (a_i^\top x + b_i) \iff \min_{x,t} t \quad \text{s.t.} \quad a_i^\top x + b_i \leq t, \quad i = 1, \dots, m.$$

(ii) Absolute value and ℓ_1 norm. $|u| \leq t$ if and only if $-t \leq u \leq t$. Since $\|r\|_1 = \sum_{i=1}^N |r_i|$,

$$\min_x \|Ax - b\|_1 \iff \min_{x,t \in \mathbb{R}^N} \mathbf{1}^\top t \quad \text{s.t.} \quad -t \leq Ax - b \leq t.$$

(iii) ℓ_∞ norm. Since $\|r\|_\infty = \max_i |r_i|$,

$$\min_x \|Ax - b\|_\infty \iff \min_{x,t \in \mathbb{R}} t \quad \text{s.t.} \quad -t\mathbf{1} \leq Ax - b \leq t\mathbf{1}.$$

Proof. A maximum is at most t exactly when each of its terms is, which with [Proposition 4.5](#) gives (i). For (ii), $|u| = \max\{u, -u\}$, so $|u| \leq t$ means $u \leq t$ and $-u \leq t$; give each component its own ceiling $t_i \geq |r_i|$ and minimise $\sum_i t_i$, every ceiling being tight at an optimum. For (iii), $\|r\|_\infty$ is the maximum of the $2N$ affine functions $\pm r_i$ of x , so (i) applies with one common ceiling t . $\square$

Pattern (i) is the operational version of the rule that a pointwise maximum of convex functions is convex ([Proposition 3.19](#)): the epigraph of the maximum is the intersection of the epigraphs of the pieces, here halfspaces (right panel of [Figure 4.2](#)); [Model 3.25](#) was a first instance. In (iii) the variable t is the *worst-case* residual, and the residual vector is forced into a box-shaped tolerance band. In (ii) each t_i is the absolute deviation of observation i ; this is least-absolute-deviations regression, which [Week 7](#) uses as a solver benchmark. The price of linearity is size $(d + N)$ variables and $2N$ constraints for the ℓ_1 model), which LP solvers handle easily.

The three norms weigh large errors differently. Fitting a single number x to the observations $b = (1, 2, 6)$, so that $A = \mathbf{1}$, the ℓ_1 criterion returns the *median* $x^* = 2$, the ℓ_2 (least-squares) criterion the *mean* 3 and the ℓ_∞ criterion the *mid-range* 3.5: the outlying observation 6 pulls the ℓ_∞ answer hardest and the ℓ_1 answer not at all, so the choice of norm is a modelling decision.

4.2.3 Two linear models: transportation and the Chebyshev centre

Model 4.9 (Transportation). A firm ships one product from m warehouses with supplies s_i to n stores with demands d_j , at unit cost c_{ij} from warehouse i to store j :

$$\begin{aligned} \min \quad & \sum_{i=1}^m \sum_{j=1}^n c_{ij} x_{ij} \\ \text{s.t.} \quad & \sum_{j=1}^n x_{ij} \leq s_i, \quad i = 1, \dots, m \quad (\text{ship at most the supply}), \\ & \sum_{i=1}^m x_{ij} = d_j, \quad j = 1, \dots, n \quad (\text{meet every demand}), \\ & x_{ij} \geq 0. \end{aligned} \tag{4.4}$$

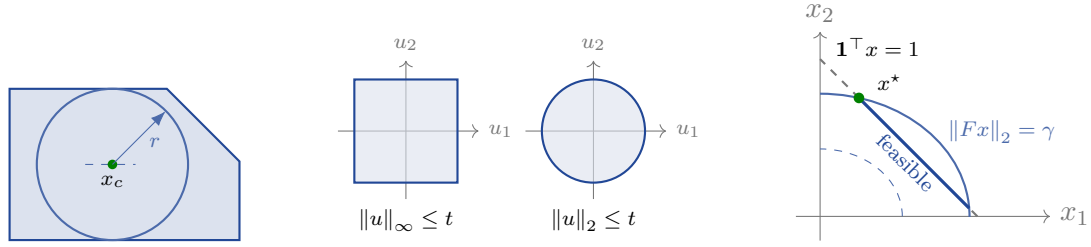


Figure 4.3: Left: the Chebyshev centre of [Example 4.11](#); the largest inscribed ball has radius 1.25, and every centre on the dashed segment is optimal. Centre: the ℓ_∞ ball is a box, cut out by linear inequalities; the ℓ_2 ball is round and needs the second-order cone. Right: a two-asset risk-budget portfolio ([Model 4.15](#)), schematically: contours of the risk $\|Fx\|_2$ ($F^\top F$ the covariance matrix, γ the risk budget), the feasible part of the budget line $\mathbf{1}^\top x = 1$ (thick), and its end x^* with the higher expected return.

The decision is a whole *matrix* of shipment quantities, so there are mn variables and $m + n$ functional constraints. (Here m and n count warehouses and stores and d_j is a demand; they are not the m and d of (4.1).) Every expression is affine, so the model is an LP: the same family as the media-mix model, at industrial scale. It assumes that cost is proportional to quantity (no fixed charge for opening a lane) and that shipments are divisible. Summing the constraints shows that the model is feasible if and only if total supply covers total demand, $\sum_i s_i \geq \sum_j d_j$ (the plan $x_{ij} = s_i d_j / \sum_k s_k$ is then feasible), so an “infeasible” status has an immediate business reading. In an optimal plan the positive x_{ij}^* are the lanes to use, and a supply constraint with slack identifies a warehouse with spare stock. Diet, blending, resource-allocation, maximal-flow and supply-chain planning models are relatives (Luenberger and Ye, *Linear and Nonlinear Programming*, §2.2).

The second model, a classic from Boyd and Vandenberghe, shows that a question which does not look linear can become an LP once its constraint is rewritten. Given a polyhedron $\{x : a_i^\top x \leq b_i, i = 1, \dots, m\}$, we seek the largest Euclidean ball $B(x_c, r) = \{x_c + ru : \|u\|_2 \leq 1\}$ ([Definition 2.14](#)) that fits inside it. The ball satisfies constraint i when *every* one of its points does, that is, when $\sup_{\|u\|_2 \leq 1} a_i^\top (x_c + ru) \leq b_i$. By the Cauchy–Schwarz inequality ([Proposition 1.11](#)) $a_i^\top u \leq \|a_i\|_2 \|u\|_2 \leq \|a_i\|_2$, with equality at $u = a_i / \|a_i\|_2$, so for $r \geq 0$ the supremum equals $a_i^\top x_c + r \|a_i\|_2$. Infinitely many constraints, one per point of the ball, collapse into one.

Model 4.10 (Chebyshev centre of a polyhedron).

$$\begin{aligned} \max_{x_c \in \mathbb{R}^d, r \in \mathbb{R}} \quad & r \\ \text{s.t.} \quad & a_i^\top x_c + r \|a_i\|_2 \leq b_i, \quad i = 1, \dots, m. \end{aligned} \quad (4.5)$$

The variables are the centre x_c and the radius r (here r is a radius, not a residual). The numbers $\|a_i\|_2$ are *data*, so each constraint is affine in (x_c, r) and the model is an LP even though a norm appears in it. The optimal x_c^* is the most interior operating point, and r^* is the largest perturbation, in any direction, that every rule can absorb: maximal simultaneous slack. The derivation assumed $r \geq 0$. If the polyhedron is non-empty then $r = 0$ is feasible and $r^* \geq 0$ automatically, but adding $r \geq 0$ is good practice: without it an empty polyhedron yields a negative “radius” instead of an infeasible status.

Example 4.11 (A Chebyshev centre by hand). The polygon of [Figure 4.3](#) (left) is $\{x : 0.6 \leq x_1 \leq 4.4, 0.5 \leq x_2 \leq 3, x_1 + x_2 \leq 6.2\}$; its first four constraint vectors have norm 1 and the last has norm $\sqrt{2}$. Model (4.5) therefore maximises r subject to $r \leq x_{c,1} - 0.6$, $r \leq x_{c,2} - 0.5$, $r \leq 4.4 - x_{c,1}$, $r \leq 3 - x_{c,2}$ and $x_{c,1} + x_{c,2} + \sqrt{2}r \leq 6.2$. The second and fourth inequalities add to $2r \leq 2.5$, and $r^* = 1.25$ is attained with $x_{c,2} = 1.75$ and any $x_{c,1}$ between 1.85 and $6.2 - 1.75 - 1.25\sqrt{2} \approx 2.68$. The largest ball touches the top and bottom edges, but its centre is not unique: an LP objective is not strictly convex, so [Proposition 4.2](#) does not apply.

4.2.4 Quadratic and second-order-cone models

Definition 4.12 (Convex quadratic programme). A **convex quadratic programme (QP)** minimises a convex quadratic over a polyhedron:

$$\begin{aligned} \min_x \quad & \frac{1}{2}x^\top Qx + q^\top x \\ \text{s.t.} \quad & Ax \leq b, \quad Cx = e, \quad Q \succeq 0. \end{aligned} \quad (4.6)$$

This rung of the ladder admits curvature: $Q \succeq 0$ is exactly the condition for the objective to be convex ([Proposition 3.14](#)), and an LP is the special case $Q = 0$.

Model 4.13 (Constrained least squares).

$$\begin{aligned} \min_{x \in \mathbb{R}^d} \quad & \|Ax - b\|_2^2 \\ \text{s.t.} \quad & x \in \mathcal{C}. \end{aligned} \quad (4.7)$$

This is the least-squares model of [Model 1.3](#) with side constraints, written without the factor $1/N$ of [Week 1](#) or the $\frac{1}{2}$ used from [Week 3](#) on; positive factors change the optimal value, not the minimiser. Here x holds the coefficients of a fitted model, row i of A the features $a_i^\top$ of observation i , and b the N responses. The convex set $\mathcal{C}$ encodes what the business knows or requires: a box $\ell \leq x \leq u$, the nonnegative orthant (effects that cannot be negative), the simplex of [Definition 2.12](#) (weights of a blend or of a forecast combination), or another policy set. Since $\|Ax - b\|_2^2 = x^\top A^\top Ax - 2b^\top Ax + b^\top b$, the objective is a convex quadratic with $Q = 2A^\top A \succeq 0$ ([Example 3.13](#)), and for polyhedral $\mathcal{C}$ the model is a QP. If the columns of A are linearly independent then $A^\top A \succ 0$ ([Proposition 1.13](#)) and the fit is unique by [Proposition 4.2](#). There is in general no closed-form solution, since the active constraints are not known in advance; a QP solver finds them. This template is the bridge to statistics and machine learning in [Weeks 8](#) and [9](#).

When the ℓ_2 norm appears in a *constraint*, a new primitive is needed.

Definition 4.14 (Second-order-cone constraint and programme). A **second-order-cone (SOC) constraint** on the variables (x, t) has the form

$$\|Ax - b\|_2 \leq t. \quad (4.8)$$

A **second-order-cone programme (SOCP)** has an affine objective and linear or SOC constraints.

The constraint says that the affine image $(Ax - b, t)$ of the variables lies in the norm cone $\{(u, t) : \|u\|_2 \leq t\}$ of [Definition 2.15](#), so it is convex by [Proposition 2.18](#). The contrast with pattern (iii) of [Proposition 4.8](#) is geometric ([Figure 4.3](#), centre): the ℓ_∞ ball is a box with flat faces, described exactly by $2N$ linear inequalities, whereas the ℓ_2 ball is round, so no finite list of linear inequalities describes it and the solver needs the cone itself as a primitive. Squared norms belong to the same family, because $\|u\|_2^2 \leq t$ holds if and only if $\|(2u, t - 1)\|_2 \leq t + 1$ (square both sides). This is why risk bounds and many fitting models land in the norm/SOCP family.

Model 4.15 (Portfolio with a risk budget).

$$\begin{aligned} \max_{x \in \mathbb{R}^d} \quad & \mu^\top x \\ \text{s.t.} \quad & \|Fx\|_2 \leq \gamma, \\ & \mathbf{1}^\top x = 1, \quad x \geq 0. \end{aligned} \quad (4.9)$$

Here x_k is the fraction of wealth placed in asset k ([Model 1.2](#) called the weights w , a symbol needed for another purpose below), μ is the vector of expected returns, Σ the covariance matrix of the returns, and F any matrix with $F^\top F = \Sigma$: for example, when $\Sigma \succ 0$, $F = L^\top$, where $\Sigma = LL^\top$ with L lower triangular is the Cholesky factorisation computed by `numpy.linalg.cholesky`. A

covariance matrix is only guaranteed to be positive semidefinite ([Week 3](#)), and for a singular Σ that routine fails; a sample covariance $\Sigma = \tilde{R}^\top \tilde{R} / (T - 1)$ built from T centred return observations (the rows of $\tilde{R}$) has the factor $F = \tilde{R} / \sqrt{T - 1}$ directly. Then $\|Fx\|_2^2 = x^\top F^\top Fx = x^\top \Sigma x$ is the variance of the portfolio return and $\|Fx\|_2$ its standard deviation (volatility). The objective maximises expected return; $\mathbf{1}^\top x = 1$ enforces full investment, $x \geq 0$ forbids short selling, and the SOC constraint caps the volatility at the budget γ (so the variance is capped at γ^2). The model is an SOCP, solved by Gurobi or MOSEK. It takes μ and Σ as known; in practice they are estimates, and the recommendation is only as good as they are. At an optimum the risk constraint is typically active. Its Lagrange multiplier ([Definition 5.7](#)) measures the expected return that a slightly larger budget would buy, the reading that [Definition 5.21](#) gives to the resources of a linear programme.

Note on the lecture slides

The symbol γ changes role between weeks: in [Model 1.2](#), $\max \mu^\top w - \gamma w^\top \Sigma w$, it is a risk-aversion *weight*; on frame 4.23 it is a risk *budget* (a penalty and a hard cap for the same risk-return trade-off). This week's deck presents only the risk-budget form.

Example 4.16 (Two assets). Let $\mu = (0.12, 0.06)$, $\Sigma = \begin{pmatrix} 0.04 & 0.006 \\ 0.006 & 0.01 \end{pmatrix}$ and $\gamma = 0.12$ (asset volatilities 20% and 10%, budget 12%). Writing $x = (\xi, 1 - \xi)$ with $0 \leq \xi \leq 1$, the variance $0.038\xi^2 - 0.008\xi + 0.01$ is at most $\gamma^2 = 0.0144$ exactly when $\xi \leq 0.4615$, and the expected return $0.06 + 0.06\xi$ increases with ξ . The optimum is therefore the end of the feasible segment at which the risk budget binds (compare [Figure 4.3](#), right, drawn for a case in which asset 2 has the higher return): $x^* = (0.4615, 0.5385)$, with expected return 8.77% and volatility exactly 12%. The lowest volatility on the budget line is 9.79%, so any $\gamma < 0.0979$ makes the model infeasible: a fact about the risk target, not a solver failure.

Model 4.17 (Soft-margin linear classifier, SVM). Given data (a_i, y_i) , $i = 1, \dots, N$, with features $a_i \in \mathbb{R}^d$ and labels $y_i \in \{-1, +1\}$,

$$\begin{aligned} \min_{w \in \mathbb{R}^d, b \in \mathbb{R}, \xi \in \mathbb{R}^N} \quad & \frac{1}{2} \|w\|_2^2 + C \sum_{i=1}^N \xi_i \\ \text{s.t.} \quad & y_i(a_i^\top w + b) \geq 1 - \xi_i, \quad i = 1, \dots, N, \\ & \xi_i \geq 0. \end{aligned} \tag{4.10}$$

A new case with features a is classified by the sign of the score $a^\top w + b$. Here w is the weight vector, b a scalar intercept (which is why the labels are called y_i) and $C > 0$ a penalty weight. Observation i is classified correctly, with a safety margin, when $y_i(a_i^\top w + b) \geq 1$, and the **hinge loss** $\max\{0, 1 - y_i(a_i^\top w + b)\}$ measures by how much that fails. The constraints make ξ_i at least each of the two affine pieces of this maximum, so ξ_i is an epigraph variable for the hinge loss — pattern (i) of [Proposition 4.8](#), reused — and because $C > 0$ it equals the hinge loss at an optimum. The hyperplanes $a^\top w + b = \pm 1$ are a distance $2/\|w\|_2$ apart, so the term $\frac{1}{2} \|w\|_2^2$ rewards a wide, robust margin, while C prices the violations. A convex quadratic objective with linear constraints makes the model a convex QP. In the solution w^* is an interpretable linear score (credit approval, churn prediction, fraud flags; see Luenberger and Ye, §11.6), and $\xi_i^* > 0$ marks the observations inside the margin ($\xi_i^* > 1$: misclassified). The hinge loss returns as a nonsmooth objective in [Week 7](#).

4.3 Convexity by modelling

The models so far were convex as written. We now meet a convex model that is neither an LP nor a QP, and then two problems that become convex only after a change of decision variables.

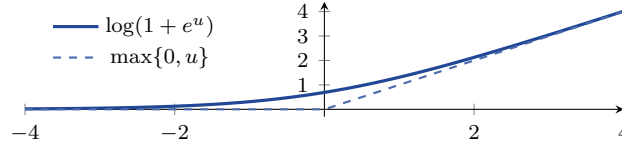


Figure 4.4: Softplus is a smooth, convex version of $\max\{0, u\}$; the gap never exceeds $\log 2 \approx 0.69$.

4.3.1 Logistic regression and the softplus block

Model 4.18 (Regularised logistic regression). Given features $a_i \in \mathbb{R}^d$ and labels $b_i \in \{-1, +1\}$, $i = 1, \dots, N$, and a weight $\lambda \geq 0$,

$$\min_{\theta \in \mathbb{R}^d} \sum_{i=1}^N \log(1 + \exp(-b_i a_i^\top \theta)) + \lambda \|\theta\|_2^2. \quad (4.11)$$

The parameter vector $\theta \in \mathbb{R}^d$ plays the role of the decision x ; it is not the scalar weight $\theta \in [0, 1]$ of a convex combination used in [Week 3](#) and in the proofs of [Section 4.1](#). The model posits that a case with features a has label $+1$ with probability $1/(1 + e^{-a^\top \theta})$; the probability of the observed label b_i is then $1/(1 + \exp(-b_i a_i^\top \theta))$, so the first term of (4.11) is the negative log-likelihood of the data and minimising it is maximum-likelihood estimation. The second term regularises: it shrinks θ towards zero, and for $\lambda > 0$ it makes the objective strictly convex, so the fitted model is unique ([Proposition 4.2](#)). Convexity follows from [Proposition 3.19](#): the **softplus** function $u \mapsto \log(1 + e^u)$ is log-sum-exp of the pair $(0, u)$ ([Example 3.22](#)), each summand composes it with an affine function of θ , and sums of convex functions are convex. But the objective is neither affine nor quadratic: there is no closed-form minimiser, and no LP or QP solver accepts it. The labels are the ± 1 labels of [Model 4.17](#) (y_i there, as b was the intercept); (a_i, b_i) is the data convention of the unit ([Weeks 6, 9 and 11](#)).

Note on the lecture slides

Frame 4.25 gives the reason the other way round: the labels are b_i , not y_i , “because b is the SVM intercept”. That clash explains the y_i of the SVM, not the b_i used here.

Softplus and log-sum-exp are standard building blocks that act as *smooth maxima* ([Figure 4.4](#)). A sum of n exponentials lies between its largest term and n times that term, so $\max_i u_i \leq \log \sum_{i=1}^n e^{u_i} \leq \max_i u_i + \log n$. Log-sum-exp is thus a smooth maximum of many terms; softplus, its value at the pair $(0, u)$, is a smooth $\max\{0, u\}$ that exceeds it by at most $\log 2$; and the logistic loss is a smooth relative of the hinge loss of [Model 4.17](#). To hand softplus to a solver we rewrite its epigraph.

Proposition 4.19 (Softplus epigraph). *For real u and t : $t \geq \log(1 + e^u) \iff e^{u-t} + e^{-t} \leq 1$.*

Proof. As exp is increasing, $t \geq \log(1 + e^u)$ if and only if $e^t \geq 1 + e^u$; divide by $e^t > 0$. $\square$

The right-hand side is a sum of exponentials of affine functions of (u, t) bounded by one: a convex constraint by construction, and exactly the rewriting of [Example 3.22](#) with $z = (0, u)$. Solvers such as MOSEK accept constraints of this exponential type directly ([Section 4.4.1](#)), and RSOME generates the rewriting from `rso.softplus(u) <= t`: rewriting an epigraph until every piece is a recognised convex block is what the modelling layer automates. Applying [Proposition 4.5](#) to each summand of (4.11) gives the solver-facing form

$$\begin{aligned} \min_{\theta \in \mathbb{R}^d, t \in \mathbb{R}^N, r \in \mathbb{R}} \quad & \sum_{i=1}^N t_i + \lambda r \\ \text{s.t.} \quad & \log(1 + \exp(-b_i a_i^\top \theta)) \leq t_i, \quad i = 1, \dots, N, \\ & \|\theta\|_2^2 \leq r, \end{aligned} \quad (4.12)$$

in which t_i is a ceiling on the loss of observation i and r a ceiling on $\|\theta\|_2^2$ (here r is neither a residual nor a radius). The objective is now affine and the nonlinear convex pieces sit in constraints; it is still the same logistic-regression problem, and only the solver-facing representation has changed. An exponential-cone solver returns θ^* , in which $\theta_k^* > 0$ means that feature k raises the probability of the label $+1$. For large N the same model is minimised by the gradient methods of [Weeks 6](#) and [11](#).

4.3.2 Geometric programming: convexity by a change of variables

Design problems in engineering and economics multiply quantities (areas, volumes, power laws), and products and powers of the variables fit none of the families met so far.

Definition 4.20 (Posynomial, geometric programme). A **monomial** is a function $c x_1^{a_1} \dots x_n^{a_n}$ of $x > 0$ with $c > 0$ and real exponents. A **posynomial** is a sum of monomials, $f(x) = \sum_{k=1}^K c_k x_1^{a_{1k}} \dots x_n^{a_{nk}}$ with $c_k > 0$, defined for $x > 0$. A **geometric programme (GP)** minimises a posynomial subject to posynomial constraints of the form $f_i(x) \leq 1$.

Here n is the number of design variables, c_k are coefficients and $a_k = (a_{1k}, \dots, a_{nk})$ collects the exponents of term k ; none of these is the c or a_i of an LP. In the variables x a posynomial is typically *not* convex. Even $f(x) = x_1/x_2$ fails the Hessian test ([Theorem 3.12](#)): its Hessian $\begin{pmatrix} 0 & -1/x_2^2 \\ -1/x_2^2 & 2x_1/x_2^3 \end{pmatrix}$ has determinant $-1/x_2^4 < 0$ and so is not positive semidefinite. Now substitute $x_i = e^{y_i}$, which is legitimate because $x > 0$, and take logarithms. Term k becomes $c_k e^{a_{1k}y_1} \dots e^{a_{nk}y_n} = \exp(a_k^\top y + \log c_k)$, so

$$\log f(e^{y_1}, \dots, e^{y_n}) = \log \sum_{k=1}^K \exp(a_k^\top y + \log c_k), \quad (4.13)$$

a log-sum-exp of affine functions of y , which is convex ([Example 3.22](#)). Because the logarithm is increasing, minimising f is the same as minimising $\log f$, and $f_i(x) \leq 1$ is the same as $\log f_i(e^y) \leq 0$. In the variables $y = \log x$ the GP is thus a convex problem in standard form ([Definition 4.3](#)) of the exponential-cone family. The problem did not change; the decision variables did, and that made it convex.

Example 4.21 (Box design). Choose the dimensions $x_1, x_2, x_3 > 0$ of a box to maximise its volume $x_1 x_2 x_3$ under a surface-area budget $2(x_1 x_2 + x_2 x_3 + x_1 x_3) \leq A$ (here A is a number, not a matrix). Maximising the volume is minimising the monomial $x_1^{-1} x_2^{-1} x_3^{-1}$, and the constraint is the posynomial inequality $(2/A)(x_1 x_2 + x_2 x_3 + x_1 x_3) \leq 1$, so this is a GP, nonconvex as written. With $x_i = e^{y_i}$ the problem becomes

$$\begin{aligned} \max_{y \in \mathbb{R}^3} \quad & y_1 + y_2 + y_3 \\ \text{s.t.} \quad & \log(e^{y_1+y_2} + e^{y_2+y_3} + e^{y_1+y_3}) \leq \log(A/2), \end{aligned}$$

a linear objective with one convex log-sum-exp constraint. Convexity even lets us solve it by hand: the problem is symmetric in the coordinates of y , so averaging an optimal y^* with its permutations gives an optimal point with equal coordinates $y_i = \eta$ (feasible because the feasible set is convex, equally good because the objective is linear). The constraint $3e^{2\eta} \leq A/2$ is tight at the optimum, so the best box is the cube $x_i^* = e^\eta = \sqrt[3]{A/6}$, with volume $(A/6)^{3/2}$; for $A = 6$, MOSEK applied to the convex form returns $x^* = (1, 1, 1)$.

4.3.3 Convexity belongs to the formulation: pricing under logit demand

Convexity is a property of the *formulation*, not of the business problem: the same problem can be convex in one set of decision variables and nonconvex in another. Choosing the decision variables is itself a modelling decision, and often the one that decides convexity. Before declaring

a model nonconvex, ask whether a change of variables or a reformulation reveals hidden convexity. Geometric programming is one case; pricing is a second.

A firm sells n products at prices p_i (in this example p denotes prices and s market shares, not the equality count of (4.1) or a supply). Under the **multinomial logit (MNL)** demand model the fraction of customers who choose product i , its market share, is

$$s_i = \frac{e^{\alpha_i - \beta p_i}}{1 + \sum_{j=1}^n e^{\alpha_j - \beta p_j}}, \quad s_0 = 1 - \sum_{i=1}^n s_i = \frac{1}{1 + \sum_{j=1}^n e^{\alpha_j - \beta p_j}} \quad (\text{no purchase}), \quad (4.14)$$

where α_i measures the attractiveness of product i and $\beta > 0$ the price sensitivity of customers. The firm wants to maximise the revenue per customer $R = \sum_i p_i s_i(p)$.

Example 4.22 (MNL pricing in market shares). *In prices the problem is not convex.* Take $n = 2$, $\alpha = (1, 2)$ and $\beta = 0.5$. Then $R(6, 6) = 2.009$ and $R(14, 14) = 0.128$, with average 1.068, but at the midpoint $R(10, 10) = 0.638$. A concave function lies above its chords, so R is not concave in p : direct price optimisation is a nonconvex problem.

Change of variables. Take the market shares $s = (s_1, \dots, s_n)$ as the decisions. Dividing the two formulas in (4.14) gives $s_i/s_0 = e^{\alpha_i - \beta p_i}$, which inverts to

$$p_i = \frac{\alpha_i - \log(s_i/s_0)}{\beta}, \quad i = 1, \dots, n. \quad (4.15)$$

Substituting into $R = \sum_i p_i s_i$ and using $\sum_i s_i = 1 - s_0$,

$$R(s) = \frac{1}{\beta} \left[\sum_{i=1}^n \alpha_i s_i - \sum_{i=1}^n s_i \log s_i + (1 - s_0) \log s_0 \right], \quad s_i > 0, \quad \sum_{i=1}^n s_i < 1. \quad (4.16)$$

Concavity. Every piece is familiar from [Week 3](#). The first sum is affine. Each $-s_i \log s_i$ is concave because $x \log x$ is convex on $\mathbb{R}_{++}$ ([Example 3.6](#)). The function $\phi(t) = (1 - t) \log t$ has $\phi''(t) = -1/t - 1/t^2 < 0$ for $t > 0$, so it is concave, and $s_0 = 1 - \sum_i s_i$ is affine in s , so $\phi(s_0)$ is concave in s . A sum of concave functions multiplied by $1/\beta > 0$ is concave. Maximising $R(s)$ over the convex domain in (4.16) is therefore a convex problem, of the exponential-cone family because of its logarithms, and the optimal prices follow from s^* through (4.15).

Numbers. For the instance above, maximising $R(s)$ gives $s^* = (0.145, 0.393)$ and $s_0^* = 0.462$; (4.15) then returns the prices $p_1^* = p_2^* = 4.325$, with revenue $R^* = 2.325$ per customer. The prices coincide because $\partial R / \partial s_i = p_i - 1/(\beta s_0)$: at the optimum every product carries the price $1/(\beta s_0^*)$, and a more attractive product earns a larger share, not a higher price.

Note on the lecture slides

Frame 4.31 does not state the sign of β . The argument needs $\beta > 0$: the factor $1/\beta$ multiplies a concave bracket, so for $\beta < 0$ the revenue $R(s)$ would be convex. $\beta > 0$ is also what makes a product's share fall when its price rises.

4.4 Conic representation and modelling hygiene

4.4.1 Three cones and the representability ladder

[Week 2](#) introduced cones geometrically ([Definition 2.6](#)); here they return as the primitives that solvers understand, and a light view of three of them suffices. The *nonnegative orthant* $\{x : x \geq 0\}$ expresses sign constraints and, through slack variables, every linear inequality; it is all an LP needs. The *second-order cone* $\{(u, t) : \|u\|_2 \leq t\}$, the norm cone of [Definition 2.15](#), carries ℓ_2 norms, sums of squares and convex quadratics. The third cone is new.

Definition 4.23 (Exponential cone). The **exponential cone** is the set $\{(v_1, v_2, v_3) : v_1 \geq v_2 e^{v_3/v_2}, v_2 > 0\}$ together with its limit points $\{(v_1, 0, v_3) : v_1 \geq 0, v_3 \leq 0\}$. In particular, $e^u \leq v$ holds exactly when $(v, 1, u)$ belongs to it.

The exponential cone carries exponentials, logarithms, $x \log x$ terms, softplus and log-sum-exp. The softplus constraint of [Proposition 4.19](#), for instance, reaches the solver as $e^{u-t} \leq z_1, e^{-t} \leq z_2, z_1 + z_2 \leq 1$: two exponential-cone constraints and a linear one. A modelling layer such as RSOME converts every model of this week into one canonical shape — an affine objective, linear constraints, and affine expressions required to lie in copies of these cones — and routes it to a solver that supports the cones present.

This gives a *representability ladder*, which refines the families of [Section 4.2](#) into four rungs: LP, then QP and least squares, then norm and SOCP models, then exponential-cone models, each rung admitting the constraint types of those below it and one more. Mathematical convexity alone does not tell us which solver back end to use; the rung does ([Table 4.2](#)). Geometric programming is not a further rung: it is a modelling class that lands on the exponential-cone rung after the substitution $y = \log x$.

Note on the lecture slides

Frames 4.12 and 4.47 count four families, geometric programmes being the fourth, and say that the MOSEK Modeling Cookbook uses “exactly this order”. Frames 4.29 and 4.33, like the Cookbook itself, place GP inside the exponential-cone class. The notes therefore count three solver families, GP being a modelling class that reduces to the third.

4.4.2 Slack variables, split variables and domain restrictions

Three small moves account for a large fraction of modelling errors when they are forgotten. A **slack variable** turns an inequality into an equality plus a sign constraint, $a^\top x \leq b \iff a^\top x + s = b, s \geq 0$; it is useful when the gap s (unused budget, spare capacity) is itself reported, penalised or bounded. A **split variable** represents a free scalar as $u = u^+ - u^-$ with $u^+, u^- \geq 0$ when the two signs must be treated separately, as when buying and selling cost differently; if both parts carry a positive cost, at most one is nonzero at an optimum and $u^+ + u^- = |u|$. A domain restriction ([Definition 1.4](#)) such as $x > 0$ for $\log x$ or $x \log x$ belongs in the model, as in ([4.16](#)).

Remark 4.24 (Perspective and linear-fractional models). The perspective $t f(x/t), t > 0$, of a convex f is convex ([Proposition 3.19](#)), which underlies rescaling substitutions. A ratio of affine functions is in general not convex, yet on the region $c^\top x + d > 0$ the substitution $t = 1/(c^\top x + d)$ (here t is a scaling variable, not an epigraph variable) and $y = xt$ turns a linear-fractional problem into an LP:

$$\min_x \frac{a^\top x + b}{c^\top x + d} \text{ s.t. } Gx \leq h \quad \rightsquigarrow \quad \min_{y,t} a^\top y + bt \text{ s.t. } c^\top y + dt = 1, Gy \leq ht, t \geq 0.$$

Each constraint is multiplied by t , and $x = y/t$ recovers a solution when $t^* > 0$. This is the perspective at work. For $t > 0$ the new objective is $a^\top y + bt = t(a^\top(y/t) + b)$ and the new constraints are $t(G(y/t) - h) \leq 0$: perspectives ([Proposition 3.19\(c\)](#)) of the affine numerator and of the original constraint functions. That is why the rescaled model stays convex, and here even linear.

4.4.3 Common modelling traps

Five mistakes break convexity or correctness often enough to deserve a list. *Maximising a convex function*: convexity makes *minimisation* reliable (equivalently, maximisation of a concave function); the sign convention matters. *A convex function in an equality constraint*: as the circle

Table 4.1: Convex building blocks in RSOME (import rsome as rso, import numpy as np).

Building block	RSOME form	Typical use
Euclidean norm	<code>rso.norm(u)</code>	$\ u\ _2 \leq t$
ℓ_1 norm	<code>rso.norm(u, 1)</code>	$\ u\ _1 \leq t$
ℓ_∞ norm	<code>rso.norm(u, np.inf)</code>	$\ u\ _\infty \leq t$
sum of squares	<code>rso.sumsqr(u)</code>	least-squares epigraphs $\ u\ _2^2 \leq t$
quadratic form	<code>rso.quad(x, Q)</code>	convex quadratic constraints, $Q \succeq 0$
softplus	<code>rso.softplus(u)</code>	logistic-loss epigraphs
exp / log	<code>rso.exp(u)</code> , <code>rso.log(u)</code>	exponential-cone and domain-sensitive models

Table 4.2: Model families of this week and their solver lanes.

Family	Examples in this week	Gurobi	MOSEK
LP / affine models	media mix, transportation, Chebyshev centre, ℓ_1/ℓ_∞	✓	✓
Quadratic / least squares	constrained least squares, soft-margin SVM	✓	✓
Norm / SOCP models	portfolio risk budget, ℓ_2 norm bounds	✓	✓
Softplus / exponential cone	logistic regression, log-sum-exp, GP, MNL pricing	–	✓

after [Proposition 4.4](#) showed, a nonlinear equality usually destroys convexity. *A forgotten domain* such as $x > 0$ for $\log x$ leaves the model mathematically incomplete, and possibly meaningless to the solver. *Nonlinear pieces buried in the objective*: RSOME accepts convex objectives, but an affine objective, via an epigraph variable, keeps the conic structure explicit. *Treating “infeasible” or “unbounded” as a solver bug*: these statuses usually reveal a modelling mistake or a missing business rule ([Section 4.5.4](#)).

4.5 The RSOME workflow

4.5.1 Workflow, variable arrays and convex atoms

[Section 1.6](#) built a first model in RSOME. Every model follows the same six steps: create the model, define variables, build expressions, add constraints, solve, read the results. The `ro` environment suffices for all the deterministic convex models of this unit. Variables behave like NumPy arrays: `x = model.dvar(d)` creates a vector $x \in \mathbb{R}^d$ and `model.dvar((m, d))` a matrix, with slicing, broadcasting and matrix multiplication. With data arrays `A` and `b`, `r = A @ x - b` is a vector of N affine expressions, and `model.st(r <= 1)` or `model.st(x >= 0)` adds a whole block of constraints at once. Nonlinear convex pieces use the functions of [Table 4.1](#), placed on the smaller side of a $\leq$ constraint (the concave `rso.log` on the larger side). As throughout, the objective stays affine by convention, although RSOME also accepts convex objectives directly.

Note on the lecture slides

Frame 4.39 lists the single call `rso.norm(u)` for both $\|u\|_2 \leq t$ and $\|u\|_1 \leq t$. That call is the *Euclidean* norm; the ℓ_1 and ℓ_∞ norms need the second argument of [Table 4.1](#).

4.5.2 Choosing a solver back end

[Table 4.2](#) summarises the lanes. Once the constraints have been added, `model.do_math()` returns the canonical form that RSOME will send to the solver; printed, it shows the numbers of second-order-cone (SOC) and exponential-cone (ExpCone) constraints, the quickest check that the model belongs to the family you think it does. One pitfall deserves emphasis: RSOME’s Gurobi interface *drops* exponential-cone constraints, with no more than the warning `The SOCP solver ignores exponential cone constraints`, and the model Gurobi then solves is not your model.

Listing 4.1: An LP in RSOME: solve, extract and check.

```

import numpy as np
from rsome import ro
from rsome import grb_solver as grb
c = np.array([2.0, 3.0, 1.0])          # unit costs
A, b = np.eye(3), np.array([0.40, 0.35, 0.55]) # per-channel caps
C, e = np.ones((1, 3)), np.array([1.0])      # shares sum to one
model = ro.Model()
x = model.dvar(3)
model.min(c @ x)
model.st(A @ x <= b)
model.st(C @ x == e)
model.st(x >= 0)
model.solve(grb)
x_star, obj = x.get(), model.get()          # extract results
print('objective:', obj, ' x:', x_star)
print('feasible:', (A @ x_star <= b + 1e-6).all(),
      '| obj match:', abs(c @ x_star - obj) <= 1e-6)

```

For the logistic model below the stripped model is unbounded (nothing holds the ceilings t_i up): Gurobi stops with status 4 (“infeasible or unbounded”) and reading the results raises the error `No solution available`. In a less fortunate model the answer could be silently wrong. Solver switching is easy only once the model has been written in a disciplined, family-aware way.

4.5.3 Three coded models

The first model is an LP small enough to solve by hand. A budget is split into shares x_1, x_2, x_3 across three channels with unit costs $c = (2, 3, 1)$ and caps $(0.40, 0.35, 0.55)$: minimise $c^\top x$ subject to the caps, $\mathbf{1}^\top x = 1$ and $x \geq 0$. Filling the cheapest channel first (channel 3 to its cap, then channel 1, the remaining 0.05 to channel 2) gives $x^* = (0.40, 0.05, 0.55)$ with cost 1.5. [Listing 4.1](#) codes it in the template form of [Model 4.6](#) and then *checks* the answer in NumPy.

Running it prints `Solution status: 2`, the objective value 1.5, the plan `[0.4 0.05 0.55]` and the checks `feasible: True` and `obj match: True`, as in the hand solution; `model.do_math()` reports four inequality rows, one equality and no SOC or ExpCone constraints: an LP. (The fourth inequality is RSOME’s own epigraph constraint $c^\top x \leq t$ for the objective; the sign constraints become variable bounds.) The printed check tests only $Ax \leq b$ and the objective; a complete check also tests $Cx = e$ and $x \geq 0$ (both hold).

[Listing 4.2](#) codes two models with epigraph variables, given data arrays A, b , labels y and a weight lam . Part (a) is nonnegative least squares, an instance of [Model 4.13](#) written as $\min t$ subject to $\|Ax - b\|_2^2 \leq t, x \geq 0$; `do_math()` reports one SOC constraint and either solver may be used. Part (b) is the logistic model ([4.12](#)), line for line: `margin` holds $-b_i a_i^\top \theta$, the softplus constraint is vectorised over all observations, and the solver must be MOSEK. For $N = 40$, `do_math()` reports $80 = 2N$ ExpCone constraints and one SOC constraint (two per softplus, as predicted in [Section 4.4.1](#)). On random data both optimal values agree with independent SciPy computations (`nnls`, and a direct minimisation of [\(4.11\)](#)).

4.5.4 Reading output and diagnosing a failed solve

Read solver output in a fixed order. First the status: is it acceptable? (Gurobi prints the code 2 for “optimal”, MOSEK the word `Optimal`.) Never skip this line; after a failed solve the numbers are meaningless. Second the objective: do its direction, units and value make sense? Third the decisions: do they pass an independent feasibility check and the test of business logic? If not, revisit the formulation before blaming the solver.

Listing 4.2: Two epigraph models: (a) nonnegative least squares, (b) logistic regression.

```

import rsome as rso
from rsome import ro
from rsome import grb_solver as grb, msk_solver as msk
# (a) least squares with  $x \geq 0$ , epigraph form: Gurobi or MOSEK
model = ro.Model()
x = model.dvar(d)
t = model.dvar()
resid = A @ x - b
model.min(t)
model.st(rso.sumsqr(resid) <= t)
model.st(x >= 0)
model.solve(grb)
# (b) regularised logistic regression: MOSEK only
model = ro.Model()
theta = model.dvar(d)
t = model.dvar(N)
r = model.dvar()
margin = -(y * (A @ theta))      # y holds the labels  $b_i$  in  $\{-1, +1\}$ 
model.min(t.sum() + lam * r)
model.st(rso.softplus(margin) <= t)
model.st(rso.sumsqr(theta) <= r)
model.solve(msk)

```

A status other than “optimal” is a diagnostic signal about the model. *Infeasible* points to conflicting rules, impossible targets or a forgotten slack; ask which business rules cannot all hold together (recall [Model 4.9](#) and [Example 4.16](#)). *Unbounded* points to a missing budget, sign or domain restriction; ask what stops the objective from improving forever. The printed status rarely uses these words: through RSOME, Gurobi reports code 4 (“infeasible or unbounded”) in both cases and MOSEK a word such as *Undefined*, so the modeller must settle which diagnosis applies, as the minimal volatility did in [Example 4.16](#). A model may also be *poorly scaled*: huge coefficient ranges destabilise the numerics; rescale the data.

The workflow condenses into a reusable checklist: (1) state variables, data and units clearly; (2) write the objective and every business rule mathematically; (3) audit convexity and domain restrictions; (4) identify the solver-friendly representation and the solver lane; (5) after solving, interpret the status, the values and their managerial meaning.

4.6 Summary and looking ahead

Convex modelling means assembling variables, objective terms and constraints from safe building blocks. A model in the standard form of [Definition 4.3](#) has a convex feasible set, so every local minimum is global ([Theorem 4.1](#)). The epigraph reformulation of [Proposition 4.5](#) is the main bridge from mathematical convexity to a solver-ready form: it turns maxima, absolute values and ℓ_1/ℓ_∞ norms into linear programmes, hinge losses into a QP, and softplus losses into exponential-cone constraints. The linear, quadratic and second-order conic, and exponential conic classes cover a wide range of analytics applications; the LP examples of Luenberger and Ye cited in [Section 4.2.3](#) and the least-squares and maximum-volume-cuboid notebooks of the MOSEK Modeling Cookbook give further practice. Geometric programmes and logit pricing show that convexity may appear only after the right change of variables. In RSOME the represented family decides the solver: Gurobi or MOSEK for LP, QP and SOCP models, MOSEK for the exponential cone. Two limits remain: models that stay nonconvex under every reformulation ([Section 2.4](#)), and models too large for a general-purpose solver ([Week 6](#) onwards).

This week asked how to formulate a model; [Week 5](#) asks how to justify and interpret its solution. Starting from the standard form (4.1), it derives optimality conditions and duality,

which certify optimality and show how each constraint creates a trade-off.

4.7 Exercises

Exercise 4.1 (Classify the model families). Classify each model as LP, convex quadratic, norm/SOCP, exponential-cone or GP: media mix, transportation, constrained least squares, soft-margin SVM, portfolio with a risk budget, logistic regression, box-volume design.

Exercise 4.2 (Epigraph and norm reformulations). (a) Reformulate $\min_x \max_{i=1,\dots,m} (a_i^\top x + b_i)$ using an epigraph variable (Model 3.25), and write the result in the LP template (4.3): give c , A and b for the stacked variable $(x, t) \in \mathbb{R}^{d+1}$. (b) Reformulate $\min_x \|Ax - b\|_\infty$ using linear inequalities.

Exercise 4.3 (Domains are part of the model). Explain why forgetting the domain in a model that contains $\log(x)$ is a modelling error rather than a minor notational issue.

Exercise 4.4 (Two RSOME models). (a) Model $\min\{c^\top x : Ax \leq b, x \geq 0\}$ in RSOME and solve it with Gurobi. (b) Model $\min\{t : \|Ax - b\|_2^2 \leq t\}$ and inspect its standard form with `model.do_math()`.

Exercise 4.5 (Which solver lane?). In words, explain why a logistic-regression skeleton that uses `rso.softplus` belongs in the MOSEK lane.

Exercise 4.6 (A geometric programme). Show that the box-volume design problem (maximise $x_1 x_2 x_3$ subject to $2(x_1 x_2 + x_2 x_3 + x_1 x_3) \leq A, x > 0$) becomes convex after substituting $x_i = e^{y_i}$.

Exercise 4.7 (MNL pricing). For the MNL pricing model, verify that $\phi(t) = (1 - t) \log t$ is concave on $0 < t < 1$, and explain why this makes the revenue $R(s)$ of (4.16) concave in the market shares.

Week 5 Optimality Conditions, Duality and Shadow Prices

Week 4 ended with a convex model handed to a solver and a decision vector handed back. A manager who receives that vector is entitled to ask four further questions. *How do we know it is optimal*, that is, what evidence is there that no better feasible decision exists? *Which constraints matter*, in the sense that they are the business rules actually driving the trade-off? *What is one more unit worth*, if extra budget, time or capacity could be bought? And, when the solver returns a status instead of a solution, *what does that status mean*? This week turns solver output into answers to these questions, under the single heading: how do we *certify* that a solution is optimal?

The central idea is a certificate built from two bounds. For a minimisation problem, any feasible point gives an *upper* bound on the optimal value. We shall construct a second, “dual” problem whose feasible points give *lower* bounds. If a feasible decision and a dual-feasible point produce the same number, the bounds have met and nothing better can exist. For convex problems such matching pairs exist under mild conditions, which is why convex optimisation can certify its answers instead of merely proposing them.

Two earlier results carry most of the weight: the first-order inequality of [Theorem 3.8](#), which already showed that a stationary point of a convex function is a global minimiser ([Theorem 3.10](#)), and the standard convex form of [Definition 4.3](#), the starting point for everything that involves constraints. The development runs from geometry to formulas to managerial interpretation. [Section 5.1](#) treats problems without constraints, where the certificate is $\nabla f(x^*) = 0$. [Section 5.2](#) shows in pictures why constraints break that test and what replaces it: the objective gradient is balanced by the normals of the active constraints. [Section 5.3](#) turns the pictures into algebra through the Lagrangian and the dual problem, and [Section 5.4](#) derives the Karush–Kuhn–Tucker (KKT) conditions from matching primal and dual values. [Section 5.5](#) specialises to linear programmes, where the dual variables become resource prices, the *shadow prices* of the title. Four small examples recur: an unconstrained quadratic, the projection of a point onto a line, a projection onto a segment in the first quadrant, and a two-resource allocation LP. The long-only portfolio of [Week 1](#) returns through its KKT system, and an ℓ_1 -regularised least-squares condition previews later weeks.

5.1 Certifying an unconstrained minimum

5.1.1 From the derivative test to the gradient condition

In one variable the test is familiar from school calculus: an interior minimiser of a differentiable function satisfies $f'(x^*) = 0$. For $f(x) = (x - 2)^2 + 0.6$ we get $f'(x) = 2(x - 2)$, which vanishes at $x^* = 2$, where the tangent line is the horizontal line at height 0.6. For a general function this test is only a *necessary* condition, and only a local one: $f(x) = x^3$ has $f'(0) = 0$ although 0 is not a minimiser. What is special about a convex function is that its tangent line lies below the graph everywhere, so a horizontal tangent is a floor under the whole function. Convexity upgrades a local derivative test into a global optimality certificate.

The same reasoning works in $\mathbb{R}^d$. Recall from [Theorem 3.8](#) that a differentiable convex function satisfies the first-order inequality

$$f(y) \geq f(x) + \nabla f(x)^\top (y - x) \quad \text{for all } x, y \in \mathbb{R}^d, \quad (5.1)$$

so the affine approximation built at x underestimates f everywhere. Recall also from [Definition 1.15](#) that $\nabla f(x)$ is perpendicular to the contour of f through x and points towards higher values (left panel of [Figure 5.1](#)). If the gradient vanishes at x^* , the right-hand side of (5.1) is the constant $f(x^*)$, and the inequality says that no point does better.

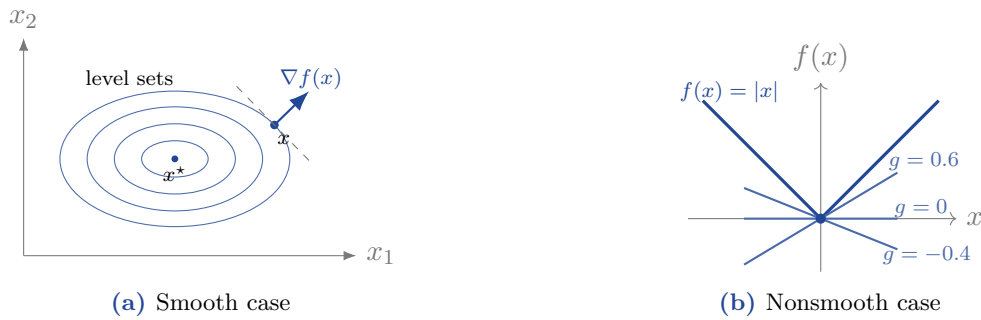


Figure 5.1: Two pictures of unconstrained optimality. (a) The gradient at x is perpendicular to the level set through x (dashed tangent) and points uphill; it vanishes only at the centre x^* . (b) At the kink of $|x|$ every line gx with $g \in [-1, 1]$ lies below the graph, so every such g is a subgradient at 0; because the horizontal line $g = 0$ is among them, $x^* = 0$ is the minimiser.

Theorem 5.1 (Unconstrained optimality, smooth case). *Let $f : \mathbb{R}^d \rightarrow \mathbb{R}$ be convex and differentiable. Then*

$$x^* \text{ is a global minimiser of } f \iff \nabla f(x^*) = 0.$$

If, in addition, f is strictly convex, then f has at most one minimiser.

Proof. ($\Leftarrow$) Put $x = x^*$ in (5.1): $f(y) \geq f(x^*) + 0^\top(y - x^*) = f(x^*)$ for every y . This is Theorem 3.10. ($\Rightarrow$) Suppose $\nabla f(x^*) \neq 0$ and move in the direction $v = -\nabla f(x^*)$. The directional derivative of Definition 1.15 is $\nabla f(x^*)^\top v = -\|\nabla f(x^*)\|_2^2 < 0$, so $f(x^* + tv) < f(x^*)$ for all small $t > 0$ and x^* is not a minimiser. (Uniqueness) This is Proposition 4.2 with $\mathcal{X} = \mathbb{R}^d$. $\square$

The theorem tells us what the certificate is (a zero gradient), why it works (the tangent plane is a global floor), and when it applies: f must be convex, differentiable, and minimised over all of $\mathbb{R}^d$. Its limitations are equally clear. It does not promise that a minimiser exists ($f(x) = e^x$ is convex and has none, as noted after Proposition 4.2); it says nothing at a kink; and it fails as soon as constraints are present (Section 5.2). Checking $\nabla f(x) = 0$ at a *proposed* point is a calculation, not a search; Week 6 builds its algorithms around this condition: they generate a sequence of points x^k , $k = 0, 1, 2, \dots$, and stop once $\|\nabla f(x^k)\|_2$ falls below a small tolerance $\varepsilon > 0$.

Key idea

For a smooth convex objective without constraints, stationarity is a certificate of global optimality: $\nabla f(x^*) = 0$ is both necessary and sufficient.

5.1.2 Nonsmooth objectives: a first look at the subgradient condition

Many useful convex functions, such as $|x|$, $\max\{0, x\}$ and the ℓ_1 norm, have kinks at exactly the points where their minimisers tend to sit, so a test that needs $\nabla f(x^*)$ is not enough. The remedy was introduced in Definition 3.17: a vector $g \in \mathbb{R}^d$ is a *subgradient* of a convex function f at x if it can play the role of the gradient in the first-order inequality,

$$f(y) \geq f(x) + g^\top(y - x) \quad \text{for all } y, \quad (5.2)$$

and the *subdifferential* $\partial f(x)$ is the set of all subgradients at x . Where f is differentiable, $\partial f(x) = \{\nabla f(x)\}$; at a kink it contains many vectors.

Theorem 5.2 (Unconstrained optimality, nonsmooth case). *Let $f : \mathbb{R}^d \rightarrow \mathbb{R}$ be convex. Then*

$$x^* \text{ is a global minimiser of } f \iff 0 \in \partial f(x^*).$$

Proof. By (5.2), $0 \in \partial f(x^*)$ means $f(y) \geq f(x^*) + 0^\top(y - x^*) = f(x^*)$ for all y , which is precisely the statement that x^* is a global minimiser. $\square$

The logic is identical to the smooth case: the affine lower bound becomes flat. When f is differentiable at x^* the condition reads $0 \in \{\nabla f(x^*)\}$ and we recover Theorem 5.1. The proof is one line because the theorem is close to a restatement of the definition; its practical value depends on being able to *compute* ∂f , and the rules for doing so are developed in Week 7. For the moment one subdifferential suffices. At $x = 0$, inequality (5.2) for $f(x) = |x|$ reads $|y| \geq gy$ for all y , which holds exactly when $|g| \leq 1$, so

$$\partial|x| = \begin{cases} \{\text{sign}(x)\}, & x \neq 0, \\ [-1, 1], & x = 0. \end{cases} \quad (5.3)$$

Since $0 \in [-1, 1]$, the theorem certifies that $x^* = 0$ minimises $|x|$ even though no derivative exists there (right panel of Figure 5.1).

5.1.3 Two worked examples: a convex quadratic and least squares

Example 5.3 (Unconstrained quadratic). Minimise $f(x_1, x_2) = 2x_1^2 + x_2^2 - 4x_1 - 2x_2 + 5$ over $\mathbb{R}^2$. *Step 1: convexity.* The Hessian is $\nabla^2 f(x) = \text{diag}(4, 2)$, which is positive definite, so f is strictly convex by Theorem 3.12. *Step 2: stationarity.*

$$\nabla f(x) = \begin{bmatrix} 4x_1 - 4 \\ 2x_2 - 2 \end{bmatrix} = 0 \implies x^* = (1, 1), \quad f(x^*) = 2 - 4 + 1 - 2 + 5 = 2.$$

Step 3: certificate. By Theorem 5.1, x^* is the global minimiser and it is unique. Completing the squares confirms this without calculus: $f(x) = 2(x_1 - 1)^2 + (x_2 - 1)^2 + 2 \geq 2$. The contours are nested ellipses centred at $(1, 1)$, as in the left panel of Figure 5.1.

Example 5.4 (Least squares and the normal equations). Let $A \in \mathbb{R}^{N \times d}$ and $b \in \mathbb{R}^N$ hold the data of a regression problem and consider $f(x) = \|Ax - b\|_2^2$, the least-squares objective of Model 1.3. It is convex for any data (Example 3.13), and the chain rule of Proposition 1.17 gives $\nabla f(x) = 2A^\top(Ax - b)$. By Theorem 5.1, x^* is a least-squares solution if and only if it satisfies the normal equations of Proposition 1.13,

$$A^\top A x^* = A^\top b. \quad (5.4)$$

If the columns of A are linearly independent, then $A^\top A$ is positive definite, f is strictly convex, and by the same proposition the unique minimiser is $x^* = (A^\top A)^{-1} A^\top b$. If the columns are dependent, the normal equations have infinitely many solutions, all with the same objective value. Later weeks scale the objective as $\frac{1}{2} \|Ax - b\|_2^2$, whose gradient is $A^\top(Ax - b)$; the factor changes neither the minimiser nor the normal equations.

Least squares is the rare model whose optimality condition can be solved directly; for most objectives $\nabla f(x) = 0$ is a target to be approached iteratively, which is where Week 6 begins.

5.2 Constrained optimality: active constraints and balancing normals

5.2.1 Why constraints change the picture

With constraints the objective gradient need not vanish at the optimum. The smallest example is $\min x^2$ subject to $x \geq 1$: the solution is $x^* = 1$, where $f'(1) = 2 \neq 0$. The objective would like

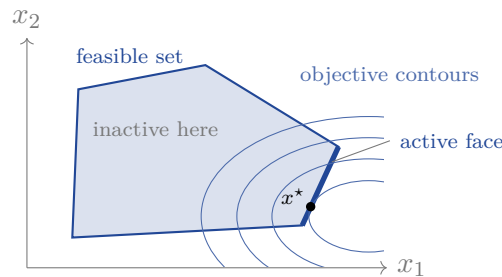


Figure 5.2: A constrained optimum. The contours are centred outside the feasible polygon; the lowest contour that touches it is tangent to one edge, the active face, at x^* . There the gradient is not zero but perpendicular to the active edge; the other edges are inactive.

to move left, but the boundary blocks the only improving direction. Figure 5.2 shows the same effect in two dimensions. The unconstrained minimiser (the centre of the elliptical contours) lies outside the feasible polygon, and the best feasible point x^* is where the lowest contour that still touches the polygon meets it. At x^* the gradient is not zero; instead, every direction that would lower the objective leads out of the feasible set. The question is therefore no longer “is the gradient zero?” but “which constraints are holding the solution in place, and how do they balance the gradient?”

Definition 5.5 (Active and inactive constraints). Let x^* be feasible for a problem with inequality constraints $f_i(x) \leq 0$. The i th inequality is **active** (or binding) at x^* if $f_i(x^*) = 0$ and **inactive** (or slack) if $f_i(x^*) < 0$. An equality constraint is active at every feasible point.

An inactive constraint is locally irrelevant: it remains satisfied for every sufficiently small move away from x^* , so it cannot enter the first-order balance. The active constraints describe the face of the feasible set on which the solution lies; in Figure 5.2 one edge of the polygon is active and the other four are inactive.

5.2.2 One equality constraint: tangency and the multiplier

Consider first $\min f(x)$, with f differentiable, subject to a single linear equality $a^\top x = b$ with $a \neq 0$; by Definition 2.10 the feasible set is a hyperplane with normal vector a . From a feasible point we may move only in directions v that keep $a^\top(x + tv) = b$, that is, in directions with $a^\top v = 0$, tangent to the hyperplane. At an optimum x^* no such direction may lower f to first order, so $\nabla f(x^*)^\top v \geq 0$; since $-v$ is feasible whenever v is, in fact $\nabla f(x^*)^\top v = 0$ for every v with $a^\top v = 0$. A vector orthogonal to everything that is orthogonal to a must be a multiple of a . (Write $\nabla f(x^*) = \alpha a + r$ with $a^\top r = 0$; then r is itself a feasible direction and $0 = \nabla f(x^*)^\top r = \|r\|_2^2$.) Hence there is a number ν , the *multiplier* of the constraint, with

$$\nabla f(x^*) + \nu a = 0. \quad (5.5)$$

Geometrically, the contour of f through x^* is tangent to the constraint: if it crossed the hyperplane, sliding along the hyperplane would reach a lower contour. The multiplier ν may have either sign, because an equality can be left on either side.

Example 5.6 (Projection onto a line). Find the point of the line $x_1 + x_2 = 2$ closest to $c = (3, 1)$:

$$\begin{aligned} \min \quad & (x_1 - 3)^2 + (x_2 - 1)^2 \\ \text{s.t.} \quad & x_1 + x_2 = 2. \end{aligned}$$

By substitution. Put $x_2 = 2 - x_1$. The objective becomes $(x_1 - 3)^2 + (1 - x_1)^2 = 2x_1^2 - 8x_1 + 10 = 2(x_1 - 2)^2 + 2$, a convex function of one variable minimised at $x_1 = 2$. Hence $x^* = (2, 0)$ with

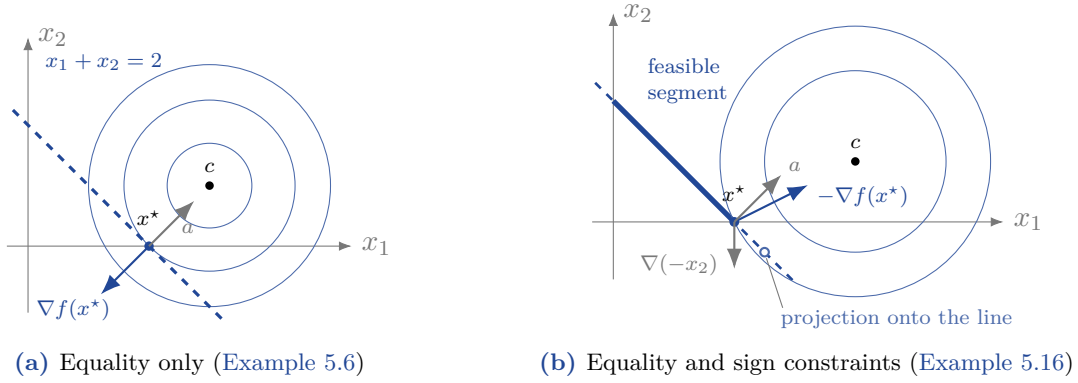


Figure 5.3: The geometry of constrained optimality. (a) The contour through $x^* = (2, 0)$ is tangent to the feasible line, so $\nabla f(x^*)$ is parallel to the normal $a = (1, 1)$. (b) The projection of c onto the whole line (hollow dot) is infeasible, and the optimum moves to the endpoint $(2, 0)$. There $-\nabla f(x^*)$ is a combination of the equality normal $a = (1, 1)$ and the normal $(0, -1)$ of the active inequality $-x_2 \leq 0$, the latter with a nonnegative weight.

optimal value 2. *By a multiplier.* Here $a = (1, 1)$, and (5.5) together with feasibility gives three equations in (x_1, x_2, ν) :

$$2(x_1 - 3) + \nu = 0, \quad 2(x_2 - 1) + \nu = 0, \quad x_1 + x_2 = 2.$$

The first two give $x_1 = 3 - \nu/2$ and $x_2 = 1 - \nu/2$; substituting into the third, $4 - \nu = 2$, so $\nu^* = 2$ and again $x^* = (2, 0)$. As a check, $\nabla f(x^*) = (-2, -2) = -2(1, 1)$. *Geometry.* The contour through x^* is the circle of radius $\sqrt{2}$ about c , tangent to the line at x^* ; the gradient is normal to the line and is cancelled by ν^*a (left panel of Figure 5.3). Substitution needs a constraint that can be solved for one variable; the multiplier method does not, and it extends to inequalities.

5.2.3 Equality plus inequalities: an endpoint solution

Now move the target point to $c = (4, 1)$ and add sign constraints, written in the standard form $f_i(x) \leq 0$:

$$\begin{aligned} \min \quad & (x_1 - 4)^2 + (x_2 - 1)^2 \\ \text{s.t.} \quad & x_1 + x_2 = 2, \\ & -x_1 \leq 0, \quad -x_2 \leq 0. \end{aligned} \tag{5.6}$$

The equality gives a line and the sign constraints cut it down to the segment from $(0, 2)$ to $(2, 0)$. Ignoring the inequalities and repeating the multiplier calculation of Example 5.6 gives the projection of c onto the whole line, $(2.5, -0.5)$, which violates $-x_2 \leq 0$. On the segment itself, $x_2 = 2 - x_1$ turns the objective into $2x_1^2 - 10x_1 + 17$, whose derivative $4x_1 - 10$ is negative on $[0, 2]$; the objective decreases all the way to the endpoint, so $x^* = (2, 0)$ with optimal value 5. At this point the equality and the inequality $-x_2 \leq 0$ are active, while $-x_1 \leq 0$ is inactive.

The balance condition now involves both active normals. With $\nabla f(x^*) = (-4, -2)$,

$$\nabla f(x^*) + \nu \begin{bmatrix} 1 \\ 1 \end{bmatrix} + \lambda \begin{bmatrix} 0 \\ -1 \end{bmatrix} = 0, \quad \lambda \geq 0, \tag{5.7}$$

that is, $-\nabla f(x^*) = (4, 2) = 4(1, 1) + 2(0, -1)$: the negative gradient is assembled from the normals of the active constraints (right panel of Figure 5.3). The new feature is the sign condition on λ , the multiplier of the inequality (in Weeks 3 and 4 λ was a regularisation weight; in this week it is a multiplier, except in the ℓ_1 preview of Section 5.4): an inequality $f_i(x) \leq 0$ can be left only towards $f_i < 0$, so its normal may push in one direction only. To see why this forces $\lambda \geq 0$, take $v = (-1, 1)$, the direction from $x^* = (2, 0)$ into the segment. It satisfies $a^\top v = 0$, and

its inner product with the normal $(0, -1)$ of the active inequality is $-1 < 0$. The inner product of (5.7) with v therefore gives $\nabla f(x^*)^\top v = \lambda$. If λ were negative, stepping back into the feasible set would *lower* the objective, contradicting optimality; here $\nabla f(x^*)^\top v = 4 - 2 = 2$. The multipliers are computed systematically in Example 5.16, once the general conditions are available.

5.3 Lagrangian duality

The pictures of Section 5.2 attach a multiplier to each active constraint. A general method needs a systematic way of producing multipliers and, above all, of *proving* that a proposed point is optimal. Both come from a single construction.

5.3.1 The Lagrangian and its multipliers

We work with the standard convex problem of Definition 4.3, with its affine equality constraints stacked into one matrix equation:

$$\begin{aligned} \min \quad & f_0(x) \\ \text{s.t.} \quad & f_i(x) \leq 0, \quad i = 1, \dots, m, \\ & Ax = b, \end{aligned} \tag{5.8}$$

where $f_0, f_1, \dots, f_m$ are convex. From here on the objective is written f_0 , and A, b denote the equality-constraint data, not the regression data of Example 5.4. Problem (5.8) is called the **primal problem** and p^* denotes its optimal value (written z^* in Definition 1.5).

Definition 5.7 (Lagrangian). The **Lagrangian** of problem (5.8) is the function

$$L(x, \lambda, \nu) = f_0(x) + \sum_{i=1}^m \lambda_i f_i(x) + \nu^\top (Ax - b), \quad \lambda \geq 0.$$

The numbers $\lambda_1, \dots, \lambda_m$ and the entries of ν are the **multipliers**, one per constraint. Multipliers of inequality constraints must be nonnegative; multipliers of equality constraints are unrestricted in sign.

The Lagrangian removes the constraints and charges for them in the objective instead, and the charge can be read in two ways. In the *penalty* reading, a violated constraint ($f_i(x) > 0$) adds $\lambda_i f_i(x)$ to the cost, a positive charge as soon as $\lambda_i > 0$, and a larger multiplier punishes the violation more strictly. In the *price* reading, λ_i is the price of a scarce resource: a decision that overuses the resource pays for the excess, and one that leaves slack is credited for it. This second reading leads to the shadow prices of Section 5.5: the same multiplier has a mathematical role in optimality and a managerial role in interpretation. The sign rule for the multipliers is the one found in Section 5.2. One consequence is immediate. If x is feasible and $\lambda \geq 0$, every term $\lambda_i f_i(x)$ is at most zero and $\nu^\top (Ax - b) = 0$, so

$$L(x, \lambda, \nu) \leq f_0(x) \quad \text{for every feasible } x. \tag{5.9}$$

5.3.2 The dual function and weak duality

For fixed multipliers the Lagrangian is an unconstrained function of x , and minimising it is a problem of the kind treated in Section 5.1.

Definition 5.8 (Dual function and dual problem). The **dual function** of (5.8) is

$$g(\lambda, \nu) = \inf_x L(x, \lambda, \nu),$$

the infimum (Definition 1.10) being taken over all x , feasible or not; the value $-\infty$ is allowed. A pair (λ, ν) is **dual feasible** if $\lambda \geq 0$ and $g(\lambda, \nu) > -\infty$. The **dual problem** is

$$\begin{aligned} \max \quad & g(\lambda, \nu) \\ \text{s.t.} \quad & \lambda \geq 0, \end{aligned} \tag{5.10}$$

and d^* denotes its optimal value.

From here on the symbol g denotes the dual function, no longer a subgradient as in Section 5.1 (the two never appear together); Weeks 12 and 13 write it as $q(\lambda)$. Multipliers with $g(\lambda, \nu) = -\infty$ carry no information, which is why dual feasibility excludes them; in a linear programme this implicit requirement is exactly what produces the constraints of the dual (Section 5.5).

Note on the lecture slides

The slides speak of dual-feasible and dual-infeasible points (frames 5.3, 5.21, 5.23, 5.40) but display only $\lambda \geq 0$ as the dual constraint (frames 5.22, 5.28). Read literally, $\lambda = 0$ would always be dual feasible and no dual could be infeasible. The notes therefore require $g(\lambda, \nu) > -\infty$ as well, the reading under which the status rules of Proposition 5.23 make sense.

Theorem 5.9 (Weak duality). *For every $\lambda \geq 0$ and every ν , $g(\lambda, \nu) \leq p^*$. Consequently $d^* \leq p^*$.*

Proof. Let x be any primal-feasible point. Then

$$g(\lambda, \nu) \leq L(x, \lambda, \nu) \leq f_0(x). \tag{5.11}$$

The first inequality holds because an infimum over all x is no larger than the value at one particular x ; the second is (5.9). Since the left-hand side does not depend on x , taking the infimum of $f_0(x)$ over feasible x gives $g(\lambda, \nu) \leq p^*$, and maximising over (λ, ν) gives $d^* \leq p^*$. $\square$

The proof uses neither convexity nor differentiability, so weak duality holds for every optimisation problem. The Lagrangian converts a constrained problem into a family of unconstrained lower bounds, one for each choice of multipliers, and better multipliers give tighter bounds; the dual problem (5.10) asks for the best one. The primal searches over decisions x , the dual over prices, one per constraint, which is why a dual solution is often easier to interpret. The dual is also well behaved: g is a pointwise infimum of functions affine in (λ, ν) , hence concave by the mirror image of the maximum rule in Proposition 3.19. The difference $p^* - d^* \geq 0$ is called the **duality gap**.

Example 5.10 (The dual of the projection problem). For Example 5.6 there is one equality constraint and $L(x, \nu) = (x_1 - 3)^2 + (x_2 - 1)^2 + \nu(x_1 + x_2 - 2)$. For fixed ν this is a strictly convex quadratic in x , so by Theorem 5.1 its minimiser solves $2(x_1 - 3) + \nu = 0$ and $2(x_2 - 1) + \nu = 0$, that is, $x(\nu) = (3 - \nu/2, 1 - \nu/2)$. Substituting back,

$$g(\nu) = \frac{\nu^2}{4} + \frac{\nu^2}{4} + \nu(2 - \nu) = 2\nu - \frac{1}{2}\nu^2.$$

Every value of g is a lower bound on $p^* = 2$: for instance $g(0) = 0$, $g(1) = 1.5$ and $g(3) = 1.5$. The best bound is found from $g'(\nu) = 2 - \nu = 0$, giving $\nu^* = 2$ and $d^* = g(2) = 2$. The dual function is a downward parabola that touches the level p^* at its top: the dual optimum reproduces the multiplier found geometrically, and the best lower bound equals p^* .

5.3.3 The dual problem as a source of certificates

For a minimisation problem we now have bounds on both sides of the unknown p^* : every dual-feasible pair gives a number below it and every primal-feasible point a number above it, $g(\lambda, \nu) \leq d^* \leq p^* \leq f_0(x)$. If the two outer numbers coincide there is no gap left to close.

Corollary 5.11 (Certificate from matching bounds). *If x is primal feasible, (λ, ν) is dual feasible and $f_0(x) = g(\lambda, \nu)$, then x is primal optimal, (λ, ν) is dual optimal, and $p^* = d^*$.*

In [Example 5.10](#) the feasible point $x = (2, 0)$ has value 2 and the multiplier $\nu = 2$ has dual value 2; this pair of numbers alone proves that $(2, 0)$ is optimal, and anyone can check it without repeating the search that produced it.

Key idea

A primal-feasible point and a dual-feasible point with the same objective value certify each other. Weak duality makes the certificate valid; the remaining question is when such a matching pair exists.

5.4 The Karush–Kuhn–Tucker conditions

5.4.1 Strong duality and Slater's condition

Weak duality guarantees $d^* \leq p^*$ but leaves open whether the best lower bound reaches p^* .

Definition 5.12 (Strong duality and Slater's condition). **Strong duality** holds for problem (5.8) if $d^* = p^*$, that is, if the duality gap is zero. A point x^{sl} is **strictly feasible** if $f_i(x^{\text{sl}}) < 0$ for all $i = 1, \dots, m$ and $Ax^{\text{sl}} = b$; **Slater's condition** requires that such a point exists.

For nonconvex problems the gap is typically positive. For convex problems Slater's condition closes it, a result we use as a practical checklist and do not prove.

Theorem 5.13 (Strong duality under Slater's condition). *Suppose problem (5.8) is convex and has a strictly feasible point. Then $p^* = d^*$, and if this value is finite there is a dual-feasible pair (λ^*, ν^*) with $g(\lambda^*, \nu^*) = p^*$. For a linear programme no strictly feasible point is required: strong duality holds whenever the programme is feasible with a finite optimal value.*

The Slater point need not be a good decision; it only shows that the inequality constraints leave some room. Checking the condition means exhibiting one point: $x = (1, 1)$ for the segment problem (5.6), since $-1 < 0$ twice and $1 + 1 = 2$.

5.4.2 From strong duality to the optimality conditions

Assume that the problem is convex, that Slater's condition holds, and that x^* is primal optimal and (λ^*, ν^*) dual optimal. Writing out what each of these words means gives the chain

$$f_0(x^*) = g(\lambda^*, \nu^*) = \inf_x L(x, \lambda^*, \nu^*) \leq L(x^*, \lambda^*, \nu^*) \leq f_0(x^*). \quad (5.12)$$

The first equality is strong duality, the second is the definition of g , and the two inequalities are those of (5.11) evaluated at x^* . The two ends of the chain are the same number, so both inequalities must hold with equality, and each equality delivers one optimality condition.

Equality in the second inequality says $L(x^*, \lambda^*, \nu^*) = f_0(x^*)$, that is, $\sum_i \lambda_i^* f_i(x^*) + \nu^{*\top}(Ax^* - b) = 0$. Feasibility gives $Ax^* - b = 0$, hence $\sum_i \lambda_i^* f_i(x^*) = 0$. Every term of this sum satisfies $\lambda_i^* f_i(x^*) \leq 0$, because $\lambda_i^* \geq 0$ and $f_i(x^*) \leq 0$, and a sum of nonpositive terms vanishes only if each term does:

$$\lambda_i^* f_i(x^*) = 0, \quad i = 1, \dots, m. \quad (5.13)$$

This is **complementary slackness**. It is no extra assumption: it falls out of matching bounds.

Equality in the first inequality says $L(x^*, \lambda^*, \nu^*) = \inf_x L(x, \lambda^*, \nu^*)$: the point x^* minimises the Lagrangian over all x , without constraints. The function $L(\cdot, \lambda^*, \nu^*)$ is convex, being a nonnegative combination of convex functions plus an affine term, so [Theorem 5.2](#) gives $0 \in \partial_x L(x^*, \lambda^*, \nu^*)$, where ∂_x is the subdifferential with respect to x . When $f_0, \dots, f_m$ are differentiable this is the gradient equation

$$\nabla f_0(x^*) + \sum_{i=1}^m \lambda_i^* \nabla f_i(x^*) + A^\top \nu^* = 0, \quad (5.14)$$

which is the balance of normals of [Section 5.2](#): for affine constraints ∇f_i is the constraint normal, and (5.5) and (5.7) are special cases.

5.4.3 The KKT conditions as a certificate

Definition 5.14 (KKT conditions). Let $f_0, \dots, f_m$ be differentiable. A triple (x^*, λ^*, ν^*) satisfies the **Karush–Kuhn–Tucker (KKT) conditions** for problem (5.8) if

- (i) *primal feasibility*: $f_i(x^*) \leq 0$ for $i = 1, \dots, m$, and $Ax^* = b$;
- (ii) *dual feasibility*: $\lambda_i^* \geq 0$ for $i = 1, \dots, m$;
- (iii) *complementary slackness*: $\lambda_i^* f_i(x^*) = 0$ for $i = 1, \dots, m$;
- (iv) *stationarity*: $\nabla f_0(x^*) + \sum_{i=1}^m \lambda_i^* \nabla f_i(x^*) + A^\top \nu^* = 0$.

Theorem 5.15 (KKT and optimality for convex problems). *Let problem (5.8) be convex with $f_0, \dots, f_m$ differentiable.*

- (a) *Necessity. If Slater's condition holds and x^* is primal optimal, then there exist multipliers (λ^*, ν^*) such that (x^*, λ^*, ν^*) satisfies the KKT conditions.*
- (b) *Sufficiency. If (x^*, λ^*, ν^*) satisfies the KKT conditions, then x^* is primal optimal, (λ^*, ν^*) is dual optimal, and the duality gap is zero. Slater's condition is not needed for this direction.*

Proof. (a) By [Theorem 5.13](#) a dual optimal pair with $g(\lambda^*, \nu^*) = p^* = f_0(x^*)$ exists, and the argument following (5.12) yields (iii) and (iv); (i) and (ii) hold by feasibility. (b) Because $\lambda^* \geq 0$, the function $L(\cdot, \lambda^*, \nu^*)$ is convex and differentiable, and by (iv) its gradient vanishes at x^* . By [Theorem 5.1](#), x^* minimises it, so $g(\lambda^*, \nu^*) = L(x^*, \lambda^*, \nu^*)$. By (iii) and $Ax^* = b$ the multiplier terms of $L(x^*, \lambda^*, \nu^*)$ vanish, hence $g(\lambda^*, \nu^*) = f_0(x^*)$. The point x^* is primal feasible by (i), and (λ^*, ν^*) is dual feasible in the full sense of [Definition 5.8](#), since $\lambda^* \geq 0$ by (ii) and $g(\lambda^*, \nu^*)$ is finite; so [Corollary 5.11](#) applies. $\square$

The two directions play different roles. Sufficiency is the certificate: whoever exhibits multipliers satisfying (i)–(iv) has proved global optimality by mere substitution. Necessity explains why it is reasonable to *look* for such multipliers: under Slater's condition every optimum has them. Without it they may not exist: in $\min x$ subject to $x^2 \leq 0$ the only feasible point is $x^* = 0$, where the stationarity equation $1 + 2\lambda x^* = 0$ fails for every λ . For nonconvex problems KKT points need not be optimal, so convexity is what makes the conditions a certificate.

Complementary slackness works as a switch. A slack inequality, $f_i(x^*) < 0$, must have $\lambda_i^* = 0$: a constraint that is not binding has no price and drops out of (5.14), in agreement with [Definition 5.5](#). A positive multiplier, $\lambda_i^* > 0$, forces $f_i(x^*) = 0$. The converse of the second rule is false: an active inequality may still have a zero multiplier. In $\min x^2$ subject to $-x \leq 0$ the solution $x^* = 0$ lies on the boundary, yet stationarity $2x^* - \lambda^* = 0$ gives $\lambda^* = 0$; the constraint is active but is not what holds the solution in place.

In hand calculations the switch suggests a recipe: guess which inequalities are active; set the multipliers of the others to zero; solve stationarity together with the active constraints as

equations for (x, λ, ν) ; then check the remaining inequalities and $\lambda \geq 0$. If the check fails, another active set is tried.

5.4.4 Worked examples: the segment problem and the long-only portfolio

Example 5.16 (KKT for the segment problem). Consider again problem (5.6), with $f_0(x) = (x_1 - 4)^2 + (x_2 - 1)^2$, $f_1(x) = -x_1$, $f_2(x) = -x_2$ and the equality $x_1 + x_2 = 2$. Stationarity reads

$$\begin{bmatrix} 2(x_1 - 4) \\ 2(x_2 - 1) \end{bmatrix} + \lambda_1 \begin{bmatrix} -1 \\ 0 \end{bmatrix} + \lambda_2 \begin{bmatrix} 0 \\ -1 \end{bmatrix} + \nu \begin{bmatrix} 1 \\ 1 \end{bmatrix} = 0.$$

Guess 1: no inequality active ($\lambda_1 = \lambda_2 = 0$). This is the calculation of [Example 5.6](#) with the new centre and gives $x = (2.5, -0.5)$, $\nu = 3$; primal feasibility fails because $x_2 < 0$. *Guess 2: $-x_1 \leq 0$ active* ($x_1 = 0$, $\lambda_2 = 0$). Then $x = (0, 2)$ and $\nabla f_0(x) = (-8, 2)$; the second row gives $\nu = -2$ and the first $\lambda_1 = -8 + \nu = -10 < 0$. Dual feasibility fails: the negative multiplier says that the objective improves on leaving this endpoint along the segment. *Guess 3: $-x_2 \leq 0$ active* ($x_2 = 0$, $\lambda_1 = 0$). Then $x^* = (2, 0)$ and $\nabla f_0(x^*) = (-4, -2)$; the first row gives $\nu^* = 4$ and the second $\lambda_2^* = -2 + \nu^* = 2 \geq 0$. With $\lambda^* = (0, 2)$ all four blocks hold: x^* is feasible, $\lambda^* \geq 0$, $\lambda_1^* f_1(x^*) = 0 \cdot (-2) = 0$ and $\lambda_2^* f_2(x^*) = 2 \cdot 0 = 0$, and stationarity holds by construction. By [Theorem 5.15\(b\)](#), $x^* = (2, 0)$ is the global minimiser, with $p^* = 5$. Indeed, minimising $L(x, \lambda^*, \nu^*)$ over $x \in \mathbb{R}^2$ gives the minimiser $(2, 0)$ and $g(\lambda^*, \nu^*) = 5$: the bounds match.

The second example revisits the long-only mean–variance portfolio of [Model 1.2](#), rewritten as a minimisation so that the sign conventions of [Definition 5.14](#) apply directly.

Model 5.17 (Long-only portfolio and its KKT system).

$$\begin{array}{ll} \max_w & \mu^\top w - \gamma w^\top \Sigma w \\ \text{s.t.} & \mathbf{1}^\top w = 1, \quad w \geq 0 \end{array} \quad \text{is equivalent to} \quad \begin{array}{ll} \min_w & \gamma w^\top \Sigma w - \mu^\top w \\ \text{s.t.} & \mathbf{1}^\top w = 1, \quad -w \leq 0. \end{array} \quad (5.15)$$

With a multiplier $\nu \in \mathbb{R}$ for the budget constraint and $\lambda \in \mathbb{R}^d$ for the d sign constraints, the KKT conditions of the minimisation form are

$$2\gamma \Sigma w^* - \mu + \nu^* \mathbf{1} - \lambda^* = 0, \quad \mathbf{1}^\top w^* = 1, \quad w^* \geq 0, \quad \lambda^* \geq 0, \quad \lambda_i^* w_i^* = 0 \quad (i = 1, \dots, d). \quad (5.16)$$

The decision variables are the portfolio weights $w \in \mathbb{R}^d$; the data are the mean returns μ , the positive semidefinite covariance matrix Σ and the risk-aversion weight $\gamma \geq 0$. The objective trades expected return against variance, the equality makes the weights sum to one, and $w \geq 0$ forbids short selling. The minimisation form is a convex quadratic programme ([Proposition 3.14](#)) with affine constraints, and $w = \mathbf{1}/d$ is strictly feasible, so by [Theorem 5.15](#) the system (5.16) characterises the optimal portfolios exactly; its first equation is the w -gradient of $\gamma w^\top \Sigma w - \mu^\top w + \nu(\mathbf{1}^\top w - 1) - \lambda^\top w$. In practice the model is handed to a solver as in [Section 4.5](#), and the KKT system is how the returned weights are read. Rearranged, row i of stationarity says $\mu_i - 2\gamma(\Sigma w^*)_i = \nu^* - \lambda_i^*$, where the left-hand side is the marginal benefit of asset i : its expected return less its marginal contribution to the risk penalty. If $w_i^* > 0$ then $\lambda_i^* = 0$, so every asset that is held has the same marginal benefit ν^* . If $w_i^* = 0$ then λ_i^* may be positive: the marginal benefit of the asset falls short of ν^* by λ_i^* , and the no-short-selling constraint is what keeps it out. This is the active-constraint logic of [Example 5.16](#) in a finance model. For instance, with $\mu = (0.12, 0.10, 0.05)$, $\Sigma = 0.05 I$ and $\gamma = 1$ the system is solved by $w^* = (0.6, 0.4, 0)$, $\nu^* = 0.06$, $\lambda^* = (0, 0, 0.01)$: both held assets have marginal benefit 0.06, and asset 3 offers only 0.05.

Note on the lecture slides

Frames 5.4 and 5.31 attribute (5.15) to Week 4 (“Week 4’s portfolio model”, “Week 4’s mean–variance model”). The model displayed, with γ a risk-aversion weight in the objective, is the one from [Week 1](#); [Week 4](#) used a different formulation ([Model 4.15](#)) in which γ is a hard cap on the standard deviation. The KKT system on the slide is correct for the model it displays.

5.4.5 What KKT delivers, and three optimality templates

In practice the KKT conditions give four things: a certificate that a reported point is truly optimal; an explanation of which constraints drive the trade-off, namely those with positive multipliers; the raw material for sensitivity analysis, since the multipliers are prices ([Section 5.5](#)); and a structured way to debug a model and to interpret a solver status. For the algorithm half of the course the week leaves three optimality templates:

$$\begin{aligned} \text{smooth, unconstrained} & : \nabla f(x^*) = 0, \\ \text{convex, nonsmooth, unconstrained} & : 0 \in \partial f(x^*), \\ \text{smooth, convex, constrained} & : \text{the KKT conditions.} \end{aligned} \quad (5.17)$$

[Week 6](#) starts from the first line with gradient methods, [Week 7](#) returns to the second with subgradient methods, and the projected and proximal methods of [Weeks 8](#) and [9](#) turn the conditions into practical stopping tests.

The second template can already be used on a model developed in [Week 9](#) ([Model 9.3](#)), with the subdifferential calculus it needs supplied by [Week 7](#): ℓ_1 -regularised least squares minimises $\frac{1}{2} \|A\beta - b\|_2^2 + \lambda \|\beta\|_1$ over $\beta \in \mathbb{R}^d$, with data $A \in \mathbb{R}^{N \times d}$, $b \in \mathbb{R}^N$. For this model, here and in the exercises, $\lambda > 0$ is a penalty weight chosen by the analyst, not a multiplier. The objective is convex, with kinks wherever a coefficient is zero, and the template reads

$$0 \in A^\top (A\beta^* - b) + \lambda \partial \|\beta^*\|_1. \quad (5.18)$$

The smooth part contributes its gradient and $\partial \|\beta\|_1$ collects the componentwise subdifferentials $\partial|\beta_j|$ of (5.3); that subdifferentials may be added and assembled in this way is taken on trust until [Week 7](#). Written out coordinate by coordinate,

$$(A^\top (A\beta^* - b))_j = -\lambda \text{sign}(\beta_j^*) \quad \text{if } \beta_j^* \neq 0, \quad |(A^\top (A\beta^* - b))_j| \leq \lambda \quad \text{if } \beta_j^* = 0.$$

A coefficient may therefore stay at exactly zero whenever the inner product of its feature (column j of A) with the residual $A\beta^* - b$ is at most λ in absolute value, which is why this penalty produces sparse models.

5.5 Linear programming duality, shadow prices and diagnostics

For linear programmes, the model class of [Model 4.6](#), the dual can be written down explicitly, strong duality holds without a Slater point, and the multipliers acquire a concrete economic meaning.

5.5.1 The primal–dual pair and its economic reading

Model 5.18 (Canonical primal–dual LP pair). For $A \in \mathbb{R}^{m \times d}$, $b \in \mathbb{R}^m$ and $c \in \mathbb{R}^d$, the primal and dual linear programmes are

$$\begin{aligned} \text{(primal)} \quad & \max \quad c^\top x \\ & \text{s.t.} \quad Ax \leq b, \\ & \quad \quad x \geq 0, \\ \text{(dual)} \quad & \min \quad b^\top \lambda \\ & \text{s.t.} \quad A^\top \lambda \geq c, \\ & \quad \quad \lambda \geq 0. \end{aligned} \quad (5.19)$$

The pair is written in resource-allocation form, a maximisation with explicit sign constraints and no equalities, rather than in the minimisation template of [Model 4.6](#). Any LP can be brought to this form: negate a cost vector, split each free variable into two nonnegative parts as in [Week 4](#), and write each equality as two opposite inequalities. In the primal, x_j is the level of activity j (how much of campaign, product or route j to run), c_j is the value of one unit of it, not a cost as in [Model 4.6](#), b_i is the available amount of resource i , and A_{ij} is the amount of resource i used by one unit of activity j (the letters A, b change meaning once more); value and resource use are assumed proportional to the activity levels. Both programmes are solved by any LP solver, as in [Listing 5.1](#); what is new is how the two solutions are read together. The dual has one variable λ_i per resource, the multiplier of [Definition 5.7](#) (many LP texts call this vector y), and its sign restrictions mirror the constraint directions of the primal.

The dual is not a new definition; it is (5.10) computed for this problem. Write the primal as the minimisation of $-c^\top x$ subject to $Ax - b \leq 0$ and $-x \leq 0$, with multipliers $\lambda \geq 0$ and $s \geq 0$ respectively (here s is a multiplier, which will turn out to be the slack $A^\top \lambda - c$ of the dual constraints, not the primal slack variable of [Week 4](#)). Then

$$L(x, \lambda, s) = -c^\top x + \lambda^\top (Ax - b) - s^\top x = -b^\top \lambda + (A^\top \lambda - c - s)^\top x,$$

which is affine in x . An affine function is unbounded below unless its slope is zero, so $g(\lambda, s) = -b^\top \lambda$ if $A^\top \lambda - c = s$, and $g(\lambda, s) = -\infty$ otherwise. Dual feasibility therefore requires $A^\top \lambda - c = s \geq 0$; eliminating s and replacing “maximise $-b^\top \lambda$ ” by “minimise $b^\top \lambda$ ” gives the dual in (5.19). Because the primal is now a maximisation, the bounds change sides: for primal-feasible x and dual-feasible λ ,

$$c^\top x \leq (A^\top \lambda)^\top x = \lambda^\top (Ax) \leq b^\top \lambda, \quad (5.20)$$

where the first step uses $A^\top \lambda \geq c$ with $x \geq 0$ and the last uses $Ax \leq b$ with $\lambda \geq 0$. Feasible plans give lower bounds on the optimal value, feasible prices give upper bounds, and by [Theorem 5.13](#) the two optimal values coincide whenever the primal is feasible and bounded.

Note on the lecture slides

The general theory (and frame 5.40) uses a primal *minimisation*, where feasible points give upper bounds; frames 5.34–5.39 use a primal *maximisation* without saying that the directions reverse, as in (5.20). Frame 5.34 also joins the two programmes by “ $\Longleftrightarrow$ ”, although they are a primal–dual pair, not equivalent problems.

Economically, λ_i is a price per unit of resource i . The dual constraint $(A^\top \lambda)_j \geq c_j$ says that the resources consumed by one unit of activity j , valued at these prices, are worth at least what the activity earns: no activity earns more than its resource bill. Subject to this, the dual minimises the total value $b^\top \lambda$ of the resources on hand. The quantity $(A^\top \lambda)_j - c_j \geq 0$ is called the **reduced cost** of activity j .

Proposition 5.19 (Complementary slackness for linear programmes). *Let x^* be primal feasible and λ^* dual feasible for (5.19), and let $a_i^\top$ denote row i of A . Then x^* and λ^* are both optimal if and only if*

$$\lambda_i^* (b_i - a_i^\top x^*) = 0 \quad (i = 1, \dots, m) \quad \text{and} \quad x_j^* ((A^\top \lambda^*)_j - c_j) = 0 \quad (j = 1, \dots, d).$$

Proof. In (5.20) the gap in the first inequality is $\sum_j x_j^* ((A^\top \lambda^*)_j - c_j)$ and the gap in the second is $\sum_i \lambda_i^* (b_i - a_i^\top x^*)$; all terms are nonnegative. Both gaps vanish exactly when $c^\top x^* = b^\top \lambda^*$, which by weak and strong duality happens exactly when both points are optimal. $\square$

In words: an unused resource has zero price, and an activity that is run at a positive level has zero reduced cost, so it exactly breaks even at the optimal prices. This explains why only a few constraints and activities are critical at an optimum. The conditions are the KKT complementary slackness conditions (5.13) for the two groups of inequalities, with the reduced costs $s^* = A^\top \lambda^* - c$ in the role of the multipliers of $x \geq 0$.

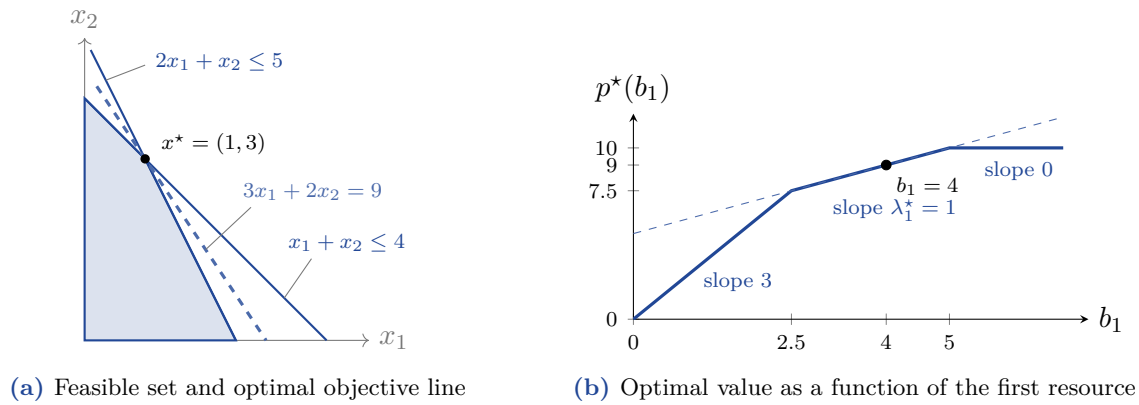


Figure 5.4: The resource-allocation LP of Example 5.20. (a) Both resource constraints are active at $x^* = (1, 3)$, where the objective line $3x_1 + 2x_2 = 9$ last touches the feasible set. (b) The optimal value $p^*(b_1)$ is concave and piecewise linear. Its slope at $b_1 = 4$ is the shadow price $\lambda_1^* = 1$, and the linear prediction (dashed) is exact for $2.5 \leq b_1 \leq 5$ and an overestimate outside that range.

5.5.2 A worked resource-allocation LP

Example 5.20 (A two-resource LP and its dual). Two activities share two resources:

$$\begin{array}{ll}
 \max & 3x_1 + 2x_2 \\
 \text{s.t.} & x_1 + x_2 \leq 4, \\
 & 2x_1 + x_2 \leq 5, \\
 & x_1, x_2 \geq 0,
 \end{array}
 \qquad
 \begin{array}{ll}
 \min & 4\lambda_1 + 5\lambda_2 \\
 \text{s.t.} & \lambda_1 + 2\lambda_2 \geq 3, \\
 & \lambda_1 + \lambda_2 \geq 2, \\
 & \lambda_1, \lambda_2 \geq 0.
 \end{array}$$

The dual on the right follows from (5.19) with $c = (3, 2)$, $b = (4, 5)$ and the rows (1, 1) and (2, 1) of A : its objective uses b , and its constraints use the *columns* of A . *Primal*. The feasible set is the polygon with vertices $(0, 0)$, $(2.5, 0)$, $(1, 3)$ and $(0, 4)$ (left panel of Figure 5.4), with objective values 0, 7.5, 9 and 8. The candidate is $x^* = (1, 3)$, where both resource constraints are active: $1 + 3 = 4$ and $2 + 3 = 5$. *Dual, by complementary slackness*. Since $x_1^* > 0$ and $x_2^* > 0$, both dual constraints must hold with equality: $\lambda_1 + 2\lambda_2 = 3$ and $\lambda_1 + \lambda_2 = 2$, so $\lambda^* = (1, 1)$, which is nonnegative and hence dual feasible. *Certificate*. The primal value is $3 + 6 = 9$ and the dual value is $4 + 5 = 9$. The bounds match, so both points are optimal; no comparison with other vertices is needed. Other feasible points illustrate (5.20): the plan $x = (2, 1)$ earns 8 and the prices $(2, 0.5)$ value the resources at 10.5, so $8 \leq 9 \leq 10.5$. *Interpretation*. Each resource is worth 1 per unit. A unit of activity 1 uses resources worth $1 + 2 = 3 = c_1$ and a unit of activity 2 resources worth $1 + 1 = 2 = c_2$: both activities break even, consistent with both being run.

5.5.3 Shadow prices

The third managerial question, what one more unit is worth, is answered by the dual solution. Let $p^*(b)$ denote the optimal value of the primal in (5.19) as a function of the right-hand side b .

Definition 5.21 (Shadow price). The **shadow price** of resource constraint i is its optimal multiplier λ_i^* . It is the marginal value of one more unit of the right-hand side b_i , in the sense that for a small change Δb of the resource vector

$$p^*(b + \Delta b) \approx p^*(b) + \lambda^{*\top} \Delta b. \quad (5.21)$$

The reason is visible in (5.19): the dual feasible set does not involve b . The prices λ^* therefore remain dual feasible when b changes, and weak duality (5.20) gives $p^*(b + \Delta b) \leq (b + \Delta b)^\top \lambda^* = p^*(b) + \lambda^{*\top} \Delta b$ for *every* Δb , the last step being strong duality at b , $b^\top \lambda^* = p^*(b)$. The prediction

is never too low, and it is exact while λ^* remains optimal for the new right-hand side, that is, while the same constraints stay active. Formula (5.21) is thus a first-order, local statement: a larger change can alter the active set, and with it the price. At a right-hand side where the active set changes, the optimal dual solution is not unique and the value of one unit more differs from that of one unit less (Example 5.22). The formula is also tied to the maximisation convention. For the minimisation problem (5.8), relax the i th inequality to $f_i(x) \leq u_i$ and write $p^*(u)$ for the perturbed optimal value; then $\partial p^*/\partial u_i = -\lambda_i^*$ at $u = 0$ (where the derivative exists), because loosening a constraint can only lower a minimal cost, at the rate λ_i^* .

Example 5.22 (Perturbing a resource). In Example 5.20 raise the first resource from $b_1 = 4$ to 4.2. If both resource constraints stay active, then $x_1 + x_2 = b_1$ and $2x_1 + x_2 = 5$ give $x = (5 - b_1, 2b_1 - 5) = (0.8, 3.4)$, which is nonnegative, with value $3(0.8) + 2(3.4) = 9.2$. The shadow-price prediction is $9 + \lambda_1^* \cdot 0.2 = 9.2$: exact, because the active set has not changed. The formula $x = (5 - b_1, 2b_1 - 5)$ remains feasible for $2.5 \leq b_1 \leq 5$, and there $p^*(b_1) = 5 + b_1$. For $b_1 < 2.5$ only the first resource binds, $x = (b_1, 0)$, $\lambda = (3, 0)$ and $p^* = 3b_1$; for $b_1 > 5$ the first resource is no longer scarce, $x = (0, 5)$, $\lambda = (0, 2)$ and $p^* = 10$ (right panel of Figure 5.4). At $b_1 = 6$ the prediction would be 11 but the true value is 10. At the breakpoint $b_1 = 5$ both $(1, 1)$ and $(0, 2)$ are optimal prices: a unit less costs 1, a unit more is worth 0. The firm should thus pay at most 1 per unit for a little more of resource 1, and nothing beyond $b_1 = 5$.

5.5.4 Reading solver statuses through duality

The fourth question concerns the statuses of Definition 1.7. When a solver returns “infeasible” or “unbounded” there is no solution to certify, but duality still has something to say. We return to the convention of (5.8) and (5.10).

Proposition 5.23 (Status implications). *For a primal minimisation and its dual maximisation: (a) if the primal is unbounded below, the dual is infeasible; (b) if the dual is unbounded above, the primal is infeasible.*

Proof. Both are weak duality. (a) If $p^* = -\infty$, then $g(\lambda, \nu) \leq p^*$ forces $g(\lambda, \nu) = -\infty$ for all $\lambda \geq 0$ and ν , so no pair is dual feasible. (b) If some x were primal feasible, (5.11) would bound every dual value by $f_0(x) < \infty$. $\square$

These implications are safe; their converses need more care, since a primal and its dual can both be infeasible. The infeasible and unbounded examples of Section 4.1, the second written as a minimisation, show the typical modelling mistakes. The *contradictory model* $\min x_1$ subject to $x_1 \geq 1$ and $x_1 \leq 0$ has no feasible point. Its Lagrangian is $x_1 + \lambda_1(1 - x_1) + \lambda_2 x_1 = \lambda_1 + (1 - \lambda_1 + \lambda_2)x_1$, so the dual is $\max \lambda_1$ subject to $\lambda_1 - \lambda_2 = 1$, $\lambda \geq 0$; the choices $\lambda = (1 + t, t)$ are feasible with value $1 + t$ for every $t \geq 0$, and the dual is unbounded above, in accordance with (b). The *missing-bound model* $\min -x_1$ subject to $x_1 \geq 0$ improves forever. Its Lagrangian is $-x_1 - \lambda x_1 = -(1 + \lambda)x_1$, whose infimum over x_1 is $-\infty$ for every $\lambda \geq 0$: no multiplier is dual feasible, in accordance with (a). This is the dual side of the diagnosis in Section 4.5. When a solver reports only “infeasible or unbounded” (Gurobi’s code 4), any dual-feasible point rules out an unbounded primal by (a), and a dual that is unbounded above proves the primal infeasible by (b).

5.5.5 Checking duality numerically in RSOME

RSOME remains the modelling layer of Section 4.5, but this week we deliberately do not ask the solver for dual variables. A recipe that makes the theory visible is: write and solve the primal; write and solve the dual as a second, separate model; compare the two optimal values and identify the active constraints; then perturb one right-hand side, re-solve, and compare the actual change with the shadow-price prediction. Listing 5.1 does this for Example 5.20.

Listing 5.1: Primal and dual of [Example 5.20](#) as two separate RSome models.

```

from rsome import ro
from rsome import grb_solver as grb
P = ro.Model()                                # primal LP
x = P.dvar(2)
P.max(3*x[0] + 2*x[1])
P.st(x[0] + x[1] <= 4)
P.st(2*x[0] + x[1] <= 5)
P.st(x >= 0)
P.solve(grb)

D = ro.Model()                                # dual LP
lam = D.dvar(2)
D.min(4*lam[0] + 5*lam[1])
D.st(lam[0] + 2*lam[1] >= 3)
D.st(lam[0] + lam[1] >= 2)
D.st(lam >= 0)
D.solve(grb)
print(P.solution.status, D.solution.status)
print(P.get(), D.get())                        # both 9.0: the bounds match

```

Each block follows the pattern of [Section 1.6](#): create a model, declare the variables, state the objective and constraints, and solve. The dual variable is named `lam` because `lambda` is a reserved word in Python. After RSome’s solver messages, the listing prints 2 2, Gurobi’s status code for “optimal” on both models, and then 9.0 9.0; `x.get()` and `lam.get()` return (1, 3) and (1, 1). With `msk_solver` the status is the word `Optimal` and the values are the same. Changing 4 to 4.2 and re-solving gives 9.2 and $x = (0.8, 3.4)$, as in [Example 5.22](#).

5.6 Summary and looking ahead

This week answered the four questions a manager may put to a solver’s output. Optimality can be *certified*: without constraints by $\nabla f(x^*) = 0$ for a smooth convex objective and $0 \in \partial f(x^*)$ for a nonsmooth one, with constraints by multipliers satisfying the KKT conditions. Optimality together with Slater’s condition forces these conditions, because matching primal and dual values squeeze the chain (5.12), and conversely any KKT point of a convex problem is a certified optimum, with no Slater point needed. The constraints that matter are the active ones with positive multipliers, and the multipliers are prices: in a linear programme they are shadow prices, valid while the active set does not change. Finally, weak duality turns the statuses “infeasible” and “unbounded” into diagnostic clues.

All of this rests on convexity, and a certificate says that a given point is optimal, not how to find it. [Week 4](#) asked how to formulate convex models and this week how to certify and interpret their solutions; [Week 6](#) asks how an algorithm finds them, beginning with the first template in (5.17), where gradient descent drives $\|\nabla f(x^k)\|_2$ to zero. The nonsmooth and constrained variants follow in [Weeks 7](#) and [8](#), and the Lagrangian returns as an algorithmic tool in [Weeks 12](#) and [13](#).

5.7 Exercises

Exercise 5.1 (True or false). Decide whether each statement is true, and correct the false ones.

- (a) “An active inequality must have a positive multiplier.” (b) “A dual-feasible point gives a valid bound.” (c) “An equality multiplier may be positive or negative.”

Exercise 5.2 (Unconstrained certificates). Following [Example 5.3](#), solve $\min x_1^2 + x_1x_2 + x_2^2 - 3x_1 - 3x_2 + 4$ over $\mathbb{R}^2$ by the gradient condition and explain why the point found is the unique global minimiser. Then, as in [Example 5.4](#), write the optimality condition for the ridge objective $f(x) = \|Ax - b\|_2^2 + \rho \|x\|_2^2$ with $\rho > 0$, and explain why its solution is unique even when the columns of A are dependent (compare [Proposition 1.13](#)).

Exercise 5.3 (Complementary slackness from strong duality). Assume that strong duality holds and that x^* and (λ^*, ν^*) are primal and dual optimal. Starting from $g(\lambda^*, \nu^*) \leq L(x^*, \lambda^*, \nu^*) \leq f_0(x^*)$, explain why x^* minimises the Lagrangian $L(\cdot, \lambda^*, \nu^*)$ and why $\lambda_i^* f_i(x^*) = 0$ for every i .

Exercise 5.4 (Equality-only KKT geometry). Repeat [Example 5.6](#) for the line $x_1 + 2x_2 = 2$: find its point closest to $(3, 1)$ by substitution and by a multiplier, and sketch the geometry. Then compute the dual function $g(\nu)$ as in [Example 5.10](#) and verify that $d^* = p^*$.

Exercise 5.5 (Mixed KKT geometry). In problem (5.6) replace the centre $(4, 1)$ first by $(1, 4)$ and then by $(2, 1)$. In each case identify the active set, solve for the multipliers, verify the four KKT blocks, and sketch the analogue of the right panel of [Figure 5.3](#).

Exercise 5.6 (LP duality and shadow prices). (a) Proceeding as in [Example 5.20](#), for the LP of [Example 1.6](#), $\max 2x_1 + 3x_2$ subject to $x_1 \leq 3$, $x_2 \leq 3$, $x_1 + x_2 \leq 5$, $x \geq 0$, derive the dual, solve both problems, and verify complementary slackness. Which resource has shadow price zero, and why? (b) In [Example 5.20](#), over which range of the second right-hand side b_2 (with $b_1 = 4$) does $\lambda_2^* = 1$ remain its shadow price?

Exercise 5.7 (An ℓ_1 -regularised optimality condition). Using $\partial|\beta_j|$ componentwise, write the optimality condition for $\min_{\beta} \frac{1}{2} \|A\beta - b\|_2^2 + \lambda \|\beta\|_1$ and interpret what it says when $\beta_j^* = 0$. For which values of λ is $\beta^* = 0$ optimal?

Exercise 5.8 (RSOME mini-lab). Code the primal and the dual of [Example 5.20](#) and confirm that the optimal values agree. Increase the first right-hand side from 4 to 4.2 and compare the actual change with the shadow-price prediction. Repeat with the second right-hand side raised from 5 to 5.2, and with the first raised to 6; explain any discrepancy.

Week 6 Smooth Unconstrained Optimisation and Gradient Descent

How do we reach x^* when we cannot solve $\nabla f(x) = 0$? [Week 5](#) gave us a certificate: for a differentiable convex function, a point is optimal exactly when its gradient vanishes ([Theorem 5.1](#)). A certificate tells us how to *recognise* a solution, not how to *find* one. So far the finding has been delegated to a solver called through RSOME. This week we open the box and build the first algorithm of the course, *gradient descent*, in the cleanest possible setting: minimise $f(x)$ over $x \in \mathbb{R}^d$, where $f: \mathbb{R}^d \rightarrow \mathbb{R}$ is convex and differentiable (twice continuously differentiable wherever a Hessian appears) and there are no constraints. The tools are in place: gradients, the tangent approximation and the chain rule from [Week 1](#), and Hessians and the first-order inequality for convex functions from [Week 3](#).

The week follows one line of argument. [Section 6.1](#) explains why we iterate and fixes a template with three design choices: a direction, a step size and a stopping rule. [Section 6.2](#) derives the gradient step as the minimiser of a local model; [Section 6.3](#) chooses the step size; [Section 6.4](#) asks how fast the method converges and when to stop, which is governed by a curvature cap L and a curvature floor μ ; and [Section 6.5](#) applies the method to least squares and logistic loss and closes with a verdict. [Weeks 7 to 10](#) reuse the template, changing one ingredient at a time.

6.1 From optimality conditions to an iterative template

6.1.1 Solvers and first-order methods are complements

Solver technology remains essential in this course: it is how we model, benchmark and diagnose ([Section 4.5](#)). Large smooth models, however, often have structure that a lighter method can exploit: gradients that cost a few matrix–vector products, and repeated solves.

Definition 6.1 (First-order method). A **first-order method** is an algorithm that uses only gradients (first derivatives) of the objective, together with function values: no Hessians and no calls to a full solver.

The name echoes the *first-order approximation* of [Equation \(1.4\)](#): both use first derivatives only. The two routes are complements: RSOME with a solver gives a trustworthy reference answer on a small or medium instance, and a hand-written first-order method in NumPy scales further and lets us inspect iterates, step sizes and gradient norms.

6.1.2 The template

The optimality condition $\nabla f(x^*) = 0$ is a system of as many nonlinear equations as there are variables. Outside special cases, such as least squares with its normal equations ([Section 5.1](#)), it has no closed-form solution. We therefore generate a *sequence* of points

$$x^{k+1} = x^k + t_k d^k, \quad k = 0, 1, 2, \dots, \quad (6.1)$$

starting from an initial guess x^0 . Here k counts iterations, $x^k \in \mathbb{R}^d$ is the current iterate (the superscript is an index, not a power), $d^k \in \mathbb{R}^d$ is the **search direction** and $t_k > 0$ is the **step size**. Each update should be cheap and should move towards a point where the optimality condition nearly holds. Designing such a method means making three choices: which direction looks promising, how far it can be trusted, and when the current point is good enough to stop. Every algorithm in the remaining weeks is organised around the same three choices.

Three remarks on notation. This week writes the step size as t_k (or t when it is fixed); from [Week 7](#) onwards it is written α_k . Secondly, the plain letter d remains the dimension of x , whereas

d^k , d^* , subscripted letters such as d_{scaled} and, in [Section 6.2.2](#) only, a bare d denote directions. Finally, three letters of [Week 5](#) take new jobs: L is now a curvature constant, not the Lagrangian $L(x, \lambda, \nu)$; μ is a curvature floor, not the vector of mean returns of the portfolio model; and d^* is a direction, not the dual optimal value.

6.1.3 Stationarity as the target

Recall from [Theorems 3.10](#) and [5.1](#) that x^* minimises a differentiable convex f if and only if $\nabla f(x^*) = 0$; this is the stationarity condition of [Definition 5.14](#) when there are no constraints. An algorithm can therefore measure its distance from optimality by the **gradient norm** $\|\nabla f(x^k)\|_2$, the residual of the optimality condition. Driving this residual to zero is the job of gradient descent, and “ $\|\nabla f(x^k)\|_2 \leq \varepsilon$ ” will be our first stopping rule. Four examples run through the week: one-variable quadratics; the two-variable quadratic $2x_1^2 + x_2^2 - 4x_1 - 2x_2 + 5$, whose minimiser $(1, 1)$ was certified by $\nabla f = 0$ in [Section 5.1](#); least squares, whose normal equations were derived there; and the logistic loss of [Week 4](#).

6.2 Local models and the descent direction

6.2.1 Slope, curvature and Taylor’s formula

For a function of one variable, the slope $f'(x)$ gives the direction of change and the curvature $f''(x)$ says how quickly the slope itself changes. For $f(x) = 0.55(x - 2.3)^2 + 0.55$ the slope is -1.21 at $x = 1.2$ and -0.44 at $x = 1.9$: it has risen by 0.77 over a distance of 0.7 , and the ratio $0.77/0.7 = 1.1 = f''$ is the curvature at work. For algorithms, the slope picks the move and the curvature decides how far the linear picture can be trusted. In $\mathbb{R}^d$ the slope becomes the gradient $\nabla f(x)$ ([Definition 1.15](#)) and the curvature becomes the Hessian $\nabla^2 f(x)$; recall from [Theorem 3.12](#) that f is convex exactly when $\nabla^2 f(x) \succeq 0$ for all x . The one quantity we use throughout is the **directional curvature** $u^\top \nabla^2 f(x) u$, which says how much f bends at x along the direction u .

The tangent approximation of [Equation \(1.4\)](#) says $f(y) \approx f(x) + \nabla f(x)^\top (y - x)$. The new ingredient this week is an exact formula for the gap.

Proposition 6.2 (Taylor’s formula with integral remainder). *Let $f : \mathbb{R}^d \rightarrow \mathbb{R}$ be twice continuously differentiable. For all $x, y \in \mathbb{R}^d$,*

$$f(y) = \underbrace{f(x) + \nabla f(x)^\top (y - x)}_{\text{tangent prediction}} + \underbrace{\int_0^1 (1-s)(y-x)^\top \nabla^2 f(x + s(y-x))(y-x) ds}_{\text{remainder: curvature accumulated from } x \text{ to } y}. \quad (6.2)$$

For $d = 1$ the remainder reads $\int_0^1 (1-s) f''(x + s(y-x))(y-x)^2 ds$.

Proof. Restrict f to the segment from x to y : let $\varphi(s) = f(x + s(y-x))$ for $s \in [0, 1]$, so that $\varphi(0) = f(x)$ and $\varphi(1) = f(y)$. By the fundamental theorem of calculus and one integration by parts (with $-(1-s)$ as the antiderivative of 1),

$$\varphi(1) - \varphi(0) = \int_0^1 \varphi'(s) ds = \left[-(1-s)\varphi'(s) \right]_0^1 + \int_0^1 (1-s)\varphi''(s) ds = \varphi'(0) + \int_0^1 (1-s)\varphi''(s) ds.$$

The chain rule ([Proposition 1.17](#)) gives $\varphi'(s) = \nabla f(x + s(y-x))^\top (y-x)$ and $\varphi''(s) = (y-x)^\top \nabla^2 f(x + s(y-x))(y-x)$. Substituting, with $\varphi'(0) = \nabla f(x)^\top (y-x)$, yields (6.2). $\square$

The straight segment from x to y reduces the multivariate question to curvature accumulated along one line: the integrand is the directional curvature of f along $y-x$ at the points of that segment. The left panel of [Figure 6.1](#) illustrates the formula for $f(x) = 0.35(x - 2.3)^2 + 0.6$ with

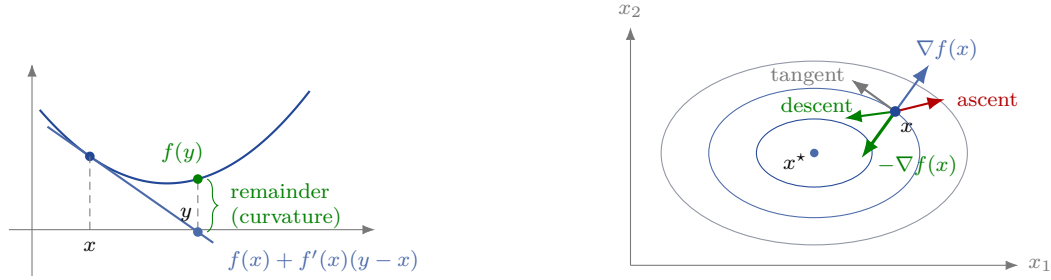


Figure 6.1: Left: the tangent prediction at x and the true value $f(y)$; the gap is the Taylor remainder of Proposition 6.2, nonnegative for convex f . Right: directions at a point x of a contour. A direction is downhill (descent), tangent (no first-order change) or uphill (ascent) according to the sign of its inner product with $\nabla f(x)$, which is perpendicular to the contour; $-\nabla f(x)$ is the steepest descent direction.

$x = 1.3$ and $y = 2.7$: the tangent predicts $0.95 - 0.7 \cdot 1.4 = -0.03$, the true value is $f(y) = 0.656$, and the remainder is $\int_0^1 (1-s) 0.7 \cdot 1.4^2 ds = 0.686$, exactly the difference. If f is convex the integrand is nonnegative, so the tangent prediction lies below $f(y)$: we recover the first-order inequality of Theorem 3.8. More is true, and it drives the whole week. Since $\int_0^1 (1-s) ds = \frac{1}{2}$, a *cap* on the curvature, $u^\top \nabla^2 f(x) u \leq L \|u\|_2^2$ for all x and u , makes the remainder at most $\frac{L}{2} \|y - x\|_2^2$, so f lies *below* a quadratic (Section 6.3.3); a *floor*, $u^\top \nabla^2 f(x) u \geq \mu \|u\|_2^2$, makes it at least $\frac{\mu}{2} \|y - x\|_2^2$, so f lies *above* a quadratic (Section 6.4.3).

6.2.2 Descent directions and steepest descent

Before worrying about how far to move, we classify candidate directions. Recall from Definition 1.15 the directional derivative $Df(x; d) = \nabla f(x)^\top d$, the rate of change of f when we leave x along d . Taylor's formula with $y = x + td$ gives $f(x + td) = f(x) + t \nabla f(x)^\top d + t^2 R(t)$, where the remainder of (6.2) has been written as $t^2 R(t)$ and $R(t)$ stays bounded as $t \rightarrow 0$. For small $t > 0$ the term linear in t therefore dominates.

Definition 6.3 (Descent direction). A vector d is a **descent direction** of f at x if $\nabla f(x)^\top d < 0$. Then $f(x + td) < f(x)$ for all sufficiently small $t > 0$.

If $\nabla f(x)^\top d > 0$ the direction is uphill, and if $\nabla f(x)^\top d = 0$ it is tangent to the contour through x , with no first-order change (right panel of Figure 6.1). One downhill direction is steepest.

Proposition 6.4 (Steepest descent). If $\nabla f(x) \neq 0$, then

$$\min_{\|d\|_2=1} \nabla f(x)^\top d = -\|\nabla f(x)\|_2, \quad \text{attained at} \quad d^* = -\frac{\nabla f(x)}{\|\nabla f(x)\|_2}.$$

Proof. By the Cauchy–Schwarz inequality (Proposition 1.11), every unit vector d satisfies

$$\nabla f(x)^\top d \geq -\|\nabla f(x)\|_2 \|d\|_2 = -\|\nabla f(x)\|_2,$$

and d^* attains this bound. □

Among all unit Euclidean directions, the negative gradient gives the most negative first-order change; it is the **steepest-descent direction**, perpendicular to the contour. The qualifier “Euclidean” matters: another norm, or constraints on the move, change the best local move. We meet the first effect at the end of Section 6.4 and the second in Week 8.

6.2.3 Gradient descent as the minimiser of a local model

Around the current iterate, the cheapest description of f is the linear model $m_k(z) = f(x^k) + \nabla f(x^k)^\top(z - x^k)$. It tells us which direction looks promising *locally*, but by itself it is dangerous: unless $\nabla f(x^k) = 0$ it has no minimiser, because along $z = x^k - s\nabla f(x^k)$ its value $f(x^k) - s\|\nabla f(x^k)\|_2^2$ decreases without bound as s grows. The true function does not behave like this: curvature eventually bends it upwards. We therefore stabilise the model by charging for movement:

$$\min_{z \in \mathbb{R}^d} \phi(z) = f(x^k) + \nabla f(x^k)^\top(z - x^k) + \frac{1}{2t_k} \|z - x^k\|_2^2. \quad (6.3)$$

The first two terms are the local linear predictor; the quadratic term says *do not trust the linear model too far from x^k* , and a small t_k makes movement expensive. The function ϕ is a convex quadratic in z with Hessian $\frac{1}{t_k}I \succ 0$, so its unique minimiser is its stationary point:

$$\nabla_z \phi(z) = \nabla f(x^k) + \frac{1}{t_k}(z - x^k) = 0 \implies z = x^k - t_k \nabla f(x^k).$$

Taking this minimiser as the next iterate gives the **gradient-descent update**

$$x^{k+1} = x^k - t_k \nabla f(x^k), \quad (6.4)$$

which is the template (6.1) with $d^k = -\nabla f(x^k)$. The model value at the new point is $\phi(x^{k+1}) = f(x^k) - \frac{t_k}{2} \|\nabla f(x^k)\|_2^2$: this is the decrease the model *predicts*, and Section 6.3.3 will show when f actually delivers it. Gradient descent is best remembered not as a formula but as the minimiser of a local model: the tangent prediction plus a penalty for moving away, with the step size t_k as the trust placed in it.

Algorithm 6.1 Gradient descent

Require: differentiable convex f with gradient ∇f ; starting point x^0 ; step-size rule; tolerance $\varepsilon > 0$; iteration cap K

Ensure: an approximate minimiser of f

```

1: for  $k = 0, 1, \dots, K - 1$  do
2:    $g^k \leftarrow \nabla f(x^k)$ 
3:   if  $\|g^k\|_2 \leq \varepsilon$  then return  $x^k$  ▷ stopping test, Section 6.4.1
4:   choose  $t_k > 0$  by the step-size rule ▷ Section 6.3
5:    $x^{k+1} \leftarrow x^k - t_k g^k$ 
6: end for
7: return  $x^K$ 
```

Algorithm 6.1 costs one gradient evaluation and a few vector operations per iteration, plus whatever the step-size rule needs; we write $g^k = \nabla f(x^k)$ throughout. The local-model view is what generalises: adding a constraint $z \in \mathcal{X}$ to (6.3) gives the projected-gradient step of Week 8, and adding a nonsmooth penalty gives the proximal-gradient step of Week 9.

6.2.4 Worked example: gradient steps on a two-variable quadratic

Example 6.5 (The running quadratic). Let $f(x_1, x_2) = 2x_1^2 + x_2^2 - 4x_1 - 2x_2 + 5$ (the quadratic of Section 5.1), start at $x^0 = (0, 0)$ and use the fixed step $t = \frac{1}{4}$. The gradient and Hessian are

$$\nabla f(x) = \begin{bmatrix} 4x_1 - 4 \\ 2x_2 - 2 \end{bmatrix}, \quad \nabla^2 f(x) = \begin{bmatrix} 4 & 0 \\ 0 & 2 \end{bmatrix},$$

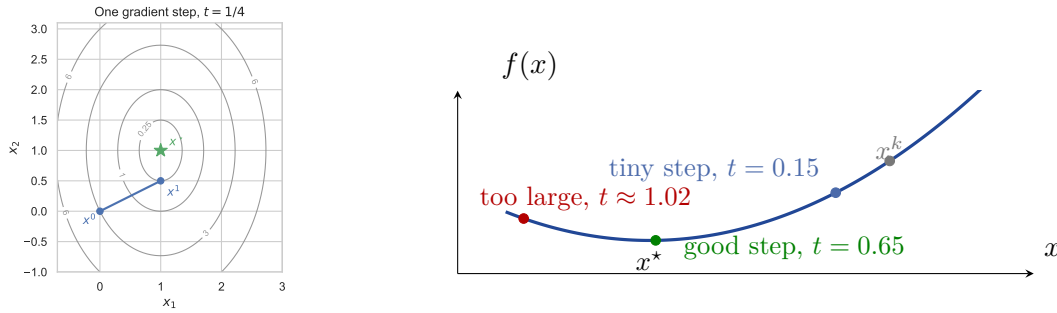


Figure 6.2: Left: one gradient step with $t = \frac{1}{4}$ in Example 6.5, on contours of $f - f^*$: x^0 lies on level 3 and x^1 lands exactly on level 0.25. Right: three step sizes from the same point $x^k = 3$ on $f(x) = 0.75(x - 1)^2 + 0.35$; same direction, different step size.

so, as found in Section 5.1, f is strictly convex with unique minimiser $x^* = (1, 1)$ and $f^* = f(x^*) = 2$. At the start $f(x^0) = 5$, so the objective gap is $f(x^0) - f^* = 3$, and $g^0 = \nabla f(x^0) = (-4, -2)$ with $\|g^0\|_2 = \sqrt{20} \approx 4.47$. One step gives $x^1 = x^0 - t g^0 = (0, 0) - \frac{1}{4}(-4, -2) = (1, \frac{1}{2})$ and $f(x^1) = 2.25$, so the gap has fallen from 3 to 0.25 (left panel of Figure 6.2). Continuing, $g^1 = (0, -1)$ and $x^2 = (1, \frac{3}{4})$, then $x^3 = (1, \frac{7}{8})$, and in general $x^k = (1, 1 - 2^{-k})$ for $k \geq 1$, with gap $f(x^k) - f^* = 4^{-k}$ and residual $\|g^k\|_2 = 2^{1-k}$. The two coordinates behave differently because the update acts on them separately: $x_1^{k+1} - 1 = (1 - 4t)(x_1^k - 1)$ and $x_2^{k+1} - 1 = (1 - 2t)(x_2^k - 1)$. With $t = \frac{1}{4}$ the first factor is 0, so the high-curvature coordinate is solved in one step, while the second is $\frac{1}{2}$, so the low-curvature coordinate halves its error each iteration.

6.3 Choosing the step size

6.3.1 Why the step size matters

Direction alone is not enough. Take $f(x) = 0.75(x - 1)^2 + 0.35$, whose minimiser is $x^* = 1$, and stand at $x^k = 3$, where $f(x^k) = 3.35$ and $f'(x^k) = 3$. The gradient step is $x^{k+1} = 3 - 3t$, and the right panel of Figure 6.2 shows three choices. With $t = 0.15$ we reach 2.55 and $f = 2.152$: progress, but slow. With $t = 0.65$ we reach 1.05 and $f = 0.352$, almost the minimum. With $t \approx 1.02$ we jump across the valley to -0.05 , where $f = 1.177$: the step overshoots the minimiser and lands on the opposite slope, keeping only 72% of the decrease that was available. In general $x^{k+1} - 1 = (1 - 1.5t)(x^k - 1)$, since $f'(x) = 1.5(x - 1)$; the error shrinks only if $|1 - 1.5t| < 1$, that is, $0 < t < \frac{4}{3}$, and for $t > \frac{4}{3}$ every step *increases* the objective. Likewise in Example 6.5 the first coordinate has the factor $1 - 4t$, and its error shrinks only if $|1 - 4t| < 1$, that is, $t < \frac{1}{2}$: with $t = 0.55$ the gap after 25 steps is 1.8×10^4 . Step-size design is as important as the choice of direction. Both thresholds, $\frac{4}{3}$ and $\frac{1}{2}$, are of the form $2/(\text{curvature})$; the rest of this subsection explains why.

6.3.2 Quadratics and exact line search

Quadratic functions are the laboratory in which step sizes can be computed by hand. Let

$$f(x) = \frac{1}{2}x^\top Qx + q^\top x + r, \quad Q = Q^\top \in \mathbb{R}^{d \times d}, \quad q \in \mathbb{R}^d, \quad r \in \mathbb{R}. \quad (6.5)$$

Then $\nabla f(x) = Qx + q$ and $\nabla^2 f(x) = Q$: the curvature does not change from point to point, and f is convex exactly when $Q \succeq 0$ (Proposition 3.14). The running example has $Q = \text{diag}(4, 2)$, $q = (-4, -2)$ and $r = 5$. An important family is $Q = A^\top A + \mu I$, for a matrix A with d columns and $\mu \geq 0$, for which $u^\top Qu = \|Au\|_2^2 + \mu \|u\|_2^2 \geq \mu \|u\|_2^2$ for all $u \in \mathbb{R}^d$: the directional curvature

is at least μ along every unit direction. When $\mu > 0$ this is the property that [Definition 6.12](#) will call strong convexity.

The most ambitious step-size rule asks for the best point on the whole search ray.

Definition 6.6 (Exact line search). Given the iterate x^k and a descent direction d^k , **exact line search** chooses $t_k \in \operatorname{argmin}_{t \geq 0} f(x^k + t d^k)$. Here argmin denotes the *set* of minimisers, hence “ $\in$ ”; when the minimiser is unique we also write “ $=$ ”.

Along a search direction the multivariate problem becomes a minimisation over the single variable t . For a general f this problem must itself be solved numerically at every iteration, which is why exact line search is rarely used in practice. For a quadratic it has a closed form.

Proposition 6.7 (Exact line search on a quadratic). *Let f be the quadratic (6.5), let $g^k = \nabla f(x^k)$ and $d^k = -g^k$, and suppose $(g^k)^\top Q g^k > 0$ (for instance $Q \succ 0$ and $g^k \neq 0$). Then the exact line-search step is $t_k = (g^k)^\top g^k / (g^k)^\top Q g^k$.*

Proof. Let $\phi(t) = f(x^k - t g^k)$, a function of the scalar t (not the local model $\phi(z)$ of [Section 6.2.3](#)). By the chain rule, $\phi'(t) = -\nabla f(x^k - t g^k)^\top g^k = -(Q(x^k - t g^k) + q)^\top g^k = -(g^k)^\top g^k + t (g^k)^\top Q g^k$, using $Q x^k + q = g^k$. The function ϕ is a convex parabola in t because $\phi''(t) = (g^k)^\top Q g^k > 0$, so its minimiser solves $\phi'(t) = 0$, and this minimiser is positive. $\square$

The step is the squared gradient norm divided by the directional curvature along the gradient: high curvature along g^k means a short step. In [Example 6.5](#), $(g^0)^\top g^0 = 20$ and $(g^0)^\top Q g^0 = 4 \cdot 16 + 2 \cdot 4 = 72$, so $t_0 = \frac{5}{18} \approx 0.278$ and $x^1 = (\frac{10}{9}, \frac{5}{9})$, with gap $\frac{2}{9} \approx 0.222$, slightly better than the 0.25 reached with $t = \frac{1}{4}$, as it must be. The new gradient $g^1 = (\frac{4}{9}, -\frac{8}{9})$ is perpendicular to g^0 , as $\phi'(t_0) = 0$ requires: exact line search moves in right-angled turns.

6.3.3 L -smoothness and the safe fixed step

A fixed step $t_k = t$ is the simplest rule of all, but the experiments above show that it must be matched to the curvature. The assumption that makes this possible is a cap on the curvature.

Definition 6.8 (L -smoothness). A differentiable function $f : \mathbb{R}^d \rightarrow \mathbb{R}$ is **L -smooth**, for a constant $L > 0$, if its gradient is L -Lipschitz: $\|\nabla f(x) - \nabla f(y)\|_2 \leq L \|x - y\|_2$ for all $x, y \in \mathbb{R}^d$. For a convex, twice continuously differentiable f this is equivalent to the upper Hessian bound $u^\top \nabla^2 f(x) u \leq L \|u\|_2^2$ for all $x, u \in \mathbb{R}^d$, that is, $\nabla^2 f(x) \preceq LI$ in the notation of [Definition 3.11](#).

The gradient cannot change faster than L times the distance moved; since the Hessian is the rate of change of the gradient, this is a *curvature cap*. Any larger constant is also a valid L , and we normally mean the smallest one we know. The running example, with $u^\top Q u = 4u_1^2 + 2u_2^2 \leq 4 \|u\|_2^2$, has $L = 4$; the function x^4 is not L -smooth for any L , because $f''(x) = 12x^2$ is unbounded.

Proposition 6.9 (Quadratic upper bound and descent lemma). *Let f be L -smooth. Then, for all $x, y \in \mathbb{R}^d$,*

$$f(y) \leq f(x) + \nabla f(x)^\top (y - x) + \frac{L}{2} \|y - x\|_2^2. \quad (6.6)$$

Consequently, for every x and every $t > 0$,

$$f(x - t \nabla f(x)) \leq f(x) - t \left(1 - \frac{Lt}{2}\right) \|\nabla f(x)\|_2^2, \quad (6.7)$$

and in particular $f(x - t \nabla f(x)) \leq f(x) - \frac{t}{2} \|\nabla f(x)\|_2^2$ whenever $0 < t \leq 1/L$.

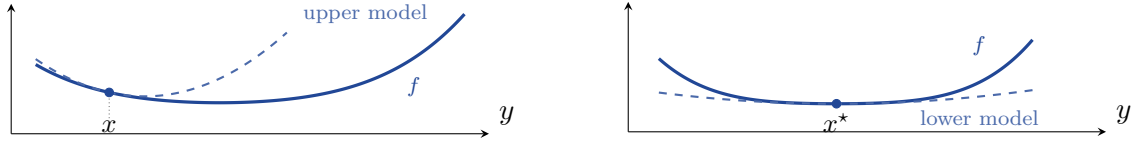


Figure 6.3: Cap and floor. Left: under a curvature cap L the upper quadratic model (dashed) touches f at x and stays above it (Proposition 6.9; here over the range shown, since the quartic drawn has no global cap); its minimiser is the gradient step with $t = 1/L$. Right: under a curvature floor μ the lower quadratic model stays below f (Definition 6.12); drawn at x^* .

Proof. Let $\varphi(s) = f(x + s(y - x))$, as in the proof of Proposition 6.2, so that $\varphi'(s) = \nabla f(x + s(y - x))^\top (y - x)$. The points $x + s(y - x)$ and x are $s \|y - x\|_2$ apart, so the Cauchy–Schwarz inequality and Definition 6.8 give $\varphi'(s) - \varphi'(0) \leq Ls \|y - x\|_2^2$, and

$$f(y) - f(x) - \nabla f(x)^\top (y - x) = \int_0^1 (\varphi'(s) - \varphi'(0)) ds \leq \int_0^1 Ls \|y - x\|_2^2 ds = \frac{L}{2} \|y - x\|_2^2,$$

which is (6.6); with a Hessian, it is the remainder bound foreseen in Section 6.2. Now put $y = x - t \nabla f(x)$. Then $\nabla f(x)^\top (y - x) = -t \|\nabla f(x)\|_2^2$ and $\|y - x\|_2^2 = t^2 \|\nabla f(x)\|_2^2$, and collecting terms gives (6.7). If $0 < t \leq 1/L$ then $1 - \frac{Lt}{2} \geq \frac{1}{2}$. $\square$

The bound (6.6) says that f lies below a quadratic that touches it at x (left panel of Figure 6.3). Three consequences follow. First, the *safe fixed step*: if $0 < t \leq 1/L$, every gradient step strictly decreases the objective unless $\nabla f(x^k) = 0$, by at least $\frac{t}{2} \|\nabla f(x^k)\|_2^2$, the decrease that the local model predicted in Section 6.2.3. Second, the *majorisation view*: the right-hand side of (6.6), as a function of y , is the local model (6.3) with $\frac{1}{2t_k} = \frac{L}{2}$. With $t = 1/L$, gradient descent therefore repeatedly minimises a curve that sits above f and touches it at x^k . Third, (6.7) still gives a decrease for any $0 < t < 2/L$, because $1 - \frac{Lt}{2} > 0$; this explains the thresholds $\frac{4}{3} = 2/1.5$ and $\frac{1}{2} = 2/4$ observed above. The gradient tells us where to move, and the curvature cap tells us how far that move can be trusted.

6.3.4 Backtracking line search, and a comparison of the rules

The safe step needs L , which is often unknown or very pessimistic. A practical rule tries a step and shrinks it until the decrease is sufficient. For a descent direction d^k the **Armijo (sufficient-decrease) test** is

$$f(x^k + t d^k) \leq f(x^k) + \alpha t \nabla f(x^k)^\top d^k, \quad (6.8)$$

with a parameter $\alpha \in (0, 1)$. With $d^k = -g^k$ it reads $f(x^k - t g^k) \leq f(x^k) - \alpha t \|g^k\|_2^2$: the linear model predicts a decrease of $t \|g^k\|_2^2$, and the test accepts t once at least the fraction α of that prediction is realised. A failed trial is multiplied by a shrink factor $\beta \in (0, 1)$. Common defaults are $\alpha \approx 0.3$ and $\beta \approx 0.5$. These customary letters have other meanings later in the course: from Week 7 onwards α_k is the step size itself, and β becomes a vector of regression coefficients (Week 7) or a step-schedule or momentum constant.

Algorithm 6.2 Backtracking line search (for the gradient direction)

Require: iterate x^k , gradient $g^k = \nabla f(x^k) \neq 0$; parameters $\alpha, \beta \in (0, 1)$; trial step $t_{\text{init}} > 0$ (for example 1)

Ensure: a step size t_k that passes the Armijo test

```

1:  $t \leftarrow t_{\text{init}}$ 
2: while  $f(x^k - t g^k) > f(x^k) - \alpha t \|g^k\|_2^2$  do
3:    $t \leftarrow \beta t$ 
4: end while
5: return  $t_k = t$ 

```

Backtracking replaces a global curvature constant by a local accept-or-reject test, at the price of one function evaluation per trial, and it keeps the guarantees of the safe step. Suppose f is L -smooth and $\alpha \leq \frac{1}{2}$. For any $t \leq 1/L$, Proposition 6.9 gives $f(x^k - t g^k) \leq f(x^k) - \frac{t}{2} \|g^k\|_2^2 \leq f(x^k) - \alpha t \|g^k\|_2^2$, so the test is sure to pass once $t \leq 1/L$ and the loop terminates. Moreover, either the first trial is accepted, or the accepted t was preceded by a rejected trial $t/\beta > 1/L$; in both cases

$$t_k \geq \min\{t_{\text{init}}, \beta/L\}. \quad (6.9)$$

Every accepted step therefore decreases f by at least $\alpha t_k \|g^k\|_2^2$, with $\min\{t_{\text{init}}, \beta/L\} \leq t_k \leq t_{\text{init}}$. With small changes, the rate arguments of Section 6.4 then go through: the rates carry over, with other constants. The minimum in (6.9) cannot be dropped: if the first trial passes, the accepted step is t_{init} , which may be smaller than β/L (for $f(x) = 0.05x^2$, so $L = 0.1$, the trial $t = 1$ is accepted although $\beta/L = 5$).

On the running quadratic with $\alpha = 0.3$, $\beta = 0.5$ and $t_{\text{init}} = 1$, the test at x^0 reads $f(x^0 - t g^0) \leq 5 - 6t$: the trials $t = 1$ ($f = 21$) and $t = \frac{1}{2}$ ($f = 4$) fail and $t = \frac{1}{4}$ ($f = 2.25$) passes. At x^1 the loop accepts $t = \frac{1}{2}$, twice the safe step $1/L$, because the curvature along g^1 is only 2.

The three step-size rules trade cost against knowledge. Exact line search needs no L but solves a one-variable subproblem at each iteration, the highest cost; its role is geometric insight on quadratics. A fixed step is cheapest, with the cleanest analysis and the simplest code, but needs at least a safe overestimate of L ; it is our baseline for least squares. Backtracking has moderate cost, needs no explicit L and adapts to the local curvature: the most flexible default.

6.4 How fast, and when to stop

6.4.1 Measuring progress and stopping rules

An algorithm can be judged by three quantities: the **objective gap** $f(x^k) - f^*$, where f^* is the optimal value; the distance of the iterate from a solution; and the gradient norm. The first two involve the unknown solution, so a running algorithm uses the checks of Table 6.1 instead. The gradient norm is the most direct, being the residual of the optimality condition; the objective history shows whether progress is still meaningful; an iteration cap protects the budget but certifies nothing. Never stop because the screen “looks stable”: stop because a residual or a change metric says the point is close enough.

6.4.2 The sublinear rate for smooth convex functions

To state how fast the gap shrinks without tracking every constant, we use the following notation.

Definition 6.10 (Big- O ; sublinear and linear convergence). For a nonnegative sequence (a_k) we write $a_k = O(1/k)$ if $a_k \leq C/k$ for some constant $C > 0$ and all large k : the sequence shrinks *at least* as fast as a constant times $1/k$. More generally $a_k = O(b_k)$ means $a_k \leq C b_k$ for all large

Table 6.1: Stopping rules for gradient descent, with tolerance $\varepsilon > 0$.

Check	Meaning	Role
$\ \nabla f(x^k)\ _2 \leq \varepsilon$	nearly stationary point	primary optimality residual (Theorem 5.1)
$ f(x^{k+1}) - f(x^k) \leq \varepsilon$	objective barely changes	useful progress monitor
$\ x^{k+1} - x^k\ _2 \leq \varepsilon$	iterates barely move	useful but secondary
k reaches the iteration cap	budget exhausted	practical safeguard, not a certificate

k . A guarantee of the form $a_k \leq C/k$ is called a **sublinear** rate: the bound shrinks ever more slowly. A guarantee $a_{k+1} \leq \rho a_k$ with a constant factor $\rho \in (0, 1)$, so that $a_k \leq \rho^k a_0$, is a **linear** rate.

Theorem 6.11 (Gradient descent on a smooth convex function). *Let f be convex and L -smooth, suppose a minimiser x^* exists, and run gradient descent with a fixed step $0 < t \leq 1/L$. Then, for every $k \geq 1$,*

$$f(x^k) - f^* \leq \frac{\|x^0 - x^*\|_2^2}{2tk}, \quad \text{which is } \frac{L\|x^0 - x^*\|_2^2}{2k} \text{ for } t = \frac{1}{L}.$$

In particular $f(x^k) - f^* = O(1/k)$.

Proof. (1) *Decrease.* By [Proposition 6.9](#), $f(x^{k+1}) \leq f(x^k) - \frac{t}{2}\|g^k\|_2^2$. (2) *Convexity.* The first-order inequality ([Theorem 3.8](#)) for the tangent at x^k , evaluated at x^* , reads $f^* \geq f(x^k) + (g^k)^\top(x^* - x^k)$. Inserting this in step (1) and completing the square with $x^{k+1} - x^* = x^k - x^* - tg^k$,

$$f(x^{k+1}) - f^* \leq (g^k)^\top(x^k - x^*) - \frac{t}{2}\|g^k\|_2^2 = \frac{1}{2t}(\|x^k - x^*\|_2^2 - \|x^{k+1} - x^*\|_2^2).$$

Since the left-hand side is nonnegative, the distance to x^* never increases. (3) *Summation.* Adding these inequalities for iterations $0, \dots, k-1$, the right-hand side telescopes to at most $\frac{1}{2t}\|x^0 - x^*\|_2^2$. By step (1) the values $f(x^j)$ are non-increasing, so each of the k terms on the left is at least $f(x^k) - f^*$, which gives the bound. $\square$

The guarantee is global: it holds from any starting point, with no assumption beyond convexity, the curvature cap and the existence of a minimiser. With $t = 1/L$, a gap of at most ε is guaranteed once $k \geq L\|x^0 - x^*\|_2^2/(2\varepsilon)$. Progress is certain but it slows down: ten times the accuracy costs ten times the iterations. Step (1) also shows that the stopping test of [Algorithm 6.1](#) is eventually passed. Summing $f(x^j) - f(x^{j+1}) \geq \frac{t}{2}\|g^j\|_2^2$ over $j = 0, \dots, k-1$ gives $\frac{t}{2}\sum_{j=0}^{k-1}\|g^j\|_2^2 \leq f(x^0) - f^*$ for every k , hence $\min_{j < k}\|g^j\|_2^2 \leq 2(f(x^0) - f^*)/(tk)$: the gradient norm tends to zero, and $\|g^k\|_2 \leq \varepsilon$ holds for some $k \leq 2(f(x^0) - f^*)/(t\varepsilon^2)$. In the running example ($\|x^0 - x^*\|_2^2 = 2$, $t = \frac{1}{4}$) the bound is $4/k$ whereas the actual gap is 4^{-k} : this function has an additional property that makes gradient descent far faster.

6.4.3 Strong convexity and the linear rate

A convex function may be almost flat around its minimiser, as x^4 is near 0; then progress is slow. The property that excludes flatness is a curvature *floor*, the mirror image of the cap.

Definition 6.12 (Strong convexity). A function $f : \mathbb{R}^d \rightarrow \mathbb{R}$ is **μ -strongly convex**, for a constant $\mu > 0$, if $x \mapsto f(x) - \frac{\mu}{2}\|x\|_2^2$ is convex. For a twice continuously differentiable f this is the lower Hessian bound $u^\top \nabla^2 f(x) u \geq \mu\|u\|_2^2$ for all x, u , that is, $\nabla^2 f(x) \succeq \mu I$. For a differentiable f it is equivalent to the *lower quadratic bound*

$$f(y) \geq f(x) + \nabla f(x)^\top(y - x) + \frac{\mu}{2}\|y - x\|_2^2 \quad \text{for all } x, y \in \mathbb{R}^d. \quad (6.10)$$

The Hessian form holds because the Hessian of $f - \frac{\mu}{2} \|\cdot\|_2^2$ is $\nabla^2 f(x) - \mu I$ (Theorem 3.12). The bound (6.10) is the first-order inequality of Theorem 3.8 for $h(x) = f(x) - \frac{\mu}{2} \|x\|_2^2$, whose gradient is $\nabla f(x) - \mu x$, after collecting the terms in μ ; with a Hessian, it is the remainder bound foreseen in Section 6.2. The function lies above a quadratic that touches it (right panel of Figure 6.3): it cannot be too flat anywhere. At $x = x^*$ the bound reads $f(y) \geq f^* + \frac{\mu}{2} \|y - x^*\|_2^2$, so the minimiser is unique. The quadratics with $Q = A^\top A + \mu I$ are μ -strongly convex, and the running example has $\mu = 2$ and $L = 4$.

Theorem 6.13 (Linear rate and gradient certificate). *Let f be L -smooth and μ -strongly convex, with minimiser x^* and $f^* = f(x^*)$. Then for every $x \in \mathbb{R}^d$*

$$\|\nabla f(x)\|_2^2 \geq 2\mu (f(x) - f^*), \quad (6.11)$$

and gradient descent with the fixed step $t = 1/L$ satisfies

$$f(x^k) - f^* \leq \left(1 - \frac{\mu}{L}\right)^k (f(x^0) - f^*), \quad k = 0, 1, 2, \dots$$

Proof. Fix x and minimise both sides of (6.10) over y . The left-hand side gives f^* . The right-hand side is a convex quadratic in y , minimised where $\nabla f(x) + \mu(y - x) = 0$, with minimum value $f(x) - \frac{1}{2\mu} \|\nabla f(x)\|_2^2$. Thus $f^* \geq f(x) - \frac{1}{2\mu} \|\nabla f(x)\|_2^2$, which is (6.11). Now Proposition 6.9 with $t = 1/L$ gives $f(x^{k+1}) - f^* \leq f(x^k) - f^* - \frac{1}{2L} \|g^k\|_2^2 \leq (1 - \frac{\mu}{L})(f(x^k) - f^*)$, and the claim follows by induction. $\square$

Compared with the proof of Theorem 6.11, the curvature floor strengthens step (2): a large gap now forces a large gradient, hence a large decrease, so the recursion contracts geometrically instead of merely summing. A gap of at most ε is guaranteed once $k \geq \frac{L}{\mu} \ln \frac{f(x^0) - f^*}{\varepsilon}$ (because $1 - u \leq e^{-u}$): every additional digit costs the same number of iterations, and a larger floor relative to the cap means faster progress. Read backwards, (6.11) justifies the primary stopping rule of Table 6.1:

$$\|\nabla f(x^k)\|_2 \leq \varepsilon \implies f(x^k) - f^* \leq \frac{\varepsilon^2}{2\mu}.$$

In the running example the theorem promises $3 \cdot (\frac{1}{2})^k$ while the true gap is 4^{-k} , and at x^1 the certificate gives $\|g^1\|_2^2 / (2\mu) = \frac{1}{4}$, which is exactly the gap.

6.4.4 The two speeds on one picture

Figure 6.4 shows both rates on a least-squares instance with $N = 60$, $d = 30$ and columns of very different scales, solved with $t = 1/L$ from $x^0 = 0$ (Section 6.5 explains how L is computed). For the plain objective $\frac{1}{2} \|Ax - b\|_2^2$ the floor is tiny relative to the cap ($L/\mu \approx 6.6 \times 10^3$), and the gap falls only from 14.8 to 2.3 in 400 iterations. Adding the ridge-type penalty $\frac{\mu}{2} \|x\|_2^2$ with $\mu = 2$ raises the floor ($L/\mu \approx 37$): the gap falls from 4.8 to 2×10^{-11} , at about 36 iterations per digit. On a logarithmic axis linear convergence is a straight line, whereas the slow run flattens like the $O(1/k)$ guarantee, under which each extra digit costs ten times more iterations (the run itself gains less than one digit). Same algorithm, same step rule: only the floor differs.

6.4.5 Uneven curvature: easy and hard quadratics

Why does the ratio L/μ matter so much? Diagonal quadratics show the mechanism.

Example 6.14 (A round and an elongated quadratic). Compare

$$f_{\text{round}}(x) = \frac{1}{2}(x_1^2 + x_2^2) \quad \text{and} \quad f_{\text{elongated}}(x) = \frac{1}{2}(x_1^2 + 25x_2^2).$$

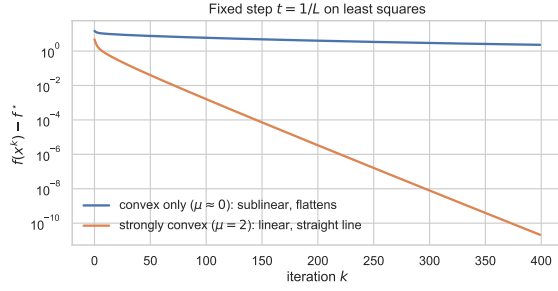


Figure 6.4: Measured gap $f(x^k) - f^*$ of gradient descent with $t = 1/L$ on a least-squares instance, logarithmic scale. With μ tiny relative to L the curve flattens (the legend calls this run “convex only”: its floor is positive but negligible over 400 iterations); with $\mu = 2$ it is a straight line.

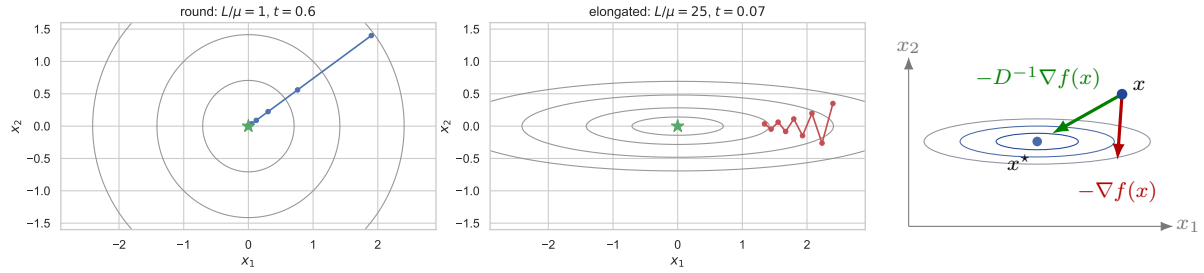


Figure 6.5: Left and middle: eight gradient-descent steps in Example 6.14. With equal curvatures the path is straight; with curvatures 1 and 25 a stable step must respect the steep x_2 direction, so x_2 overshoots back and forth while x_1 crawls. Right (sketch): on elongated contours $-\nabla f(x)$ points almost straight across the valley, whereas the scaled direction $-D^{-1}\nabla f(x)$ of Definition 6.15 points at x^* .

Both have the minimiser $x^* = 0$. Write $f(x) = \frac{1}{2}(D_1x_1^2 + D_2x_2^2)$; then $\nabla f(x) = (D_1x_1, D_2x_2)$, and a gradient step with fixed t acts on each coordinate separately: $x_i^{k+1} = (1 - tD_i)x_i^k$ for $i = 1, 2$. Coordinate i converges exactly when $|1 - tD_i| < 1$, that is, $0 < t < 2/D_i$, and the step must satisfy this for the *largest* curvature, $L = \max_i D_i$. For f_{round} ($L = \mu = 1$) the step $t = 0.6$ multiplies both coordinates by 0.4, so the iterates move along a straight line towards x^* (left panel of Figure 6.5); with $t = 1/L = 1$ a single step lands on x^* . For $f_{\text{elongated}}$ ($L = 25$, $\mu = 1$) take $x^0 = (2.4, 0.35)$ and $t = 0.07$, which lies between $1/L = 0.04$ and $2/L = 0.08$. The factors are $1 - 0.07 = 0.93$ for x_1 and $1 - 1.75 = -0.75$ for x_2 : the steep coordinate overshoots and changes sign at every step $(0.35, -0.26, 0.20, -0.15, \dots)$ while the flat coordinate crawls $(2.4, 2.23, 2.08, \dots)$, reaching 1.34 after eight steps). This is the **zig-zag** of the middle panel. The objective still decreases, from 4.41 to 0.92 in eight steps, because $t < 2/L$. With the safe step $t = 1/L = 0.04$ the step coordinate is solved at once but the flat one has the factor $1 - \mu/L = 0.96$: reaching $f \leq 10^{-6}$ takes 183 iterations, against one for the round function.

The lesson is general. With $t = 1/L$ every coordinate factor satisfies $0 \leq 1 - D_i/L \leq 1 - \mu/L$, so the slowest coordinate is multiplied by $1 - \mu/L$ at every step and its share of the gap by $(1 - \mu/L)^2$, inside the factor $1 - \mu/L$ that Theorem 6.13 guarantees: the step is dictated by the steepest direction and the speed by the flattest. We call L/μ the **curvature balance**; Week 10 names it the condition number and reduces its effect.

6.4.6 Scaled directions and a first look at Newton’s method

If uneven curvature is the problem, one remedy is to rescale the direction by the curvature.

Definition 6.15 (Scaled-direction iteration and Newton’s method). Let $D \in \mathbb{R}^{d \times d}$ be a symmetric positive definite *scaling matrix*. The **scaled-direction iteration** is

$$x^{k+1} = x^k - t_k D^{-1} \nabla f(x^k).$$

Newton’s method uses the exact local curvature, $D = \nabla^2 f(x^k)$ (assumed positive definite), normally with $t_k = 1$: $x^{k+1} = x^k - t_k [\nabla^2 f(x^k)]^{-1} \nabla f(x^k)$.

The scaled direction $d_{\text{scaled}} = -D^{-1}\nabla f(x^k)$ is a descent direction, since $\nabla f(x^k)^\top d_{\text{scaled}} = -d_{\text{scaled}}^\top D d_{\text{scaled}} < 0$ whenever $\nabla f(x^k) \neq 0$. In the language of [Section 6.2.3](#), the step $x^k + t_k d_{\text{scaled}}$ minimises the linear model with the movement penalty $\frac{1}{2t_k}(z - x^k)^\top D(z - x^k)$: length is measured in a different norm, so the best local move changes. With $t_k = 1$ the Newton step minimises the local *quadratic* Taylor model $f(x^k) + \nabla f(x^k)^\top(z - x^k) + \frac{1}{2}(z - x^k)^\top \nabla^2 f(x^k)(z - x^k)$, whose stationarity condition $\nabla f(x^k) + \nabla^2 f(x^k)(z - x^k) = 0$ gives $z = x^k - [\nabla^2 f(x^k)]^{-1}\nabla f(x^k)$. For $f_{\text{elongated}}$, $D = \nabla^2 f = \text{diag}(1, 25)$ gives $d_{\text{scaled}} = (-x_1, -x_2)^\top$: the scaled direction points straight at the minimiser, whereas $-\nabla f(x) = (-x_1, -25x_2)^\top$ points almost straight down the steep wall (right panel of [Figure 6.5](#)), and one step with $t = 1$ lands on x^* .

The price is paid per iteration: a Newton step solves a linear system with d equations and d unknowns, of the order of d^3 operations for dense data, far more than the matrix–vector work of a gradient step. [Week 13](#) pays this price deliberately, inside the barrier method.

6.5 Applications and verdict

6.5.1 Least squares

Quadratics with $Q = A^\top A$ or $Q = A^\top A + \mu I$ are the laboratory of algorithm design, with exact gradients, a constant Hessian and clear contour geometry: many later methods are first understood by asking what they do on a quadratic. Least squares is the most important member.

Model 6.16 (Least squares). Given a data matrix $A \in \mathbb{R}^{N \times d}$, whose i th row $a_i^\top$ holds the features of observation i , and a vector of responses $b \in \mathbb{R}^N$,

$$\min_{x \in \mathbb{R}^d} f(x) = \frac{1}{2} \|Ax - b\|_2^2. \quad (6.12)$$

This is the regression model of [Model 1.3](#) in matrix form: the decision variable x ([Week 1](#)'s β) is the vector of coefficients, the objective is half the sum of squared residuals $r(x) = Ax - b$, and there are no constraints; it is the canonical smooth convex model of analytics (fitting, forecasting, calibration, regression). Neither the factor $\frac{1}{2}$, absent in [Section 5.1](#), nor the $1/N$ of [Model 1.3](#) changes the minimiser; the $\frac{1}{2}$ removes a factor 2 from the gradient. Writing $f(x) = \frac{1}{2}r(x)^\top r(x)$, the chain rule ([Proposition 1.17](#)) gives

$$\nabla f(x) = A^\top(Ax - b), \quad \nabla^2 f(x) = A^\top A \succeq 0,$$

so the model is convex for any data ([Example 3.13](#)) and is exactly the quadratic (6.5) with $Q = A^\top A$, $q = -A^\top b$ and constant $\frac{1}{2}b^\top b$ (the letter r now denotes the residual). A gradient evaluation is a residual, one multiplication by A and one by $A^\top$, about Nd operations each; no $d \times d$ matrix is formed. For the step size we need a curvature cap. By the Cauchy–Schwarz inequality, applied row by row,

$$u^\top A^\top A u = \|Au\|_2^2 = \sum_{i=1}^N (a_i^\top u)^2 \leq \left(\sum_{i=1}^N \|a_i\|_2^2 \right) \|u\|_2^2 = \|A\|_F^2 \|u\|_2^2,$$

where $\|A\|_F^2$, the squared *Frobenius norm*, is the sum of the squared entries of A . Hence $L \leq \|A\|_F^2$: an overestimate of the true cap, so the step $t = 1/\|A\|_F^2$ is safe, merely conservative. [Theorem 6.11](#) then applies, and so does [Theorem 6.13](#) when the columns of A are linearly independent, because then $\|Au\|_2^2 \geq \mu \|u\|_2^2$ for some $\mu > 0$. The solution x^* holds the fitted coefficients, and $\nabla f(x^*) = 0$ is the system of normal equations $A^\top A x^* = A^\top b$.

Least squares can also be solved directly from the normal equations ([Proposition 1.13](#)) or by a solver. That route gives an accurate reference answer and suits moderate dimension, dense algebra and a one-off solve, but can feel like a black box. Gradient descent suits large or sparse data, repeated solves and learning or streaming settings; its risk is a poorly chosen step size or stopping rule.

6.5.2 Logistic loss

Example 6.17 (Logistic loss). With feature vectors $a_i \in \mathbb{R}^d$ and class labels $b_i \in \{-1, 1\}$, the logistic loss of a weight vector $w \in \mathbb{R}^d$ is

$$f(w) = \sum_{i=1}^N \log(1 + \exp(-b_i a_i^\top w)).$$

This is the objective of [Model 4.18](#), there written with θ and a ridge term and modelled through `rso.softplus`. Let $\ell(z) = \log(1 + e^{-z})$ and $\sigma(z) = 1/(1 + e^{-z})$. Then $\ell'(z) = -e^{-z}/(1 + e^{-z}) = -(1 - \sigma(z))$ and $\ell''(z) = \sigma'(z) = \sigma(z)(1 - \sigma(z))$. Term i is $\ell(z_i)$ with $z_i = b_i a_i^\top w$, an affine function of w with gradient $b_i a_i$, so with $\sigma_i = 1/(1 + \exp(-b_i a_i^\top w))$ the chain rule and $b_i^2 = 1$ give

$$\nabla f(w) = - \sum_{i=1}^N (1 - \sigma_i) b_i a_i, \quad \nabla^2 f(w) = \sum_{i=1}^N \sigma_i (1 - \sigma_i) a_i a_i^\top.$$

The gradient is a sum of observation-level contributions weighted by $1 - \sigma_i$: an observation that the model already classifies confidently *and correctly* ($b_i a_i^\top w \gg 0$, so $\sigma_i \approx 1$) pulls hardly at all, whereas a confidently misclassified one pulls with almost full weight. The Hessian is a sum of matrices $a_i a_i^\top$ with nonnegative weights, so $u^\top \nabla^2 f(w) u = \sum_i \sigma_i (1 - \sigma_i) (a_i^\top u)^2 \geq 0$ and f is convex; since $\sigma_i (1 - \sigma_i) \leq \frac{1}{4}$, the least-squares argument gives $L \leq \frac{1}{4} \|A\|_F^2$. Here $\nabla f(w) = 0$ has no closed-form solution: an iterative method is needed, and [Algorithm 6.1](#) applies (a minimiser exists once the ridge term of [Model 4.18](#) is added; on perfectly separable data the plain loss has none).

[Week 4](#) modelled this objective for a solver; it also fits first-order methods. Its gradient and curvature are sums over the data, the structure that the stochastic methods of [Week 11](#) exploit.

6.5.3 Workflow: a solver benchmark plus a NumPy implementation

The recommended workflow uses both routes of [Section 6.1](#): model $\rightarrow$ gradient and Hessian $\rightarrow$ step rule $\rightarrow$ update $\rightarrow$ stop and verify. [Listing 6.1](#) implements [Algorithm 6.1](#) for [Model 6.16](#) with a fixed step on synthetic data, and then solves the same instance with RSOME and a solver.

Listing 6.1: Fixed-step gradient descent for least squares, checked against the normal equations and a solver.

```
import numpy as np
def gd_least_squares(A, b, x0, t, maxit=500, tol=1e-6):
    x = x0.copy()
    hist_f, hist_g = [], []
    for k in range(maxit):
        r = A @ x - b                # residual r(x) = Ax - b
        f = 0.5 * np.dot(r, r)        # objective value
        g = A.T @ r                  # gradient A^T (Ax - b)
        hist_f.append(f)
        hist_g.append(np.linalg.norm(g))
        if np.linalg.norm(g) <= tol:  # stopping rule
            break
        x = x - t * g                # gradient step
    return x, np.array(hist_f), np.array(hist_g)
rng = np.random.default_rng(1)
N, d = 200, 5
A = rng.normal(size=(N, d))
b = A @ np.array([1.0, -2.0, 0.5, 0.0, 3.0]) + 0.1 * rng.normal(size=N)
L_safe = np.sum(A**2)                # ||A||_F^2 >= L
x, hf, hg = gd_least_squares(A, b, np.zeros(d), t=1.0 / L_safe)
x_direct = np.linalg.solve(A.T @ A, A.T @ b)  # normal equations
print(len(hf) - 1, hf[-1], hg[-1], np.abs(x - x_direct).max())
```

```

import rsome as rso                                # benchmark lane: RSOME + solver
from rsome import ro, grb_solver as grb
m = ro.Model(); xv = m.dvar(d); u = m.dvar()
m.min(0.5 * u)                                     # affine objective (1/2) u
m.st(rso.sumsqr(A @ xv - b) <= u)                 # epigraph: ||Ax - b||^2 <= u
m.solve(grb)
print(m.get(), np.linalg.norm(A.T @ (A @ xv.get() - b)))

```

The function records both histories and tests the gradient norm before each update; the script uses the safe step $1/\|A\|_F^2$. Its first `print` reports 119 iterations, a final objective 1.0335, a gradient norm 9.0×10^{-7} and a largest coordinate difference of 5.8×10^{-9} from the normal-equations solution; the objective decreases at every iteration, as [Proposition 6.9](#) promises. The run is slow because $\|A\|_F^2 \approx 976$ overestimates fourfold the true cap $L \approx 239$ (the largest value of $\|Au\|_2^2$ over unit vectors u , computed as `np.linalg.norm(A, 2)**2`): with $t = 1/L$ the same tolerance is reached in 21 iterations, and with [Algorithm 6.2](#), which needs no constant, in 18. A hand-derived gradient should first be checked against central differences $(f(x + he_i) - f(x - he_i))/(2h)$, with e_i the i th coordinate vector; at $x^0 = 0$ and $h = 10^{-5}$ the relative error is about 10^{-10} .

The last lines are the benchmark lane: the same instance as an RSOME model with the affine objective $\frac{1}{2}u$ and the epigraph constraint $\|Ax - b\|_2^2 \leq u$ ([Section 4.5](#)). In our run Gurobi returns the objective 1.03345486, equal to that of gradient descent to eight significant digits, at a point with gradient norm about 4×10^{-4} . Each comparison answers a different question. Equal objective values say that the hand-written loop reached the optimum a trusted solver reports. The gradient norm is an optimality residual that needs no solver, and the loop meets it more tightly here than the solver's default tolerances do. Iteration counts and running times measure cost, where the two routes separate as N and d grow.

6.5.4 The same objects in machine-learning vocabulary

Machine-learning software uses the mathematics of this week under other names. The step size t is the **learning rate**. The gradient that we derive by hand is supplied by `autodiff`, which is the chain rule of [Proposition 1.17](#) run by the computer. Minimising a loss f is **training** a model; the stopping rules of [Table 6.1](#) appear as the arguments `tol` and `max_iter`; and the update loop of [Algorithm 6.1](#) is the optimiser loop inside every library. Deriving one gradient by hand is how one knows what the library computes, and how to debug it when the loss will not fall. [Week 11](#) adds the rest of the vocabulary (mini-batch, epoch, learning-rate schedule, Adam) when sampling enters.

6.5.5 What the theory anchors, what practice tunes, and a verdict

The safe step $1/L$ needs L , which beyond this laboratory is unknown, expensive or far too cautious. What survives is that backtracking *measures* the local cap and that the rates anchor the tuning; theory does not hand us t . Nor is the safe step the fastest.

Example 6.18 (An unsafe schedule that wins). Take $f_{\text{elongated}}$ ($L = 25$) and $x^0 = (2.4, 0.5)$, so $f(x^0) = 6.005$. Twenty-five safe steps $t = 1/L$ zero x_2 at once and multiply x_1 by 0.96 each time, ending at $f = 0.374$. Alternatively take 24 steps with $t = 2.1/L = 0.084$, then one step with $1/L$. The long step multiplies x_1 by 0.916 and x_2 by -1.1 , so each of the 24 steps *increases* f , to a peak of 303, about 50 times the start. The last step zeroes x_2 , leaving $x_1 = 2.4 \cdot 0.916^{24} \cdot 0.96$ and $f = 0.0393$, 9.5 times lower than the safe schedule ([Figure 6.6](#)). A $1/L$ step never goes uphill, but that does not make it fastest: the long steps progress faster in the flat direction, and one safe step repairs the damage in the steep one. Step-size schedules return in [Week 11](#); [Week 10](#) keeps the step at $1/L$ (or backtracks) and puts a schedule on a momentum coefficient instead.

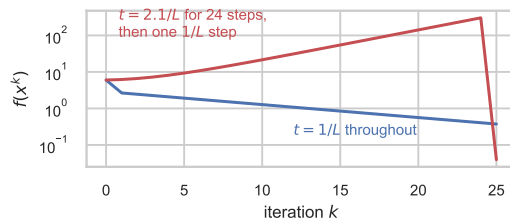


Figure 6.6: Objective $f(x^k)$ in Example 6.18, logarithmic scale: 25 safe steps $t = 1/L$ against 24 uphill steps $t = 2.1/L$ followed by one safe step.

Note on the lecture slides

Frame 6.41 places this experiment “on 6.30’s elongated quadratic” without a starting point. Its numbers (24 uphill steps, peak $\approx 50\times$, $9.5\times$ lower) belong to $x^0 = (2.4, 0.5)$; from the start $(2.4, 0.35)$ of Example 6.14 the first long step goes downhill and the peak is about 34 times the start, while the factor 9.5 is unchanged.

What gradient descent does and why it works: it repeatedly minimises a local model that, for $t \leq 1/L$, lies above f ; each iteration costs one gradient and a few vector operations, so the method scales to large problems and exploits sparsity and structure. *When it applies:* on smooth convex problems that have a minimiser it converges from any starting point, at the rate $O(1/k)$ in general and linearly under strong convexity, where the gradient norm even certifies the objective gap. *Its limitations:* it is slow when the curvature is uneven (L/μ large); it needs smoothness, since a kink breaks the descent guarantee (Week 7); it cannot handle constraints or nonsmooth penalties directly (Weeks 8 and 9); and a poor step size wastes the theory. It is the baseline against which every later method is measured.

6.6 Summary and looking ahead

Gradient descent is the first general optimisation engine of the course. A local model, the tangent prediction plus a movement penalty, produces the update $x^{k+1} = x^k - t_k \nabla f(x^k)$; the step size is the trust placed in the linear prediction, and the gradient norm, the residual inherited from Week 5, decides when to stop. Taylor’s formula explains why two quadratic models matter: a curvature cap L places f below a quadratic, which yields the safe step $t \leq 1/L$, backtracking and the $O(1/k)$ rate; a curvature floor μ places f above a quadratic, which yields a unique minimiser, the linear rate $(1 - \mu/L)^k$ and the certificate $f(x) - f^* \leq \|\nabla f(x)\|_2^2 / (2\mu)$. The ratio L/μ governs the speed: uneven curvature produces zig-zagging, which scaled directions, with Newton’s method as the extreme case, remove at a much higher cost per iteration.

The next four weeks keep the template and change only the direction, the feasible move or the local model. Week 7 removes smoothness: at a kink a subgradient replaces the gradient in the same local model, which gives the subgradient method. The descent guarantee is lost, the step sizes must shrink, and the rate slows from $O(1/k)$ to $O(1/\sqrt{k})$ (and to $O(1/k)$ rather than linear under strong convexity). Week 8 restricts the local model (6.3) to a simple feasible set and obtains the projected-gradient step; Week 9 adds a nonsmooth regulariser to it and obtains the proximal-gradient step; Week 10 keeps the same gradients and reduces the iteration count by momentum.

6.7 Exercises

Exercise 6.1 (Taylor on $\mathbb{R}$). For $f(x) = x^2 - 4x + 1$, write the Taylor expansion of $f(y)$ around x with the integral remainder (Proposition 6.2) and interpret the second-derivative term.

Exercise 6.2 (Descent directions and a Newton preview). For $f(x_1, x_2) = x_1^2 + 2x_2^2 - 2x_1 - 4x_2$, compute $\nabla f(0, 0)$ and $\nabla^2 f(x)$. Compare the directions $d_1 = (1, 0)$, $d_2 = -\nabla f(0, 0)$ and

$d_N = -[\nabla^2 f(0,0)]^{-1} \nabla f(0,0)$: which are descent directions at $(0,0)$? Compute one step with d_2 and $t = \frac{1}{4}$, and one step with d_N and $t = 1$.

Exercise 6.3 (Exact line search). For $f(x) = \frac{1}{2}x^\top Qx + q^\top x$ with $Q = \text{diag}(4, 1)$ and $q = (-8, -2)$, derive the exact line-search step from $x^0 = (0, 0)$ along the negative gradient direction.

Exercise 6.4 (Safe step from smoothness). Starting from the multivariate Taylor formula with integral remainder, prove $f(y) \leq f(x) + \nabla f(x)^\top (y - x) + \frac{L}{2} \|y - x\|_2^2$ under the upper Hessian bound, and then show that every $0 < t \leq 1/L$ gives objective decrease.

Exercise 6.5 (Least-squares gradient and Hessian). For $f(x) = \frac{1}{2} \|Ax - b\|_2^2$, derive $\nabla f(x) = A^\top (Ax - b)$ and $\nabla^2 f(x) = A^\top A$, and compare with [Proposition 1.17](#) and [Example 3.13](#). Then, for $A = \text{diag}(1, 2)$, read off the curvature cap L and floor μ of [Definitions 6.8](#) and [6.12](#) as the largest and smallest values of $u^\top A^\top A u$ over unit vectors u , and state the safe step that [Proposition 6.9](#) allows.

Exercise 6.6 (Scaled directions on an elongated quadratic). For $f(x) = \frac{1}{2}(x_1^2 + 25x_2^2)$ and $x = (3, 1)$, compare $d_g = -\nabla f(x)$, $d_s = -D^{-1}\nabla f(x)$ with $D = \text{diag}(1, 25)$, and $d_N = -[\nabla^2 f(x)]^{-1}\nabla f(x)$. Which directions coincide? Which points directly to the minimiser? What does this say about curvature-aware scaling?

Exercise 6.7 (Barzilai–Borwein step size). For gradient descent on least squares define $s^{k-1} = x^k - x^{k-1}$, $y^{k-1} = g^k - g^{k-1}$ and $t_k^{\text{BB}} = (s^{k-1})^\top s^{k-1} / (s^{k-1})^\top y^{k-1}$. In [Listing 6.1](#), implement this rule after the first iteration (store the previous iterate and gradient) and compare it numerically with a conservative fixed step and with backtracking. Plot objective value and gradient norm by iteration.

Exercise 6.8 (Solver benchmark and interpretation). Solve the same least-squares instance with your NumPy implementation and with a solver-backed model. Compare the final objective value, the final gradient norm, and runtime or iteration count. What does each comparison tell you?

Week 7 Nonsmooth Convex Optimisation and the Subgradient Method

What replaces the gradient at a kink, and what does that cost us? [Week 6](#) built gradient descent on two pillars: the gradient itself, a single vector that summarises the local slope, and the quadratic *upper* model supplied by L -smoothness ([Proposition 6.9](#)), which guarantees that a safe step decreases the objective. Many convex objectives that matter in analytics have neither: absolute deviations, maxima of scenarios, ℓ_1 norms and hinge-type losses all have kinks, points where the graph has a corner and no gradient exists. This week keeps the unconstrained problem

$$\min_{x \in \mathbb{R}^d} f(x), \quad (7.1)$$

with $f : \mathbb{R}^d \rightarrow \mathbb{R}$ convex and finite everywhere, but drops differentiability. We assume throughout that a minimiser x^* exists and write $f^* = f(x^*)$.

Much of the earlier material survives. Convexity never needed smoothness: [Week 3](#) gave a first look at *subgradients* ([Definition 3.17](#)), [Week 5](#) stated the certificate $0 \in \partial f(x^*)$ ([Theorem 5.2](#)), and the local model from which [Week 6](#) derived the gradient step ([Equation \(6.3\)](#)) needs only *a* slope, not *the* slope. What changes is equally definite: at a kink the valid slopes form a set, a step along the negative of one of them need not decrease the objective, and the convergence rate drops from $O(1/k)$ to $O(1/\sqrt{k})$ (in the $O(\cdot)$ notation of [Definition 6.10](#)).

[Section 7.1](#) explains where kinks come from and what fails there, and [Section 7.2](#) defines subgradients and gives the rules for computing them. [Section 7.3](#) proves the optimality condition, solves two problems with it by hand, and derives the subgradient method from last week's local model. [Section 7.4](#) analyses the method, which is not a descent method: its steps need not decrease f , unlike gradient steps with a safe step size, so the analysis tracks the distance to x^* , and the step-size rules are different answers to one trade-off. [Section 7.5](#) turns to stopping, to least-absolute-deviations regression, and to a verdict: the subgradient method is the simplest general baseline, and [Weeks 8 to 10](#) improve on it by exploiting structure that it ignores.

7.1 Why nonsmooth problems need their own treatment

7.1.1 Where kinks come from

Nonsmoothness is usually a modelling choice, not a pathology. It appears whenever we care about robustness, worst-case behaviour, sparsity or piecewise-linear decision rules. *Absolute deviations* $|r|$ penalise an error in proportion to its size rather than its square; they underlie the median and robust regression. A *maximum of scenarios*, $\max_i (a_i^\top x + b_i)$, is the worst-case cost of a plan; minimising it is the minimax problem of [Model 3.25](#). The ℓ_1 *norm* $\|x\|_1 = \sum_i |x_i|$ charges each coordinate in proportion to its size, at a rate that does not fade near zero, which is why it models proportional transaction costs and encourages sparse solutions. *Hinge-like losses* such as $\max\{0, 1 - u\}$ switch a penalty on only when a margin is violated, as in the support vector machine of [Model 4.17](#). Each is convex by [Proposition 3.19](#), and each has its kink exactly where the modelling interest lies: at a zero error, a tie between scenarios, a zero coordinate, the margin. Four examples recur through the week: the scalar function $|x|$; the median problem $\min_x \sum_i |x - c_i|$; the max-of-affine function $\max\{x_1, x_2, -x_1 - x_2\}$ on $\mathbb{R}^2$; and least-absolute-deviations (LAD) regression, $\min_\beta \|A\beta - b\|_1$, the week's analytics model and coding example.

7.1.2 What fails at a kink

For a differentiable convex function a single gradient summarises the local slope, and the tangent plane it defines lies below the graph everywhere ([Theorem 3.8](#)). At a kink this summary breaks

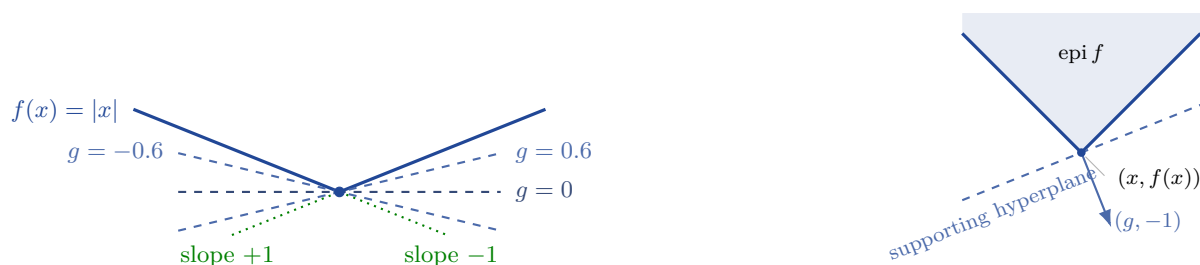


Figure 7.1: Left: at the kink of $|x|$ the one-sided tangents (dotted, slopes -1 and $+1$) disagree, but every line through the kink with slope $g \in [-1, 1]$ (three are dashed) stays below the graph. Right: in the epigraph picture, the line of slope $g = 0.4$ is a supporting hyperplane of $\text{epi } f$ at $(x, f(x))$ with normal vector $(g, -1)$.

down. Take $f(x) = |x|$ at $x = 0$. Approaching from the left the slope is -1 , approaching from the right it is $+1$; the two one-sided tangent lines disagree, so $f'(0)$ does not exist (the dotted continuations of the arms in Figure 7.1, left).

Convexity nevertheless leaves something to work with. Every line through the origin whose slope lies between -1 and $+1$ stays below the graph of $|x|$ and touches it at the kink. Such *supporting affine lower bounds* are the right replacement for the tangent: the property of the gradient that Weeks 3 and 5 actually used was never “limit of difference quotients” but “global affine under-estimator that touches at x ”.

7.2 Subgradients and the subdifferential

7.2.1 Supporting lines and supporting hyperplanes

On the real line the idea is one inequality. A number g is a subgradient of f at x if

$$f(y) \geq f(x) + g(y - x) \quad \text{for all } y \in \mathbb{R},$$

that is, if the line through $(x, f(x))$ with slope g supports the graph globally from below. For $f(x) = |x|$ at $x = 0$ the condition reads $|y| \geq gy$ for all y . Taking $y = \pm 1$ shows that $|g| \leq 1$ is necessary, and it is sufficient because $gy \leq |g||y| \leq |y|$. Every slope in $[-1, 1]$ therefore works, and no other: subgradients form a *set*, whose extreme members ± 1 lie along the two arms of the graph while the interior slopes fill the fan between them.

In $\mathbb{R}^d$ the natural home for this picture is the epigraph $\text{epi } f = \{(y, t) \in \mathbb{R}^d \times \mathbb{R} : t \geq f(y)\}$ of Definition 3.1, the set of points on or above the graph; here t is the extra “height” coordinate, not a step size. A hyperplane (Definition 2.10) **supports** a convex set at a point of the set if it passes through that point and the whole set lies in one of its two closed halfspaces. A vector $g \in \mathbb{R}^d$ satisfies $f(y) \geq f(x) + g^\top(y - x)$ for all y exactly when the hyperplane through $(x, f(x))$ with normal vector $(g, -1)$ supports $\text{epi } f$. Indeed, the statement that $\text{epi } f$ lies on one side of that hyperplane is

$$\begin{pmatrix} g \\ -1 \end{pmatrix}^\top \begin{pmatrix} y - x \\ t - f(x) \end{pmatrix} \leq 0 \quad \forall (y, t) \in \text{epi } f \quad \iff \quad t \geq f(x) + g^\top(y - x) \quad \forall (y, t) \in \text{epi } f.$$

Taking $t = f(y)$ gives the inequality for f ; conversely, if the inequality holds for f then it holds for every $t \geq f(y)$. A subgradient *is* a (non-vertical) supporting hyperplane of the epigraph (Figure 7.1, right): lifting f to $\text{epi } f$ turns slopes into geometry.

7.2.2 Definition and first properties

Definition 7.1 (Subgradient and subdifferential). Let $f : \mathbb{R}^d \rightarrow \mathbb{R}$ be convex and let $x \in \mathbb{R}^d$. A vector $g \in \mathbb{R}^d$ is a *subgradient* of f at x if

$$f(y) \geq f(x) + g^\top(y - x) \quad \text{for all } y \in \mathbb{R}^d. \quad (7.2)$$

The *subdifferential* $\partial f(x)$ is the set of all subgradients of f at x . We call (7.2) the **subgradient inequality**.

This is the definition previewed in Definition 3.17; three facts make it usable. First, $\partial f(x)$ is a closed convex set, being the intersection over all y of the closed halfspaces $\{g : g^\top(y - x) \leq f(y) - f(x)\}$ in the variable g . Second, for the finite-valued convex functions of this unit it is never empty: a convex epigraph has a non-vertical supporting hyperplane at every point of the graph, a fact we use without proof. Third, where f is differentiable, $\partial f(x) = \{\nabla f(x)\}$: the gradient is a subgradient by Theorem 3.8, and Week 3 showed that no other vector qualifies.

Key idea

A subgradient is a global supporting slope. At a differentiable point it is the gradient; at a kink it is any member of a closed convex set, and one valid member is all that an algorithm needs.

Example 7.2 (Absolute value and positive part). The two basic scalar kinks, the absolute value and the positive part $\max\{0, x\}$, have the subdifferentials

$$\partial|x| = \begin{cases} \{1\}, & x > 0, \\ [-1, 1], & x = 0, \\ \{-1\}, & x < 0, \end{cases} \quad \partial \max\{0, x\} = \begin{cases} \{0\}, & x < 0, \\ [0, 1], & x = 0, \\ \{1\}, & x > 0. \end{cases}$$

Away from 0 both functions are differentiable and the subdifferential contains only the derivative. At 0 the absolute value was treated above; for the positive part, $\max\{0, y\} \geq g y$ for all y forces $g \leq 1$ (take $y > 0$) and $g \geq 0$ (take $y < 0$), and every $g \in [0, 1]$ works. In each case the subdifferential at the kink is the interval between the left and the right slope.

7.2.3 Maximum of affine functions

The most useful nonsmooth example, geometrically, is a maximum of finitely many affine pieces.

Proposition 7.3 (Subdifferential of a max of affine functions). Let $f(x) = \max_{i=1, \dots, m} (a_i^\top x + b_i)$ with $a_i \in \mathbb{R}^d$, $b_i \in \mathbb{R}$, and let $I(x) = \{i : a_i^\top x + b_i = f(x)\}$ be the set of **active** pieces at x . Then

$$\partial f(x) = \text{conv}\{a_i : i \in I(x)\}.$$

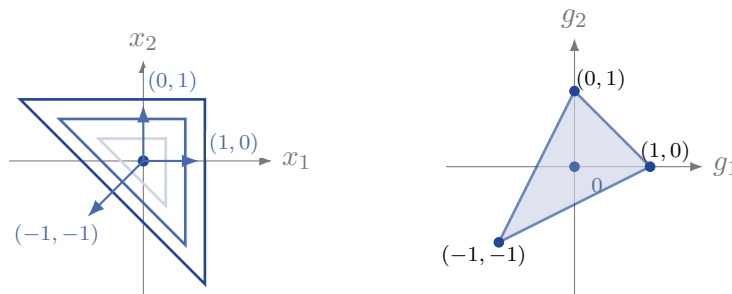


Figure 7.2: The running instance $f(x) = \max\{x_1, x_2, -x_1 - x_2\}$. Left: three level sets and the slopes of the three pieces, all active at the origin. Right: the subdifferential $\partial f(0) = \text{conv}\{(1, 0), (0, 1), (-1, -1)\}$ in the space of slopes; it contains the zero vector, which Example 7.9 reads as a certificate that $x = 0$ is optimal.

The inclusion “ $\supseteq$ ” takes one line: if i is active at x , then $f(y) \geq a_i^\top y + b_i = f(x) + a_i^\top (y - x)$ for every y , so $a_i \in \partial f(x)$, and the convex set $\partial f(x)$ then contains the convex hull (Definition 2.4) of the active slopes. The reverse inclusion says that inactive pieces, which lie strictly below f near x , contribute nothing; we state it without proof. Where a single piece is active, f is locally affine and $\partial f(x) = \{a_i\}$; where several pieces tie, the subdifferential is the polytope spanned by their slopes. Our running instance is $f(x) = \max\{x_1, x_2, -x_1 - x_2\}$ on $\mathbb{R}^2$, with slopes $(1, 0)$, $(0, 1)$ and $(-1, -1)$. For $c > 0$ its level set $\{x : f(x) = c\}$ is the boundary of the triangle with vertices (c, c) , $(c, -2c)$ and $(-2c, c)$, so the level sets are nested triangles shrinking to the origin (Figure 7.2, left). At $x = 0$ all three pieces are active, and $\partial f(0)$ is the triangle $\text{conv}\{(1, 0), (0, 1), (-1, -1)\}$.

7.2.4 Three calculus rules

Week 3 recognised convex functions by assembling them from simple pieces with closure rules (Proposition 3.19). Each rule has a companion that assembles a subgradient from subgradients of the pieces.

Proposition 7.4 (Subgradient calculus). *Let f , h and $f_1, \dots, f_m$ be convex functions.*

- (a) *Sum rule. $\partial f(x) + \partial h(x) \subseteq \partial(f + h)(x)$: if $g_1 \in \partial f(x)$ and $g_2 \in \partial h(x)$, then $g_1 + g_2$ is a subgradient of $f + h$ at x .*
- (b) *Affine composition. If $h(x) = f(Ax + b)$ with $A \in \mathbb{R}^{n \times d}$, $b \in \mathbb{R}^n$ and $f : \mathbb{R}^n \rightarrow \mathbb{R}$, then $A^\top \partial f(Ax + b) \subseteq \partial h(x)$: if $g \in \partial f(Ax + b)$, then $A^\top g \in \partial h(x)$.*
- (c) *Active max rule. If $f = \max_i f_i$ and $I(x) = \{i : f_i(x) = f(x)\}$, then $\text{conv}\{\partial f_i(x) : i \in I(x)\} \subseteq \partial f(x)$: convex combinations of subgradients of active pieces are subgradients of f .*

For the finite-valued convex functions of this unit all three inclusions hold with equality.

Proof. Each inclusion is one line from (7.2). (a) Add $f(y) \geq f(x) + g_1^\top (y - x)$ and $h(y) \geq h(x) + g_2^\top (y - x)$. (b) For every y , $h(y) = f(Ay + b) \geq f(Ax + b) + g^\top (A(y - x)) = h(x) + (A^\top g)^\top (y - x)$. (c) If $i \in I(x)$ and $g_i \in \partial f_i(x)$, then $f(y) \geq f_i(y) \geq f_i(x) + g_i^\top (y - x) = f(x) + g_i^\top (y - x)$, so $g_i \in \partial f(x)$; convex combinations stay in $\partial f(x)$ because that set is convex. We do not prove the equalities. $\square$

Rule (b) is the nonsmooth descendant of the chain rule $\nabla_x f(Ax + b) = A^\top \nabla f(Ax + b)$ of Proposition 1.17; for instance, the hinge loss $\max\{0, 1 - u\}$ is the positive part composed with $u \mapsto 1 - u$, so its subdifferential at the margin $u = 1$ is $-[0, 1] = [-1, 0]$. Proposition 7.3 is rule (c) applied to affine pieces. For more general (extended-valued) convex functions the reverse inclusions can fail, which is why the rules are stated as inclusions; for algorithms this is harmless, since the recipes only ever produce *valid* subgradients.

7.2.5 The ℓ_1 norm and LAD regression

Example 7.5 (Subdifferential of the ℓ_1 norm). For $\|x\|_1 = \sum_{i=1}^d |x_i|$,

$$\partial \|x\|_1 = \{g \in \mathbb{R}^d : g_i = \text{sign}(x_i) \text{ if } x_i \neq 0, \quad g_i \in [-1, 1] \text{ if } x_i = 0\}.$$

Each coordinate behaves like a copy of $|x|$, because the function is a sum of terms each depending on one coordinate: $\|y\|_1 - \|x\|_1 - g^\top (y - x) = \sum_i (|y_i| - |x_i| - g_i(y_i - x_i))$. If $g_i \in \partial |x_i|$ for every i , each bracket is nonnegative and g is a subgradient (the sum rule); conversely, choosing y equal to x except in coordinate i leaves only the i -th bracket, which must then be nonnegative for all y_i , so $g_i \in \partial |x_i|$. For instance, at $x = (3, 0, -2)$ the subgradients are the vectors $(1, g_2, -1)$ with $g_2 \in [-1, 1]$.

This is the formula that [Week 5](#) quoted for ℓ_1 -regularised least squares, and the prototype for sparsity penalties, transaction-cost terms and robust regression residuals; [Week 9](#) will handle such terms far more effectively with a proximal step than the raw subgradient update can. With the affine-composition rule it gives the subgradient of the week's main model.

Model 7.6 (Least-absolute-deviations regression). Given a data matrix $A \in \mathbb{R}^{N \times d}$ with rows $a_i^\top$ and responses $b \in \mathbb{R}^N$,

$$\min_{\beta \in \mathbb{R}^d} f(\beta) = \|A\beta - b\|_1 = \sum_{i=1}^N |a_i^\top \beta - b_i|. \quad (7.3)$$

The decision variables are the regression coefficients β (the least-squares model of [Model 6.16](#) called them x ; for regression models we follow the convention β , which no longer denotes the backtracking factor of [Week 6](#)); there are no constraints. The objective adds up the absolute residuals of the N data points, penalising residual size linearly where least squares ([Model 6.16](#)) penalises it quadratically. It is convex (a norm composed with an affine map), piecewise linear, and kinked wherever a residual is zero, that is, wherever the fitted model passes exactly through a data point. The model assumes a linear relation between features and response and treats a residual of size 2 as exactly twice as bad as one of size 1, whatever its sign. A minimiser always exists but need not be unique, because f is convex without being strictly convex. Since $f(\beta) = h(A\beta - b)$ with $h = \|\cdot\|_1$, rule (b) and [Example 7.5](#) give

$$g = A^\top s \in \partial f(\beta), \quad s_i = \begin{cases} 1, & (A\beta - b)_i > 0, \\ \text{any number in } [-1, 1], & (A\beta - b)_i = 0, \\ -1, & (A\beta - b)_i < 0. \end{cases} \quad (7.4)$$

Each residual contributes its sign, exactly like the one-dimensional absolute value, and the choice $s = \text{sign}(A\beta - b)$ (with $\text{sign}(0) = 0$) is always valid. Computing g costs two matrix–vector products, $O(Nd)$ operations, the same as a least-squares gradient. How the model is solved, and why one would prefer it to least squares, is the subject of [Section 7.5](#).

7.3 Optimality and the subgradient update

7.3.1 The condition $0 \in \partial f(x^*)$

For a differentiable convex function the unique slope must vanish at a minimiser, $\nabla f(x^*) = 0$ ([Theorem 5.1](#)). When the slopes form a set, the natural counterpart is that the set contains the zero vector.

Theorem 7.7 (Optimality for convex minimisation). *Let $f : \mathbb{R}^d \rightarrow \mathbb{R}$ be convex. Then x^* is a global minimiser of f if and only if $0 \in \partial f(x^*)$.*

Proof. Put $g = 0$ in (7.2) at $x = x^*$: the inequality reads $f(y) \geq f(x^*)$ for every y . So $0 \in \partial f(x^*)$ says precisely that x^* is a global minimiser, and conversely. $\square$

The proof is a single line because the subgradient inequality is global: a horizontal supporting hyperplane certifies a minimum. When f is differentiable the condition reduces to $\nabla f(x^*) = 0$. Both conditions were stated in [Week 5](#) ([Theorems 5.1](#) and [5.2](#)); what is new is that we can now *compute* subdifferentials, so the condition can be checked, and that it becomes the stopping principle behind this week's algorithm, with one difficulty taken up in [Section 7.5](#): at a nonsmooth minimiser $\partial f(x^*)$ contains 0 but usually nonzero vectors too.

7.3.2 Two worked examples

Example 7.8 (The median from subgradient balance). Given data $c_1, \dots, c_N \in \mathbb{R}$, consider

$$\min_{x \in \mathbb{R}} f(x) = \sum_{i=1}^N |x - c_i|.$$

The term $|x - c_i|$ has slope $+1$ if $c_i < x$, slope -1 if $c_i > x$, and subdifferential $[-1, 1]$ if $c_i = x$. Let $n_<$, $n_>$ and $n_ =$ count the data points to the left of x , to the right of x , and equal to x . By the sum rule (with equality),

$$\partial f(x) = [n_< - n_> - n_ =, n_< - n_> + n_ =].$$

Each point to the left pulls the slope up by one, each point to the right pulls it down by one, and points sitting at x can pull either way. By [Theorem 7.7](#), x is optimal exactly when $|n_< - n_>| \leq n_ =$, that is, when the two pulls can balance. Since $n_< + n_> + n_ = = N$, this means $n_< \leq N/2$ and $n_> \leq N/2$: at most half of the data lie strictly on either side, which is the definition of a *median*. With sorted data $c_{(1)} \leq \dots \leq c_{(N)}$, the minimiser is $c_{((N+1)/2)}$ when N is odd, and every point of $[c_{(N/2)}, c_{(N/2+1)}]$ when N is even.

For the data $c = (1, 2, 4, 7, 9)$: at $x = 4$ we have $n_< = n_> = 2$ and $n_ = = 1$, so $\partial f(4) = [-1, 1] \ni 0$ and $x^* = 4$ with $f^* = 3 + 2 + 0 + 3 + 5 = 13$; at $x = 3$ we find $\partial f(3) = \{2 - 3\} = \{-1\}$, so f still decreases to the right ($f(3) = 14$). Replacing the observation 9 by 900 changes neither the counts nor the minimiser, whereas the mean (the minimiser of $\sum_i (x - c_i)^2$) moves from 4.6 to 182.8: absolute deviations are robust to outliers because only the *side* on which a data point lies enters the optimality condition, not its distance.

Example 7.9 (A minimax objective and the convex-hull test). Consider

$$\min_{x \in \mathbb{R}^2} f(x) = \max\{x_1, x_2, -x_1 - x_2\}.$$

At $x = 0$ the three pieces are all active, with slopes $(1, 0)$, $(0, 1)$ and $(-1, -1)$. By [Proposition 7.3](#), $\partial f(0)$ is their convex hull, and

$$0 = \frac{1}{3}(1, 0) + \frac{1}{3}(0, 1) + \frac{1}{3}(-1, -1)$$

exhibits the origin as a convex combination of the active slopes ([Figure 7.2](#), right). Hence $0 \in \partial f(0)$ and $x^* = 0$ is a global minimiser with $f^* = 0$. The same weights explain the result directly: a maximum is at least any weighted average, so $f(x) \geq \frac{1}{3}x_1 + \frac{1}{3}x_2 + \frac{1}{3}(-x_1 - x_2) = 0$ for every x . In general, a point is optimal for a max-of-affine objective exactly when the zero vector lies in the convex hull of the slopes active there.

7.3.3 From the stabilised local model to the update

[Week 6](#) derived the gradient step by minimising a local model of f around the current iterate x^k : a linear prediction plus a quadratic term that keeps the next point close to x^k , where the prediction can be trusted ([Equation \(6.3\)](#)). Nothing in that construction requires the slope to be unique. Choose *any* subgradient $g^k \in \partial f(x^k)$ and minimise

$$\min_{z \in \mathbb{R}^d} f(x^k) + g^{k\top}(z - x^k) + \frac{1}{2\alpha_k} \|z - x^k\|_2^2, \quad (7.5)$$

where $\alpha_k > 0$ is the step size. ([Week 6](#) wrote the step size as t_k ; from this week onwards the course writes α_k .) The objective is a strictly convex quadratic in z , so its minimiser is where the gradient with respect to z vanishes:

$$0 = g^k + \frac{1}{\alpha_k}(z - x^k) \implies x^{k+1} = x^k - \alpha_k g^k, \quad g^k \in \partial f(x^k). \quad (7.6)$$

Algorithm 7.1 Subgradient method

Require: convex f ; a routine returning $f(x)$ and one $g \in \partial f(x)$; starting point x^0 ; step-size rule; iteration budget K

Ensure: the best point found, x_{best} , and its value f_{best}

```

1:  $f_{\text{best}} \leftarrow +\infty$ 
2: for  $k = 0, 1, \dots, K - 1$  do
3:   evaluate  $f(x^k)$  and choose any  $g^k \in \partial f(x^k)$ 
4:   if  $f(x^k) < f_{\text{best}}$  then  $f_{\text{best}} \leftarrow f(x^k)$ ,  $x_{\text{best}} \leftarrow x^k$ 
5:   if  $g^k = 0$  then return  $x^k$ ,  $f(x^k)$   $\triangleright 0 \in \partial f(x^k)$ : optimal by Theorem 7.7
6:   choose  $\alpha_k > 0$  by the step-size rule
7:    $x^{k+1} \leftarrow x^k - \alpha_k g^k$ 
8: end for
9: return  $x_{\text{best}}$ ,  $f_{\text{best}}$ 
    
```

This is the **subgradient method** (Algorithm 7.1). The formula coincides with gradient descent (Algorithm 6.1) wherever f is differentiable, and the only problem-specific ingredient is a routine that returns one valid subgradient. The hard part is no longer deriving the update; it is understanding which guarantees remain without differentiability.

It helps to compare the two methods through the three design choices of Section 6.1, namely direction, step size and stopping rule. Gradient descent needs differentiability, its smoothness upper model guarantees descent, its rate is $O(1/k)$ on smooth convex problems, and $\|\nabla f(x^k)\|_2$ is a natural stopping residual. The subgradient method accepts convex kinks, but it has no monotone-descent guarantee (hence the best point remembered in Algorithm 7.1), achieves only $O(1/\sqrt{k})$, and is more delicate to stop, as the next subsection shows.

7.4 Behaviour and convergence of the subgradient method

7.4.1 Not a descent method

Example 7.10 (Oscillation on $|x|$). Apply (7.6) to $f(x) = |x|$ from $x^0 = 1.4$ with the fixed step $\alpha = 0.6$ and $g^k = \text{sign}(x^k)$. The iterates are

$$1.4 \rightarrow 0.8 \rightarrow 0.2 \rightarrow -0.4 \rightarrow 0.2 \rightarrow -0.4 \rightarrow \dots,$$

with objective values 1.4, 0.8, 0.2, 0.4, 0.2, 0.4, $\dots$. Every step has length 0.6 whatever the distance to the minimiser, so once the iterate is within 0.6 of the kink it overshoots, and then overshoots back. The objective *increases* on every second step and the best value ever seen is 0.2: a step size that does not shrink can oscillate for ever even on the simplest convex objective.

Example 7.11 (A cycle across active pieces). Take $f(x) = \max\{x_1, x_2, -x_1 - x_2\}$ from $x^0 = (2, 0.2)$ with $\alpha = 1$, choosing the slope of the active piece as subgradient. At x^0 and at $x^1 = (1, 0.2)$ the first piece is active, $g = (1, 0)$, and the method moves left to $x^2 = (0, 0.2)$ with $f(x^2) = 0.2$. There the second piece is active, $g = (0, 1)$, and the step leads to $x^3 = (0, -0.8)$, where $f(x^3) = 0.8$: a perfectly legal step has *increased* the objective fourfold. Now the third piece is active, $g = (-1, -1)$, and $x^4 = (1, 0.2) = x^1$. The method cycles through three points with values 1, 0.2, 0.8 (Figure 7.3, left). For piecewise-linear objectives the active piece, and with it the selected slope, can change abruptly between iterates; the result is a zig-zag or even a cycle.

Why can a legal step increase the objective? In the smooth case the descent lemma (Proposition 6.9) supplies a quadratic *upper* model, $f(z) \leq f(x) + \nabla f(x)^\top(z - x) + \frac{L}{2} \|z - x\|_2^2$, and minimising an upper model is a safe way to decrease f . A subgradient supplies only the *lower* model $f(z) \geq f(x) + g^\top(z - x)$, and the model (7.5), this lower bound plus a movement penalty,

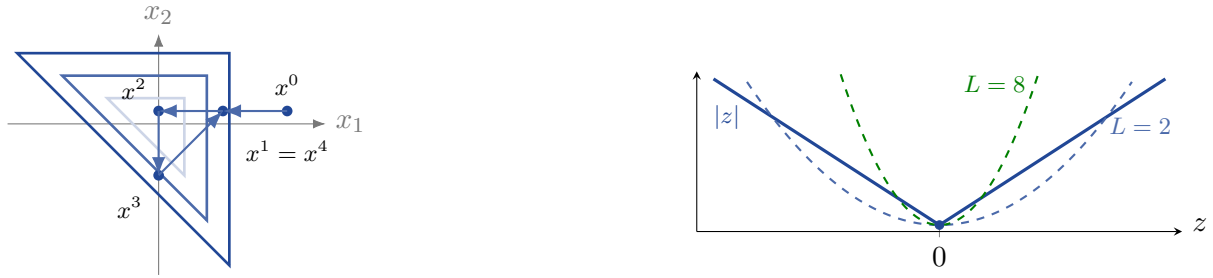


Figure 7.3: Left: the subgradient method with fixed step $\alpha = 1$ on $\max\{x_1, x_2, -x_1 - x_2\}$ cycles through x^1, x^2, x^3 and never approaches the minimiser at the origin (Example 7.11). Right: no quadratic upper model exists at a kink; near 0 every parabola $\frac{L}{2}z^2$ (dashed, $L = 2$ and $L = 8$) dips below $|z|$.

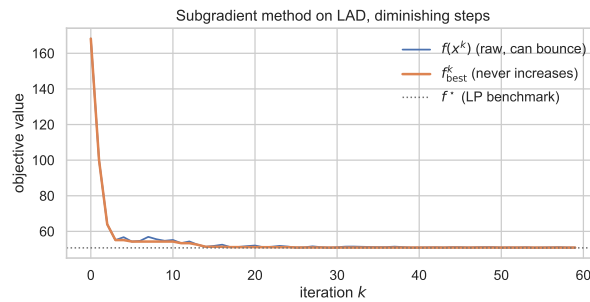


Figure 7.4: Measured behaviour on the LAD instance of Section 7.5 with $\alpha_k = 0.02/\sqrt{k+1}$ (first 60 iterations): the raw objective $f(x^k)$ bounces, the best-so-far value f_{best}^k never increases; dotted, the optimal value $f^* = 50.74$ from the linear-programming benchmark.

lies neither above nor below f in general, so minimising it says nothing about $f(x^{k+1})$. Nor can the upper model be restored by taking L large: for $f(z) = |z|$ at $x = 0$ with $g = 0$ it would require $|z| \leq \frac{L}{2}z^2$, which fails for $0 < |z| < 2/L$, however large L is (Figure 7.3, right).

There is a deeper contrast with Week 6. The direction $-\nabla f(x)$ is a descent direction whenever it is nonzero; the direction $-g$ with $g \in \partial f(x)$ need not be one in the sense that matters at a kink, where the test $\nabla f(x)^\top d < 0$ of Week 6 is unavailable: $f(x - \alpha g) < f(x)$ can fail for every $\alpha > 0$. At the minimiser $x = 0$ of $|x|$ every subgradient except $g = 0$ steps away from the solution, and the same can happen away from a minimiser: for $f(x) = |x_1| + 2|x_2|$ at $x = (1, 0)$ the vector $g = (1, 2)$ is a subgradient, yet $f(x - \alpha g) = |1 - \alpha| + 4\alpha \geq 1 + 3\alpha > f(x)$ for every $\alpha > 0$, so no step along $-g$ decreases the objective. What $-g$ does achieve, as Lemma 7.13 will show, is a decrease in the *distance to x^** .

7.4.2 What to track instead

Definition 7.12 (Best-so-far value). For iterates $x^0, x^1, \dots$ the **best-so-far** objective value after iteration k is

$$f_{\text{best}}^k = \min_{0 \leq i \leq k} f(x^i).$$

The raw sequence $f(x^k)$ can bounce, but f_{best}^k never increases, and it is the quantity that the theory bounds. Figure 7.4 shows both on a real LAD run: the raw value falls from 168.2 to 55.1 in three steps and then rises to 56.8 at the fourth, while the best-so-far trace is a descending staircase. Averaging the iterates is another common summary (it returns in Theorem 7.18); best-so-far tracking is the simplest reliable one. The practical lesson is to log and plot rather than trust any one iterate in isolation.

7.4.3 The one-step inequality and the master bound

Since objective values are unreliable, the analysis follows the squared distance to a minimiser.

Lemma 7.13 (One-step inequality). *Let f be convex with minimiser $x^\star$, let $g^k \in \partial f(x^k)$ and $x^{k+1} = x^k - \alpha_k g^k$ with $\alpha_k > 0$. Then*

$$\|x^{k+1} - x^\star\|_2^2 \leq \|x^k - x^\star\|_2^2 - 2\alpha_k(f(x^k) - f^\star) + \alpha_k^2 \|g^k\|_2^2. \quad (7.7)$$

Proof. Expand the square: $\|x^k - \alpha_k g^k - x^\star\|_2^2 = \|x^k - x^\star\|_2^2 - 2\alpha_k g^{k\top}(x^k - x^\star) + \alpha_k^2 \|g^k\|_2^2$. The subgradient inequality (7.2) at x^k with $y = x^\star$ reads $f^\star \geq f(x^k) + g^{k\top}(x^\star - x^k)$, that is, $g^{k\top}(x^k - x^\star) \geq f(x^k) - f^\star$. Substitute this into the middle term. $\square$

The computation is the one in the proof of Theorem 6.11, which expanded the same square (step (2)) and telescoped the same distances (step (3)), as the proof of Theorem 7.14 will. There the decrease $\frac{t}{2} \|g^k\|_2^2$ guaranteed by the descent lemma exactly cancelled the squared-step term of the expansion, $t^2 \|g^k\|_2^2$ divided by $2t$. Without smoothness nothing cancels the term $\alpha_k^2 \|g^k\|_2^2$, and this one surviving term is what forces the bounded subgradients, the shrinking steps and the slower rate below.

Read the inequality term by term. The middle term is the progress: the larger the current objective error, the more the step reduces the distance to $x^\star$. The last term fights against it, and grows with large steps and large subgradients. Whenever $f(x^k) > f^\star$ and $0 < \alpha_k < 2(f(x^k) - f^\star) / \|g^k\|_2^2$, the last two terms sum to a negative number: the step moves *closer* to $x^\star$, even when it moves uphill. To obtain a rate we need a scale for the last term.

We assume **bounded subgradients**: $\|g^k\|_2 \leq G$ for all iterations, a cap on how steep f can be. A function is called **G -Lipschitz** if $|f(x) - f(y)| \leq G \|x - y\|_2$ for all x, y , and the two notions agree: if every subgradient satisfies $\|g\|_2 \leq G$, then (7.2) and the Cauchy-Schwarz inequality (Proposition 1.11) give $f(x) - f(y) \leq g^\top(x - y) \leq G \|x - y\|_2$, and conversely a G -Lipschitz convex function has no subgradient longer than G . The constant G plays the role that L played in Week 6, except that L caps the *curvature* whereas G caps the *slope*; for $|x|$ we have $G = 1$, and for LAD $\|A^\top s\|_2 \leq \sum_i \|a_i\|_2$ since $|s_i| \leq 1$. The last term of (7.7) is then at most $\alpha_k^2 G^2$, a constant that no longer depends on the iterates, and the step sizes are the only free quantities left.

Theorem 7.14 (Master bound for the best iterate). *Let f be convex with minimiser $x^\star$, and suppose the subgradient method is run with $\|g^i\|_2 \leq G$ for $i = 0, \dots, k$. Then*

$$f_{\text{best}}^k - f^\star \leq \frac{\|x^0 - x^\star\|_2^2 + G^2 \sum_{i=0}^k \alpha_i^2}{2 \sum_{i=0}^k \alpha_i}. \quad (7.8)$$

Proof. Sum (7.7), with $\alpha_i^2 G^2$ as last term, over $i = 0, \dots, k$. The distances telescope, and dropping $\|x^{k+1} - x^\star\|_2^2 \geq 0$ leaves $2 \sum_{i=0}^k \alpha_i (f(x^i) - f^\star) \leq \|x^0 - x^\star\|_2^2 + G^2 \sum_{i=0}^k \alpha_i^2$. Each gap $f(x^i) - f^\star$ is at least the smallest one, $f_{\text{best}}^k - f^\star$; factor it out and divide by $2 \sum_i \alpha_i$. $\square$

The structure matters more than the algebra: the denominator, the total distance the method may travel, must be large, while the accumulated squared steps in the numerator must not explode. Every classical step-size rule is a different answer to this trade-off.

Key idea

A subgradient step need not decrease f , but a short enough one decreases the distance to x^* ; the theory therefore bounds the best-so-far value, and every step-size rule trades the distance to be covered against the error $\alpha_k^2 G^2$ that each step adds.

7.4.4 Step-size rules

Corollary 7.15 (Two step-size rules). *Under the hypotheses of Theorem 7.14, let $R \geq \|x^0 - x^*\|_2$.*

(a) Fixed horizon. *With a constant step α for K iterations, $f_{\text{best}}^{K-1} - f^* \leq \frac{R^2}{2K\alpha} + \frac{G^2\alpha}{2}$. The right-hand side is smallest for $\alpha = \frac{R}{G\sqrt{K}}$, which gives $f_{\text{best}}^{K-1} - f^* \leq \frac{RG}{\sqrt{K}}$.*

(b) Diminishing steps. *With $\alpha_k = c/\sqrt{k+1}$ for a constant $c > 0$,*

$$f_{\text{best}}^{K-1} - f^* \leq \frac{R^2 + G^2 c^2 (1 + \ln K)}{4c(\sqrt{K+1} - 1)} = O(\ln K / \sqrt{K}),$$

which tends to zero as $K \rightarrow \infty$ with no horizon fixed in advance.

Both bounds hold for f_{best}^K as well, since $f_{\text{best}}^K \leq f_{\text{best}}^{K-1}$.

Proof. (a) Put $\alpha_i = \alpha$ and $k = K - 1$ in (7.8). The two terms have constant product, so their sum is smallest when they are equal, $R^2/(2K\alpha) = G^2\alpha/2$, that is $\alpha = R/(G\sqrt{K})$; each term then equals $RG/(2\sqrt{K})$. (b) Comparing sums with integrals, $\sum_{k=0}^{K-1} (k+1)^{-1/2} \geq 2(\sqrt{K+1} - 1)$ and $\sum_{k=0}^{K-1} (k+1)^{-1} \leq 1 + \ln K$; substitute in (7.8). $\square$

In rule (a) the two terms pull in opposite directions. The first is the cost of covering the distance R : with a small α the method barely moves. The second is the error added per step, which acts like noise: with a large α every step overshoots, as in Example 7.10. The best constant step balances them, and to guarantee accuracy ε one needs $K \geq (RG/\varepsilon)^2$ iterations. For $|x|$ from $x^0 = 1.4$ ($R = 1.4$, $G = 1$), a budget of $K = 100$ gives $\alpha = 0.14$ and the guarantee 0.14; the accuracy 0.01 needs $K \geq 19600$. The drawback is that the horizon must be fixed in advance: a constant step does not converge, it only reaches a neighbourhood whose size it determines.

Rule (b) removes the horizon. The master bound needs $\sum_k \alpha_k$ to diverge, so that the iterates can travel any finite distance R , and $\sum_k \alpha_k^2$ to stay under control, so that the noise does not accumulate. The choice $c/\sqrt{k+1}$ sits at the boundary: $\sum_{k < K} \alpha_k \approx 2c\sqrt{K} \rightarrow \infty$, while $\sum_{k < K} \alpha_k^2 \approx c^2 \ln K$ grows only logarithmically. The step shrinks fast enough to stop the oscillation across a kink, yet slowly enough never to stall short of the minimum. The bound in (b) is smallest for $c = R/(G\sqrt{1 + \ln K})$, somewhat below R/G ; a much larger c prolongs the oscillation and a much smaller one slows the approach.

Definition 7.16 (Polyak step size). When the optimal value f^* is known, the **Polyak step size** is

$$\alpha_k = \frac{f(x^k) - f^*}{\|g^k\|_2^2}.$$

The formula is not arbitrary: it is the value of α_k that minimises the right-hand side of the one-step inequality (7.7), a quadratic in α_k , and it is half of the largest step that is still guaranteed to move closer to x^* . Substituting it gives $\|x^{k+1} - x^*\|_2^2 \leq \|x^k - x^*\|_2^2 - (f(x^k) - f^*)^2 / \|g^k\|_2^2$, so the distance to x^* never increases, and summing as before yields $f_{\text{best}}^k - f^* \leq RG/\sqrt{k+1}$ without choosing a horizon or a constant. The step is large when the iterate is far from optimal and shrinks automatically as the gap closes: on $|x|$ from $x^0 = 1.4$ it is $\alpha_0 = 1.4$ and lands on the minimiser in one step, and on Example 7.11 it reaches $x^* = 0$ in two steps ($\alpha_0 = 2$, $\alpha_1 = 0.2$).

What the rule needs is a true or well-estimated f^* , which is rarely available outside benchmarking or problems with a known target value; a poor estimate makes the step unreliable.

7.4.5 The price of nonsmoothness

All three rules deliver the rate $O(1/\sqrt{k})$ (up to the logarithm in rule (b)), against the $O(1/k)$ of gradient descent on smooth convex functions (Theorem 6.11). Constants aside, an accuracy of 10^{-2} costs about 10^2 iterations at rate $1/k$ and about 10^4 at rate $1/\sqrt{k}$, and each further digit multiplies the first count by ten and the second by a hundred. Nonsmoothness is algorithmically expensive even though convexity is preserved.

Remark 7.17 (The rate cannot be improved in general). The $O(1/\sqrt{k})$ rate is not a weakness of this particular method: it is known, and we state it without proof, that *no* method using only subgradient evaluations can guarantee a better order, with a bound that does not depend on the dimension d , over the whole class of convex G -Lipschitz functions. This is a worst-case statement about a class, not about each instance: with extra information a subgradient method can be far faster, as the Polyak runs above show. Week 10 returns to it: this is why momentum cannot accelerate the subgradient method.

7.4.6 Strong convexity: from $O(1/\sqrt{k})$ to $O(1/k)$

In Week 6 a curvature floor μ upgraded gradient descent from a sublinear to a linear rate. It helps here too. Recall from Definition 6.12 that f is μ -strongly convex if $f - \frac{\mu}{2} \|\cdot\|_2^2$ is convex; for a nonsmooth f the consequence we use is the first-order inequality with a subgradient in place of the gradient,

$$f(y) \geq f(x) + g^\top(y - x) + \frac{\mu}{2} \|y - x\|_2^2 \quad \text{for all } x, y \text{ and all } g \in \partial f(x). \quad (7.9)$$

It follows as in the differentiable case of Definition 6.12: $h = f - \frac{\mu}{2} \|\cdot\|_2^2$ is convex and $f = h + \frac{\mu}{2} \|\cdot\|_2^2$, so by the sum rule of Proposition 7.4 (with equality) every $g \in \partial f(x)$ satisfies $g - \mu x \in \partial h(x)$. The subgradient inequality for h , after collecting the terms in μ , is (7.9). Write $\Delta_k := \|x^k - x^*\|_2^2$ for the squared distance to the (now unique) minimiser.

Theorem 7.18 (Subgradient method under strong convexity). *Let f be μ -strongly convex and let $\|g^k\|_2 \leq G$ along the iterates of the subgradient method. Then*

$$\Delta_{k+1} \leq (1 - 2\mu\alpha_k) \Delta_k + \alpha_k^2 G^2. \quad (7.10)$$

Consequently, writing $\nu = \max\{\beta^2 G^2 / (2\mu\beta - 1), \gamma \Delta_0\}$ in part (b):

- (a) with a fixed step $0 < \alpha < 1/(2\mu)$: $\Delta_k \leq (1 - 2\mu\alpha)^k \Delta_0 + \alpha G^2 / (2\mu)$;
- (b) with $\alpha_k = \beta / (\gamma + k)$, where $2\mu\beta > 1$ and $\gamma \geq 2\mu\beta$: $\Delta_k \leq \nu / (\gamma + k) = O(1/k)$;
- (c) with $\alpha_k = 2 / (\mu(k + 2))$: $f(\bar{x}^k) - f^* \leq 2G^2 / (\mu(k + 2)) = O(1/k)$ for the weighted average $\bar{x}^k = \frac{2}{(k+1)(k+2)} \sum_{i=0}^k (i+1) x^i$.

Proof. Apply (7.9) twice: at x^k with $y = x^*$ and subgradient g^k , and at x^* with $y = x^k$ and the subgradient $0 \in \partial f(x^*)$. Adding the two inequalities, the function values cancel and

$$g^{k\top}(x^k - x^*) \geq \mu \Delta_k.$$

Curvature thus strengthens the signal: where the proof of Lemma 7.13 used only $g^{k\top}(x^k - x^*) \geq f(x^k) - f^* \geq 0$, the inner product is now bounded below by the distance itself. Substituting into the expanded square gives (7.10). (a) Unroll the recursion with constant α and bound the geometric sum $\sum_{j \geq 0} (1 - 2\mu\alpha)^j$ by $1/(2\mu\alpha)$. (b) Induction on k , writing $\hat{k} = \gamma + k$: the

Table 7.1: What each assumption buys: bounds on $f - f^*$ for the gradient method (Week 6; its analysis uses L) and the subgradient method (uses only G).

Assumption on f	Gradient method	Subgradient method
smooth, convex	$O(1/k)$, last iterate (Theorem 6.11)	$O(1/\sqrt{k})$, best-so-far
smooth, μ -strongly convex	linear, $(1 - \mu/L)^k$ (Theorem 6.13)	$O(1/k)$, average $\bar{x}^k$
nonsmooth, convex	not available	$O(1/\sqrt{k})$, best-so-far (Corollary 7.15)
nonsmooth, μ -strongly convex	not available	$O(1/k)$, average $\bar{x}^k$ (Theorem 7.18)

case $k = 0$ is $\nu \geq \gamma\Delta_0$, and if $\Delta_k \leq \nu/\hat{k}$ then, since $1 - 2\mu\beta/\hat{k} \geq 0$ by the condition on γ , $\Delta_{k+1} \leq (1 - 2\mu\beta/\hat{k})\nu/\hat{k} + \beta^2 G^2/\hat{k}^2 \leq (\hat{k} - 1)\nu/\hat{k}^2 \leq \nu/(\hat{k} + 1)$, where the middle step uses $\nu \geq \beta^2 G^2/(2\mu\beta - 1)$ and the last uses $(\hat{k} - 1)(\hat{k} + 1) \leq \hat{k}^2$. (c) Using (7.9) at x^k only keeps the objective gap in the recursion: $\Delta_{k+1} \leq (1 - \mu\alpha_k)\Delta_k - 2\alpha_k(f(x^k) - f^*) + \alpha_k^2 G^2$. Multiplying the inequality of iteration i by $(i + 1)/(2\alpha_i)$ and summing over $i = 0, \dots, k$ makes the distance terms telescope and leaves $\sum_{i=0}^k (i + 1)(f(x^i) - f^*) \leq (k + 1)G^2/\mu$; divide by the total weight $\frac{1}{2}(k + 1)(k + 2)$ and use convexity. $\square$

Parts (a) and (b) tell a story that returns in Week 11. With a fixed step the distance contracts geometrically, but only down to a *floor* $\alpha G^2/(2\mu)$: the method reaches a neighbourhood of x^* quickly and then rattles around inside it, and halving the step halves the floor while slowing the contraction. A step decaying like $1/k$ removes the floor. Here β , γ and ν are constants of the schedule (β is unrelated to the regression coefficients of Model 7.6), and since $2\mu\beta > 1$ requires a lower estimate of μ , the update is the one analysed before but the step-size rule is not.

Remark 7.19 (Objective values and the role of G). A bound on Δ_k alone gives only $f(x^k) - f^* \leq G\sqrt{\Delta_k} = O(1/\sqrt{k})$, which is why part (c) keeps the gap in the recursion and averages; its argument does not use (7.10), so the condition $\gamma \geq 2\mu\beta$ plays no role there. Moreover, G bounds the subgradients only over the bounded region that the iterates visit: over all of $\mathbb{R}^d$ a strongly convex function grows quadratically and cannot have bounded subgradients. Week 11 replaces the bound by a noise model that removes this restriction.

Note on the lecture slides

Frame 7.31 states $\Delta_k \leq \nu/(\gamma + k)$ without defining ν or restricting γ . The notes define ν and add the hypothesis $\gamma \geq 2\mu\beta$, without which the bound can fail: for $f(x) = \frac{1}{2}x^2$, $\beta = 5$, $\gamma = 1$, $x^0 = 1$ the iterates are 1, -4, 6, -4, 1, 0, so $G = 6$ and $\nu = 100$, yet $\Delta_2 = 36 > \nu/3$. The frame's averaging claim leaves $\bar{x}^k$ and the step size undefined and does not follow from “the same recursion”; part (c) supplies both (weights $i + 1$, as iterates are numbered from 0).

7.4.7 What each assumption buys

Table 7.1 puts the two methods side by side; every entry bounds $f - f^*$ at the iterate named beside it. The gradient method is *not available* in the last two rows, because ∇f does not exist at a kink. The last row is not exotic: ridge-regularised LAD, $\frac{1}{N}\|A\beta - b\|_1 + \frac{\mu}{2}\|\beta\|_2^2$, is nonsmooth and μ -strongly convex (a composite objective of the kind studied in Week 9). And the subgradient column does not improve when f happens to be smooth, because its analysis uses only $\|g^k\|_2 \leq G$. The rate follows the assumption that the analysis exploits; later weeks gain speed by exploiting more structure, not by wishing the kinks away.

7.5 Diagnostics, robust regression, and the verdict

7.5.1 Stopping and diagnostics

Gradient descent stops when $\|\nabla f(x^k)\|_2$ is small. The norm of one chosen subgradient is not a dependable certificate in the same way: at the minimiser of $|x|$ the set $\partial f(0) = [-1, 1]$ contains 0 but also nonzero slopes, and arbitrarily close to the minimiser every subgradient has norm 1. Optimality is a statement about the whole set $\partial f(x^k)$, whereas the method sees one element of it; in the Polyak run below, $\|g^k\|_2$ stays between about 3.5 and 12 throughout the 500 iterations in which the objective gap is already below 10^{-2} . Practical stopping therefore combines several signals: the best objective value and its relative improvement over a window of iterations; an iteration or time budget; and, where possible, comparison with a solver benchmark on a small instance or with a known target value. Neither a single subgradient norm nor a single jump in the raw objective should be trusted blindly.

7.5.2 Least-absolute-deviations regression: two solution routes

In LAD regression (Model 7.6) the modelling idea and the algorithmic difficulty are both visible. Penalising residuals linearly rather than quadratically makes the fit less sensitive to a few unusually large residuals, for the reason seen in Example 7.8: the condition $0 \in \partial f(\beta^*)$ involves only the *signs* of the nonzero residuals.

Example 7.20 (A line through six points, one of them an outlier). Fit $b \approx \beta_1 + \beta_2 a$ to the points $(1, 1.1)$, $(2, 1.9)$, $(3, 3.0)$, $(4, 3.8)$, $(5, 5.0)$ and $(6, 7.1)$; the last point lies well above the trend of the other five. Least squares gives the line $-0.36 + 1.146a$: it is tilted towards the outlier and consequently misses most of the other points (without the sixth point it would be $0.05 + 0.97a$). The LAD fit is the line $b = a$, with total absolute deviation $0.1 + 0.1 + 0 + 0.2 + 0 + 1.1 = 1.5$; it passes through the bulk of the data and leaves the outlier with a large residual (Figure 7.5, left). Optimality can be certified by hand. The residuals $A\beta - b$ have signs $(-, +, 0, +, 0, -)$, so by (7.4) the subgradients are $A^\top s$ with $s = (-1, 1, s_3, 1, s_5, -1)$ and $s_3, s_5 \in [-1, 1]$. The choice $s_3 = -\frac{1}{2}$, $s_5 = \frac{1}{2}$ gives $\sum_i s_i = 0$ and $\sum_i s_i a_i = -1 + 2 - \frac{3}{2} + 4 + \frac{5}{2} - 6 = 0$, so $0 \in \partial f(\beta^*)$. The two zero residuals make the balance possible: a LAD line typically passes through as many points as it has coefficients, just as the median sits on a data point.

Note on the lecture slides

On frame 7.35 the lines labelled “least squares” ($0.95x + 0.05$) and “LAD” ($0.75x + 0.3$) are schematic, not the fits of the six plotted points: the first is close to the least-squares fit *without* the outlier, and the second lies below all six points (total absolute deviation 4.35, against 1.5 for the LAD fit). Figure 7.5 shows the computed fits; the slide’s message is unchanged.

LAD can be solved by two quite different routes. The first is the reformulation pattern of Proposition 4.8: introduce one auxiliary variable per residual to convert the absolute values into linear inequalities.

Model 7.21 (LAD as a linear programme).

$$\begin{aligned} \min_{\beta \in \mathbb{R}^d, t \in \mathbb{R}^N} \quad & \mathbf{1}^\top t \\ \text{s.t.} \quad & -t \leq A\beta - b \leq t. \end{aligned} \tag{7.11}$$

Here t is a vector of auxiliary variables, unrelated to a step size. The constraints say $t_i \geq |(A\beta - b)_i|$, and since the objective pushes every t_i down, $t_i = |(A\beta - b)_i|$ at an optimum, so the optimal value is f^* and the β -part of a solution solves (7.3). This linear programme,

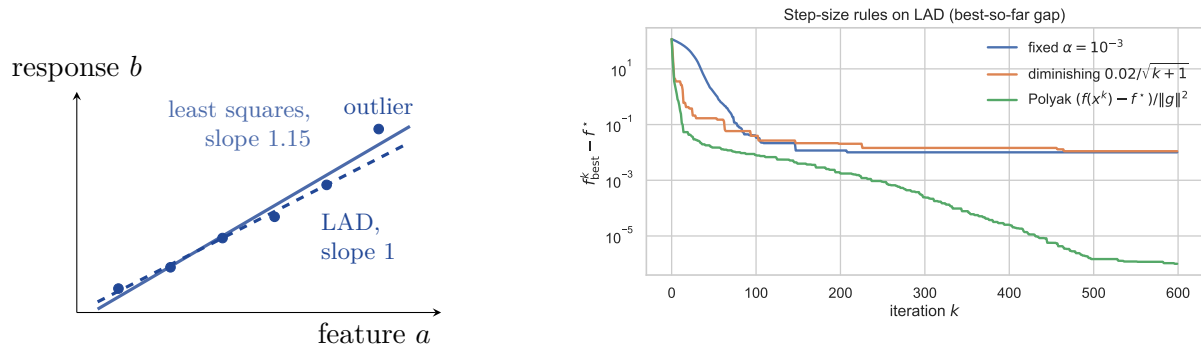


Figure 7.5: Left: the computed fits of Example 7.20; least squares is pulled towards the outlier, LAD follows the other five points. Right: measured best-so-far gap $f_{\text{best}}^k - f^*$ (logarithmic axis) of the subgradient method on a LAD instance with $N = 80$ and $d = 8$ under three step-size rules; the Polyak rule is given the true f^* from the linear-programming benchmark.

with $d + N$ variables and $2N$ constraints, is what RSOME passes to a solver (Section 4.5). The second route is Algorithm 7.1 with the subgradient (7.4): a few lines of NumPy with $O(Nd)$ work per iteration. The routes are complements. The solver gives a trusted optimal value on small instances, the custom route shows the algorithmic behaviour and invites experiments, and agreement between the two is a valuable correctness check.

7.5.3 Implementation, experiments and a verdict

Listing 7.1 carries out both routes on a synthetic instance with $N = 80$ observations, $d = 8$ coefficients (an intercept and seven Gaussian features), noise of standard deviation 0.3 and six responses shifted by ± 6 to act as outliers. The loop is problem-independent apart from the two one-line functions returning f and one subgradient, and its logging is as important as its update. In the code the variable $\mathbf{x}$ holds the coefficient vector β . The loop returns the last iterate; to return the best point as Algorithm 7.1 does, store a copy of $\mathbf{x}$ whenever **best** improves.

Listing 7.1: LAD regression: RSOME benchmark and the subgradient method with three step-size rules.

```
import numpy as np
from rsome import ro, grb_solver as grb
rng = np.random.default_rng(7)
N, d = 80, 8
A = np.column_stack([np.ones(N), rng.normal(size=(N, d - 1))])
b = A @ rng.normal(size=d) + 0.3 * rng.normal(size=N)
b[rng.choice(N, 6, replace=False)] += rng.choice([-6, 6], size=6) # outliers
model = ro.Model('LAD-LP') # route 1: LP benchmark
bv, tv = model.dvar(d), model.dvar(N) # beta and t_i >= |(A beta - b)_i|
model.min(tv.sum())
model.st(A @ bv - b <= tv, -(A @ bv - b) <= tv)
model.solve(grb, display=False)
f_star = model.get()
def objective(x): return np.abs(A @ x - b).sum()
def choose_subgradient(x): return A.T @ np.sign(A @ x - b)
def subgradient_method(x0, step_rule, maxit): # route 2
    x = x0.copy()
    best = np.inf
    hist_f, hist_best = [], []
    for k in range(maxit):
        f = objective(x)
        g = choose_subgradient(x)
        best = min(best, f)
        hist_f.append(f)
        hist_best.append(best)
```

```

    alpha = step_rule(k, f, g)
    x = x - alpha * g
    return x, np.array(hist_f), np.array(hist_best)
rules = {"fixed": lambda k, f, g: 1e-3,
        "diminishing": lambda k, f, g: 0.02 / np.sqrt(k + 1),
        "Polyak": lambda k, f, g: (f - f_star) / (g @ g)}
print(f"f* = {f_star:.4f}")
for name, rule in rules.items():
    x, hist_f, hist_best = subgradient_method(np.zeros(d), rule, 600)
    print(f"{name:12s} best-so-far gap = {hist_best[-1] - f_star:.2e}")

```

The printed output is $f^* = 50.7420$, followed by the gaps $1.01\text{e-}02$ (fixed), $1.10\text{e-}02$ (diminishing) and $1.00\text{e-}06$ (Polyak; its last digits depend on the solver); Figure 7.5 (right) shows the traces, starting from $f(0) = 168.2$. The tuned fixed step makes early progress and then stalls at a gap floor set by its size: its last improvement within the 600 iterations is at iteration 208. Diminishing steps are calmer and keep improving, but visibly slow down; after 600 iterations they are level with the fixed step. The Polyak rule is strikingly effective, but only because the benchmark supplies the *true* f^* . In every run the raw objective rises on about half of the steps.

Further runs of the same loop (not listed) sharpen the picture. After 20 000 iterations the fixed step has reached only 8.1×10^{-3} and the diminishing rule 1.4×10^{-3} : the advantage of diminishing steps shows only on long runs. With the target $0.9f^*$ in place of f^* the Polyak run ends with a gap of 0.53. The guarantees are pessimistic: with $R = \|\beta^*\|_2 = 2.29$ and $G = 60.5$ from the fixed-step run, the bound $RG/\sqrt{600}$ of Corollary 7.15 is only 5.7.

The verdict is mixed. The subgradient method is simple, general and often the right first thing to code for a nonsmooth convex problem: it applies whenever one subgradient can be computed, and an iteration is as cheap as a gradient step. It is also slow, sensitive to step-size tuning, awkward to stop, and blind to structure, treating an ℓ_1 norm and a worst-case maximum alike as black boxes that return a slope.

7.6 Summary and looking ahead

A subgradient is a slope whose affine function supports a convex function globally from below, a supporting hyperplane of the epigraph. At a differentiable point it is the gradient; at a kink there is a closed convex set of them, the subdifferential, computed by the sum, affine-composition and active-max rules. A point is a minimiser exactly when $0 \in \partial f(x^*)$, which produced the median, a convex-hull test for minimax problems and a hand-checkable certificate for a LAD fit.

What the subgradient method does is minimise the stabilised local model of Week 6 with any subgradient in place of the gradient, and remember the best point seen. *Why it works* is the one-step inequality: a short enough step need not decrease the objective, but it decreases the distance to the solution. Summing gives the master bound and the $O(1/\sqrt{k})$ rate of the fixed-horizon, diminishing and Polyak rules; strong convexity with steps proportional to $1/k$ improves this to $O(1/k)$. *It applies* whenever f is convex, one subgradient can be computed, and the subgradients met along the iterates are bounded. *Its limitations* are those of the verdict: a slow rate that no subgradient-only method can improve in the worst case, sensitive step-size constants, no reliable stopping residual, and blindness to structure.

The following weeks improve on this baseline one structural ingredient at a time. Week 8 adds a simple constraint set. It returns to differentiable objectives and keeps every iterate feasible by following each gradient step with a projection onto the set. The resulting projected-gradient method keeps the step sizes and rates of Week 6, and a projection after this week's subgradient step leaves the one-step inequality and step-size rules intact. Week 9 treats objectives of the form smooth plus simple nonsmooth, such as least squares with an ℓ_1 penalty, and recovers the $O(1/k)$ rate of Week 6 by handling the kinked term through a proximal step instead of a subgradient. Week 10 accelerates those methods and explains why the same device cannot help

here, [Week 11](#) reuses this week's analysis almost line by line for random gradients, and [Week 12](#) meets subgradients again at the kinks of a dual function.

7.7 Exercises

Exercise 7.1 (Warm-up subdifferentials). Compute $\partial f_j(x)$ at every $x \in \mathbb{R}$ for $f_1(x) = |x|$, $f_2(x) = \max\{0, x\}$ and $f_3(x) = \max\{x + 1, -x + 2\}$, and explain each answer geometrically. [Week 3](#) found such sets at single points; here describe the whole map $x \mapsto \partial f_j(x)$, sketch it as a set-valued graph, and check it against the active-max rule of [Proposition 7.4](#).

Exercise 7.2 (Median by optimality). For $\min_x \sum_{i=1}^N |x - c_i|$, use the condition $0 \in \partial f(x^*)$ to show that any median is optimal. Apply your argument to the data $c = (1, 2, 4, 7)$: describe the set of all minimisers and give the optimal value.

Exercise 7.3 (Max-of-affine subgradients). Let $f(x) = \max\{x_1 + 2x_2, -x_1 + x_2 + 1, 2 - x_2\}$ on $\mathbb{R}^2$. (a) At $x = (1, 0)$ identify the active pieces and compute a subgradient. (b) At $x = (2, 0)$ two pieces are active; describe the whole subdifferential. (c) Show by the convex-hull test that $x^* = (0.2, 0.6)$ is a minimiser, and give f^* .

Exercise 7.4 (Manual iterations). Starting from $x^0 = 2$, apply the subgradient method to $f(x) = |x|$ for six iterations, once with $\alpha_k = 0.6$ and once with $\alpha_k = 1/\sqrt{k+1}$. Record the iterates, the objective values and f_{best}^k . Compare the two sequences and comment on oscillation.

Exercise 7.5 (LAD subgradient). For $f(\beta) = \|A\beta - b\|_1$, show that one subgradient can be written as $A^\top s$ with suitable componentwise choices of s . Then write one iteration of the subgradient method.

Exercise 7.6 (Benchmark reformulation). Reformulate $\min_\beta \|A\beta - b\|_1$ as a linear programme using auxiliary variables t_i . Explain why $t_i = |(A\beta - b)_i|$ at an optimum, and why this is a useful RSOME benchmark.

Exercise 7.7 (Step-size comparison in code). Implement the subgradient method for a small LAD or max-of-affine instance, starting from [Listing 7.1](#). Compare (i) a constant step tuned for a chosen horizon K , such as $\alpha = R/(G\sqrt{K})$ with $R = \|x^0 - x^*\|_2$ and $G = \max_k \|g^k\|_2$; (ii) $\alpha_k = c/\sqrt{k+1}$; (iii) a Polyak step with a benchmark f^* from an RSOME linear programme. Plot the raw and the best-so-far objective values.

Exercise 7.8 (Stopping discussion). Why is one chosen $\|g^k\|_2$, $g^k \in \partial f(x^k)$, not by itself a reliable stopping test? Give an example using $|x|$ or a max-of-affine function, and suggest two alternatives.

Week 8 Simple Constraints, Projection, and the Projected Gradient Method

How do we stay feasible at every step without losing the gradient? [Weeks 6](#) and [7](#) built two algorithms, gradient descent for smooth objectives and the subgradient method for nonsmooth ones, and in both every iterate was free to move anywhere in $\mathbb{R}^d$. Many analytics models are not like that: regression coefficients must be nonnegative, portfolio weights must be nonnegative and sum to one, decisions must respect lower and upper bounds. [Week 5](#) showed how to *certify* a solution of such a constrained problem through the KKT conditions, but the only way we have had of *computing* one is to call a solver. This week gives the first algorithm of the course that handles constraints explicitly: we minimise a convex differentiable f over a feasible set $C \subseteq \mathbb{R}^d$ and ask what must change when every iterate has to remain in C . We return to differentiable objectives so that the constraint is the only new ingredient; the nonsmooth case, a subgradient step of [Week 7](#) followed by a projection, returns at the end of the week.

The answer is the smallest possible modification of gradient descent. It needs one new operation, the *projection* $\Pi_C(y)$, the point of C closest to y , and one new analytical fact, that projection is *non-expansive*: it never increases the distance between two points. The update becomes $x^{k+1} = \Pi_C(x^k - \alpha_k \nabla f(x^k))$, a gradient step followed by a return to the feasible set, and the step sizes, curvature constants L , μ and convergence factor of [Week 6](#) carry over. The price is a restriction on C : its projection must be cheap to compute.

[Section 8.1](#) presents three models with such constraints and shows why a bare gradient step fails on them. [Section 8.2](#) rebuilds the optimality certificate: writing the constraint as an indicator function turns the four KKT blocks of [Week 5](#) into the single line $0 \in \nabla f(x^*) + N_C(x^*)$, where N_C is the normal cone of C . [Section 8.3](#) studies projection as a computable primitive, [Section 8.4](#) combines the two into a fixed-point form of optimality, a stopping residual, the algorithm and its rate, and [Section 8.5](#) works the examples and closes with the projected subgradient method, to which the analysis of [Week 7](#) carries over. Notation: the feasible set is C this week, and again in [Week 9](#), [Week 10](#) and [Week 13](#) ([Week 1](#) to [Week 5](#), and later [Week 11](#), write $\mathcal{X}$), and the step size is α_k as in [Week 7](#) ([Week 6](#) wrote t_k).

8.1 When the unconstrained step fails

8.1.1 Three models with simple constraints

Three models motivate the week. Each has a smooth convex objective of a kind we have already minimised; what is new is a feasible set that the iterates may not leave.

Model 8.1 (Nonnegative least squares, NNLS). Given $A \in \mathbb{R}^{N \times d}$ and $b \in \mathbb{R}^N$,

$$\begin{aligned} \min_{x \in \mathbb{R}^d} \quad & \frac{1}{2} \|Ax - b\|_2^2 \\ \text{s.t.} \quad & x \geq 0. \end{aligned} \tag{8.1}$$

The decision variable x collects d coefficients, the objective is the least-squares misfit of [Model 6.16](#) over N data points, and the constraint requires every coefficient to be nonnegative. It is the right model when the coefficients are rates, intensities, concentrations or allocations, for which a negative estimate has no meaning. It is the constrained least-squares model of [Section 4.2](#) with the nonnegative orthant as constraint set, which [Week 4](#) wrote without the factor $\frac{1}{2}$ and handed to a solver through RSOME; this week computes it with an algorithm of our own, and the RSOME model becomes the benchmark ([Section 8.5](#)). The objective is convex for any data ([Example 3.13](#)), with gradient $\nabla f(x) = A^\top(Ax - b)$ by the chain rule ([Proposition 1.17](#)), and the feasible set is the nonnegative orthant $\mathbb{R}_+^d$. A solution is read like a regression fit, with

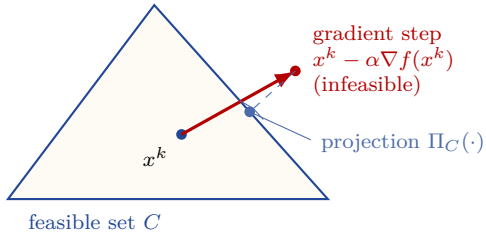


Figure 8.1: A gradient step from a feasible point x^k lands outside the feasible set C . The projected-gradient method keeps the step and then returns to the nearest point of C ; on a flat edge this is the foot of the perpendicular.

one addition: a coefficient that ends at exactly zero is one the constraint is holding back, and [Example 8.8](#) explains how the gradient certifies this.

Model 8.2 (Simplex-constrained allocation). Given a symmetric positive semidefinite $\Sigma \in \mathbb{R}^{d \times d}$ and $c \in \mathbb{R}^d$,

$$\begin{aligned} \min_{x \in \mathbb{R}^d} \quad & \frac{1}{2} x^\top \Sigma x - c^\top x \\ \text{s.t.} \quad & \mathbf{1}^\top x = 1, \quad x \geq 0. \end{aligned} \quad (8.2)$$

Here x_i is the fraction of a budget given to option i : a portfolio weight, a market share, a share of a marketing budget. The feasible set is the simplex of [Definition 2.12](#) and the objective is a convex quadratic ([Proposition 3.14](#), the model family of [Section 4.2](#)) with gradient $\Sigma x - c$. It is the long-only mean–variance portfolio of [Model 1.2](#), whose KKT system we wrote in [Section 5.4](#), with new letters: put $x = w$, $c = \mu$ (expected returns) and replace Σ by $2\gamma\Sigma$, so that minimising $\frac{1}{2} x^\top \Sigma x - c^\top x$ is maximising $\mu^\top w - \gamma w^\top \Sigma w$. (The symbol μ is reserved below for a curvature constant, hence c for the returns.)

Model 8.3 (Box-constrained smooth minimisation). Given a convex differentiable $f : \mathbb{R}^d \rightarrow \mathbb{R}$ and bounds $\ell, u \in \mathbb{R}^d$ with $\ell \leq u$,

$$\begin{aligned} \min_{x \in \mathbb{R}^d} \quad & f(x) \\ \text{s.t.} \quad & \ell \leq x \leq u. \end{aligned} \quad (8.3)$$

The bounds are legal, physical or business-policy limits on each decision separately: a price between a floor and a cap, a production quantity between zero and capacity. In all three models the objective is smooth and familiar; it is the feasible set that shapes the algorithm, and one method, that of [Section 8.4](#), solves all three (worked runs in [Section 8.5](#)).

8.1.2 Why a gradient step alone fails

Apply the gradient step $x^k - \alpha \nabla f(x^k)$ of [Algorithm 6.1](#) to any of these models, starting from a feasible x^k . The negative gradient points towards local decrease of f , but it knows nothing about C , and the new point may be infeasible, as in [Figure 8.1](#). Shrinking the step can keep an iterate in the interior of C feasible, but it does not solve the structural problem: as soon as x^k lies on the boundary of C and $-\nabla f(x^k)$ points outwards, *every* positive step leaves C , however small. The natural repair keeps the step and adds a correction: move in the descent direction, then return to the feasible set in the simplest possible way, to the feasible point nearest to where the step landed. That nearest point is the projection $\Pi_C(x^k - \alpha \nabla f(x^k))$, defined formally in [Section 8.3](#); the intuition suffices for now.

8.1.3 What makes a constraint set “simple”

The repair is only as cheap as the projection. We call a convex set C **simple** when its projection Π_C is available in closed form or is very cheap to compute. The word is meant algorithmically, not geometrically: intervals, boxes, the nonnegative orthant, Euclidean balls, hyperplanes, affine sets and the simplex are simple in this sense ([Table 8.1](#) lists their projections), whereas a general polyhedron $\{x : Ax \leq b\}$ ([Definition 2.11](#)), although geometrically just as tame, is not: finding

its nearest point is itself an optimisation problem that needs a solver. [Week 13](#) is about what to do then. For simple sets, a first-order method can afford to restore feasibility at every iteration, and that is the organising principle of this week.

Key idea

When projection onto the feasible set is cheap, gradient descent becomes projected gradient at little extra cost: take the usual gradient step, then project back. Every iterate is feasible.

Throughout this week C denotes a *nonempty, closed, convex* subset of $\mathbb{R}^d$ ([Definition 2.1](#); closed means that C contains its boundary points), which every set in the list above is; [Section 8.3](#) explains what each of the three words buys.

8.2 Constrained optimality: indicators and normal cones

Before building an algorithm we need to know what it should converge to, in a form that an algorithm can test. [Week 5](#) already gave a certificate, the KKT conditions. This subsection repackages it into one set-valued line that does not depend on how C is written down.

8.2.1 Recall: the KKT conditions

Consider minimising $f(x)$ subject to $g_i(x) \leq 0$ for $i = 1, \dots, m$ and $Ax = b$, with f and the g_i convex and differentiable. ([Week 5](#) wrote the objective f_0 and the constraint functions f_i ; this week they are f and g_i , as in [Definition 1.4](#). The symbol g^k returns in [Section 8.5](#) with its [Week 7](#) meaning of a subgradient.) The KKT conditions of [Definition 5.14](#) ask for multipliers $\lambda^* \in \mathbb{R}^m$ and ν^* such that

$$\begin{aligned} \text{stationarity:} & \quad \nabla f(x^*) + \sum_{i=1}^m \lambda_i^* \nabla g_i(x^*) + A^\top \nu^* = 0, \\ \text{primal feasibility:} & \quad g_i(x^*) \leq 0 \quad (i = 1, \dots, m), \quad Ax^* = b, \\ \text{dual feasibility:} & \quad \lambda_i^* \geq 0 \quad (i = 1, \dots, m), \\ \text{complementary slackness:} & \quad \lambda_i^* g_i(x^*) = 0 \quad (i = 1, \dots, m). \end{aligned} \tag{8.4}$$

By [Theorem 5.15](#) they are sufficient for optimality in a convex problem, and necessary under Slater's condition. Geometrically ([Section 5.2](#)) $-\nabla f(x^*)$ is balanced by the normals of the active constraints: a nonnegative combination of the $\nabla g_i(x^*)$ with $g_i(x^*) = 0$, plus any combination of the rows of A . The aim now is to describe the set of all such balancing vectors directly from C .

8.2.2 The indicator function

Definition 8.4 (Indicator function). The **indicator function** of a set $C \subseteq \mathbb{R}^d$ is

$$I_C(x) = \begin{cases} 0, & x \in C, \\ +\infty, & x \notin C. \end{cases}$$

The indicator charges nothing inside C and an infinite penalty outside, so

$$\min_{x \in C} f(x) \quad \Longleftrightarrow \quad \min_{x \in \mathbb{R}^d} f(x) + I_C(x) : \tag{8.5}$$

the two problems have the same minimisers and the same optimal value: the hard constraint has been folded into the objective, and the problem is now an unconstrained *composite* one, a smooth function plus a nonsmooth one. The function I_C takes the value $+\infty$, which the extended-value convention of [Week 3](#) allows, and it is convex exactly when C is convex, because its epigraph ([Definition 3.1](#)) is $C \times [0, \infty)$. The point is to reuse optimality logic we already have: [Theorem 7.7](#) says that a convex function is minimised where 0 belongs to its subdifferential, so

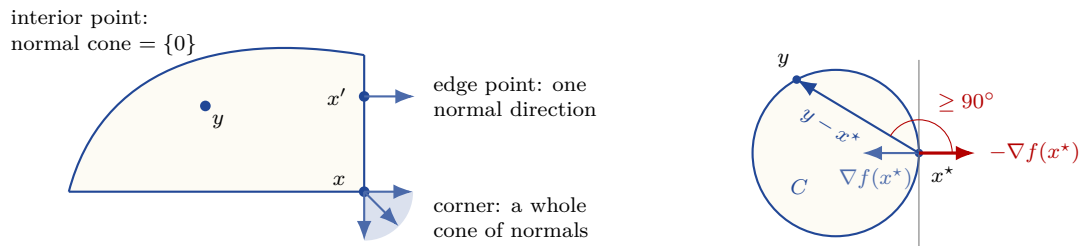


Figure 8.2: Left: the normal cone at an interior point ($\{0\}$), on a flat edge (a single outward ray) and at a corner (the cone spanned by the normals of the two edges). Right: the optimality condition of [Theorem 8.7](#): at the solution $-\nabla f(x^*)$ lies in the normal cone, so it makes an angle of at least 90° with every feasible direction $y - x^*$.

we need the subdifferential of I_C . [Definition 7.1](#) was written for finite-valued functions; for I_C we use the same inequality at points where the function is finite: $v \in \partial I_C(x)$, for $x \in C$, means $I_C(y) \geq I_C(x) + v^\top(y - x)$ for all $y \in \mathbb{R}^d$.

8.2.3 The normal cone

Definition 8.5 (Normal cone). The **normal cone** of C at a point $x \in C$ is

$$N_C(x) = \{v \in \mathbb{R}^d : v^\top(y - x) \leq 0 \text{ for all } y \in C\}.$$

For $x \notin C$ we set $N_C(x) = \emptyset$.

A vector v is normal to C at x when it makes an angle of at least 90° with every direction $y - x$ that leads from x to another feasible point: v points “away from C ”. For $x \in C$ the set $N_C(x)$ contains 0, and it is a convex cone in the sense of [Definition 2.6](#): if $v, w \in N_C(x)$ and $s, t \geq 0$ then $(sv + tw)^\top(y - x) \leq 0$ for all $y \in C$. The closure under positive scaling will matter in [Proposition 8.15](#).

Proposition 8.6 (Subdifferential of the indicator). For every $x \in C$, $\partial I_C(x) = N_C(x)$.

Proof. Let $x \in C$, so $I_C(x) = 0$ and the subgradient inequality reads $I_C(y) \geq v^\top(y - x)$ for all y . For $y \notin C$ the left-hand side is $+\infty$ and nothing is required; for $y \in C$ it reads $0 \geq v^\top(y - x)$. Hence $v \in N_C(x)$. $\square$

Note on the lecture slides

Frame 8.9 defines $N_C(x)$ and states $\partial I_C(x) = N_C(x)$ with no restriction on x . For $x \notin C$ the displayed set still contains 0 whereas $\partial I_C(x)$ is empty, so $0 \in \nabla f(x) + N_C(x)$ read literally could hold at an infeasible point ($C = \mathbb{R}_+$, $f(x) = \frac{1}{2}(x + 2)^2$, $x = -1$). The notes define the normal cone at points of C only and set it to $\emptyset$ elsewhere, as the later frames assume.

The left panel of [Figure 8.2](#) shows the three typical cases. If x is an *interior point* of C (a point with a small ball around it contained in C), then $N_C(x) = \{0\}$: for $v \in N_C(x)$ the point $y = x + \varepsilon v$ is feasible for small $\varepsilon > 0$, so $0 \geq v^\top(y - x) = \varepsilon \|v\|_2^2$ and $v = 0$. On a flat piece of the boundary the normal cone is the single outward ray perpendicular to it. At a corner it is a whole cone, spanned by the outward normals of the faces that meet there.

8.2.4 The optimality condition and the variational inequality

Theorem 8.7 (First-order optimality over a convex set). Let $f : \mathbb{R}^d \rightarrow \mathbb{R}$ be convex and differentiable and let C be nonempty, closed and convex. For $x^* \in C$ the following are equivalent:

- (i) x^* solves $\min_{x \in C} f(x)$;

(ii) $0 \in \nabla f(x^*) + N_C(x^*)$, that is, $-\nabla f(x^*) \in N_C(x^*)$;

(iii) the **variational inequality** holds:

$$\nabla f(x^*)^\top (y - x^*) \geq 0 \quad \text{for all } y \in C. \quad (8.6)$$

Proof. (ii) $\Leftrightarrow$ (iii) is Definition 8.5 with $v = -\nabla f(x^*)$. (iii) $\Rightarrow$ (i): by the first-order inequality for convex functions (Theorem 3.8), $f(y) \geq f(x^*) + \nabla f(x^*)^\top (y - x^*) \geq f(x^*)$ for every $y \in C$. (i) $\Rightarrow$ (iii): suppose $\nabla f(x^*)^\top (y - x^*) < 0$ for some $y \in C$. Because C is convex, the points $x^* + t(y - x^*)$, $0 \leq t \leq 1$, are feasible, and the directional derivative of f at x^* along $y - x^*$ is $\nabla f(x^*)^\top (y - x^*) < 0$ (Definition 1.15). Hence $f(x^* + t(y - x^*)) < f(x^*)$ for small $t > 0$, which contradicts (i). $\square$

Condition (ii) is the direct analogue of $\nabla f(x^*) = 0$ (Theorem 5.1) and the compact geometric form of KKT: the negative gradient must lie in the normal cone. Condition (iii) says the same without any new object: from x^* , no feasible direction $y - x^*$ points downhill. In the right panel of Figure 8.2, $-\nabla f(x^*)$ makes an angle of at least 90° with every feasible direction, equivalently $\nabla f(x^*)$ makes an angle of at most 90° with each of them. If the optimum is an interior point of C the normal cone is $\{0\}$ and the condition collapses to $\nabla f(x^*) = 0$: the constraint is irrelevant and we are back in Week 6. On the boundary, $\nabla f(x^*)$ is typically *not* zero, so $\|\nabla f(x^*)\|_2$ can no longer serve as a stopping test. Condition (ii) is also what (8.5) suggests, namely $0 \in \partial(f + I_C)(x^*)$ with the subdifferential of the sum split into $\nabla f(x^*) + N_C(x^*)$; but for the extended-valued I_C the sum rule of Proposition 7.4 supplies only the inclusion $\nabla f(x^*) + N_C(x^*) \subseteq \partial(f + I_C)(x^*)$ (Week 7 asserted equality only for finite-valued functions). This gives (ii) $\Rightarrow$ (i) but not the converse, which is why the proof above is direct (and uses the convexity of C in (i) $\Rightarrow$ (iii)).

8.2.5 Two languages for one fact: the orthant and the simplex

For the two most important simple sets we now check that the normal-cone condition and the KKT conditions say the same thing. Write $\nabla_i f(x) = \partial f(x) / \partial x_i$ for the i th component of the gradient.

Example 8.8 (The nonnegative orthant). Consider $\min f(x)$ subject to $x \geq 0$.

KKT language. Write the constraints as $g_i(x) = -x_i \leq 0$, so $\nabla g_i = -e_i$ (minus the i th unit vector) and there are no equalities. Stationarity in (8.4) reads $\nabla f(x^*) - \lambda^* = 0$, so $\lambda^* = \nabla f(x^*)$, and the other three blocks become

$$x^* \geq 0, \quad \nabla f(x^*) \geq 0, \quad x_i^* \nabla_i f(x^*) = 0 \quad \text{for all } i. \quad (8.7)$$

Normal-cone language. For $x \geq 0$,

$$N_{\mathbb{R}_+^d}(x) = \{v \in \mathbb{R}^d : v_i = 0 \text{ if } x_i > 0, \quad v_i \leq 0 \text{ if } x_i = 0\}.$$

Indeed, such a v has $v^\top (y - x) = \sum_{i: x_i=0} v_i y_i \leq 0$ for all $y \geq 0$; conversely, testing the definition with $y = x + e_i$ gives $v_i \leq 0$, and with $y = x - x_i e_i$ (when $x_i > 0$) gives $-v_i x_i \leq 0$, so $v_i = 0$. Now $-\nabla f(x^*) \in N_{\mathbb{R}_+^d}(x^*)$ says $\nabla_i f(x^*) = 0$ where $x_i^* > 0$ and $\nabla_i f(x^*) \geq 0$ where $x_i^* = 0$, which with $x^* \geq 0$ is exactly (8.7).

Reading. A positive coordinate is free to move both ways, so its partial derivative must vanish. A coordinate at zero may have a positive partial derivative: decreasing x_i would lower f , but the constraint forbids it. This is complementary slackness for the orthant, and for NNLS (Model 8.1) it holds with $\nabla f(x^*) = A^\top (Ax^* - b)$.

Example 8.9 (The simplex). Let $\Delta = \{x \in \mathbb{R}^d : x \geq 0, \mathbf{1}^\top x = 1\}$ and consider $\min f(x)$ subject to $x \in \Delta$. (The plain symbol Δ is unrelated to the squared distance Δ_k of [Week 7](#).) With one multiplier $\nu \in \mathbb{R}$ for the equality and $\lambda \in \mathbb{R}^d$ for $-x_i \leq 0$, the KKT conditions are $x^* \in \Delta$ and

$$\nabla f(x^*) + \nu \mathbf{1} - \lambda = 0, \quad \lambda \geq 0, \quad \lambda_i x_i^* = 0 \text{ for all } i. \quad (8.8)$$

The normal-cone form compresses this to $0 \in \nabla f(x^*) + N_\Delta(x^*)$, and the two agree because

$$N_\Delta(x) = \{\nu \mathbf{1} - \lambda : \nu \in \mathbb{R}, \lambda \geq 0, \lambda_i x_i = 0 \text{ for all } i\}, \quad x \in \Delta. \quad (8.9)$$

Indeed, $v \in N_\Delta(x)$ means $v^\top y \leq v^\top x$ for all $y \in \Delta$, that is, x minimises the linear function $-v^\top y$ over Δ . Slater's condition holds for this problem (at $y = \mathbf{1}/d$), so by [Theorem 5.15](#) its solutions are exactly its KKT points: $-v + \nu \mathbf{1} - \lambda = 0$ for some $\nu \in \mathbb{R}$ and $\lambda \geq 0$ with $\lambda_i x_i = 0$ for all i .

Reading. Equality multipliers and active inequality multipliers combine into one geometric object. Where $x_i^* > 0$ we have $\lambda_i = 0$ and so $\nabla_i f(x^*) = -\nu$; where $x_i^* = 0$, $\nabla_i f(x^*) = -\nu + \lambda_i \geq -\nu$. All options that receive a positive share have the *same* partial derivative and the options left at zero have one at least as large: shifting budget between held assets changes nothing to first order, and shifting it to an unheld asset cannot help.

8.3 Projection as a computable primitive

8.3.1 Definition: the nearest feasible point

What is the simplest optimisation problem attached to a set C ? Arguably this one: given a point y , find the feasible point nearest to it. In this subsection y is the (possibly infeasible) point being projected, z is its projection, and x stands for a generic point of C .

Definition 8.10 (Euclidean projection). Let $C \subseteq \mathbb{R}^d$ be nonempty, closed and convex. The **projection** of $y \in \mathbb{R}^d$ onto C is

$$\Pi_C(y) = \operatorname{argmin}_{z \in C} \frac{1}{2} \|z - y\|_2^2 = \operatorname{argmin}_{z \in \mathbb{R}^d} \left\{ I_C(z) + \frac{1}{2} \|z - y\|_2^2 \right\}. \quad (8.10)$$

This is not the coordinate projection of [Proposition 2.20](#), which drops auxiliary variables; from now on “projection” always means this nearest-point map.

The three hypotheses make $\Pi_C(y)$ a well-defined single point. A nearest point *exists* because C is nonempty and closed: the continuous function $\frac{1}{2} \|z - y\|_2^2$ attains its minimum over the closed bounded set of feasible points no farther from y than some fixed $z_0 \in C$ (for the open interval $C = (0, 1)$ and $y = 2$ there is none). It is *unique* because the objective is strictly convex and C is convex ([Proposition 4.2](#)); for the set $C = \{-1, 1\}$ and $y = 0$ there are two. If $y \in C$ then $\Pi_C(y) = y$.

In the second form of (8.10) the indicator enforces feasibility and the quadratic asks for closeness. Minimising “a function plus $\frac{1}{2} \|z - y\|_2^2$ ” is what [Week 9](#) calls a *proximal operator*; projection is the proximal operator of the indicator, and this is the bridge to next week.

Note on the lecture slides

The slides never state hypotheses on C : “convex” appears once (frame 8.2), “closed” and “nonempty” never, and frame 8.10 places hypotheses on f only. They are needed for the projection to exist and be unique, and without convexity both [Theorem 8.7](#) and [Proposition 8.12](#) can fail: on $C = \{-1, 1\}$ the minimiser $x^* = 1$ of $f(x) = (x - 0.4)^2$ has $\nabla f(1)(y - 1) = -2.4 < 0$ at $y = -1$, and the points 0.1 and -0.1 project to 1 and -1 . The notes assume that C is nonempty, closed and convex, as every set on the slides is.

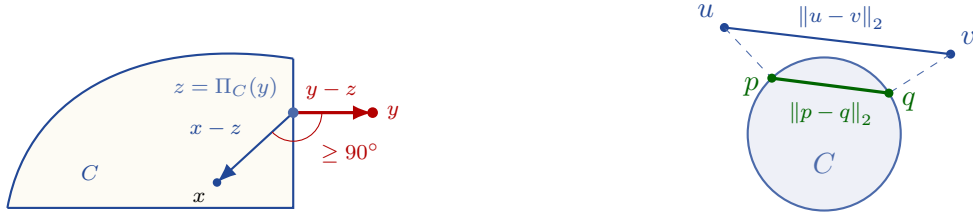


Figure 8.3: Left: the projection theorem. The residual $y - z$ makes an angle of at least 90° with the direction $x - z$ to any feasible point x , so $y - z \in N_C(z)$. Right: non-expansiveness. The projections $p = \Pi_C(u)$ and $q = \Pi_C(v)$ are closer together than u and v are.

8.3.2 The projection theorem

Projection is itself a smooth convex problem over C , so [Theorem 8.7](#) characterises its solution.

Theorem 8.11 (Projection theorem). *For C nonempty, closed and convex and $y \in \mathbb{R}^d$,*

$$\begin{aligned} z = \Pi_C(y) &\iff z \in C \text{ and } (y - z)^\top (x - z) \leq 0 \text{ for all } x \in C \\ &\iff y - z \in N_C(z) \iff 0 \in z - y + N_C(z). \end{aligned}$$

Proof. Apply [Theorem 8.7](#) to $f(z) = \frac{1}{2} \|z - y\|_2^2$, whose gradient is $\nabla f(z) = z - y$. The variational inequality (8.6) reads $(z - y)^\top (x - z) \geq 0$ for all $x \in C$, which is the first equivalence. The second is [Definition 8.5](#) with $v = y - z$, and the third rearranges it. $\square$

The residual $y - z$ behaves like an outward normal at the projected point ([Figure 8.3](#), left): it makes an angle of at least 90° with every direction from z into C . The last form, $0 \in z - y + N_C(z)$, has the same pattern as the optimality condition $0 \in \nabla f(x^*) + N_C(x^*)$. That is why projection and normal cones fit together, and [Proposition 8.15](#) will exploit it.

8.3.3 Projection is non-expansive

The convergence analysis needs one analytical fact about projection: it never increases distances.

Proposition 8.12 (Non-expansiveness). *Let C be nonempty, closed and convex. For all $u, v \in \mathbb{R}^d$,*

$$\|\Pi_C(u) - \Pi_C(v)\|_2 \leq \|u - v\|_2.$$

Proof. Let $p = \Pi_C(u)$ and $q = \Pi_C(v)$ (here p is a point, not the number of equality constraints). Apply [Theorem 8.11](#) at u with the feasible point $x = q$, and at v with $x = p$:

$$(u - p)^\top (q - p) \leq 0, \quad (v - q)^\top (p - q) \leq 0.$$

Adding them, with the second written as $(q - v)^\top (q - p) \leq 0$, gives $((u - v) - (p - q))^\top (q - p) \leq 0$, or

$$(p - q)^\top (u - v) \geq \|p - q\|_2^2. \quad (8.11)$$

The left-hand side is at most $\|p - q\|_2 \|u - v\|_2$ by the Cauchy–Schwarz inequality ([Proposition 1.11](#)). If $p \neq q$, divide by $\|p - q\|_2$; if $p = q$ there is nothing to prove. $\square$

Projection never magnifies a perturbation ([Figure 8.3](#), right): an error in the point being projected, from an earlier iteration or, in [Week 11](#), from a noisy gradient, is not amplified by the feasibility step. This is what lets the analysis of the gradient step survive the projection in [Theorem 8.18](#); inequality (8.11) returns in [Week 9](#) for proximal operators.

8.3.4 Projections you can write down

We now compute the projection for each simple set. The recipe is always the same: guess or derive z , then confirm it with [Theorem 8.11](#) or the KKT conditions of [\(8.10\)](#).

Intervals, boxes and the orthant. For the box $\{x : \ell \leq x \leq u\}$ with $\ell, u \in \mathbb{R}^d$, written $[\ell, u]$ for short (these bounds have nothing to do with the points u, v of the last proof), both the objective $\frac{1}{2} \sum_i (z_i - y_i)^2$ and the constraints $\ell_i \leq z_i \leq u_i$ separate across coordinates, so each coordinate solves its own one-dimensional problem: the point of $[\ell_i, u_i]$ nearest to y_i is y_i itself if it lies in the interval and otherwise the nearer endpoint. Hence

$$\Pi_{[\ell, u]}(y)_i = \min\{\max\{y_i, \ell_i\}, u_i\}, \quad \Pi_{\mathbb{R}_+^d}(y)_i = \max\{y_i, 0\}, \quad (8.12)$$

the second being the case $\ell = 0$ with no upper bound, also written $(y)_+$. These are coordinate-wise *clipping* rules (`np.clip`); for example, $y = (1.7, -0.4, 0.3)$ projects onto $[0, 1]^3$ as $(1, 0, 0.3)$.

Euclidean ball. For $B(r) = \{x : \|x\|_2 \leq r\}$ with $r > 0$ ([Definition 2.14](#), centred at the origin),

$$\Pi_{B(r)}(y) = \min\left\{1, \frac{r}{\|y\|_2}\right\} y, \quad (8.13)$$

read as y when $y = 0$. A point inside the ball is left alone, and a point outside is rescaled radially back to the boundary. To confirm the second case, let $\|y\|_2 > r$ and $z = ry / \|y\|_2$; then $y - z = (\|y\|_2 / r - 1) z$ is a positive multiple of z , and for every x in the ball $z^\top(x - z) \leq \|z\|_2 \|x\|_2 - r^2 \leq 0$ by Cauchy–Schwarz, so the projection theorem certifies z . With $r = 1$ the point $y = (3, 4)$ has norm 5 and projects to $(0.6, 0.8)$.

Hyperplane and affine set. Let $S = \{x : Ax = b\}$ ([Definition 2.8](#)), where now A and b are constraint data, not the least-squares data of [Model 8.1](#), and the rows of A are linearly independent (no equality is redundant), so that $AA^\top$ is invertible ([Proposition 1.13](#) applied to $A^\top$). The projection problem has equality constraints only, so its KKT conditions are stationarity, $z - y + A^\top \nu = 0$, and feasibility, $Az = b$; substituting $z = y - A^\top \nu$ into $Az = b$ gives $\nu = (AA^\top)^{-1}(Ay - b)$ and

$$\Pi_S(y) = y - A^\top (AA^\top)^{-1}(Ay - b), \quad \Pi_H(y) = y - \frac{a^\top y - b}{\|a\|_2^2} a, \quad (8.14)$$

where the second formula is the case of a single row $A = a^\top$, $a \neq 0$: the hyperplane $H = \{x : a^\top x = b\}$ of [Definition 2.10](#). Both subtract the infeasibility, $Ay - b$ or $a^\top y - b$, along the constraint normals, the equality-only geometry of [Section 5.2](#); the example solved there by tangency, the point of the line $x_1 + x_2 = 2$ nearest to $(3, 1)$, now takes one line: $(3, 1) - \frac{4-2}{2}(1, 1) = (2, 0)$.

Simplex. Here $\mathbf{1}^\top x = 1$ couples the coordinates, so clipping alone is not enough.

Proposition 8.13 (Projection onto the simplex). *For the simplex $\Delta = \{x \in \mathbb{R}^d : x \geq 0, \mathbf{1}^\top x = 1\}$,*

$$\Pi_\Delta(y)_i = \max\{y_i - \tau, 0\}, \quad i = 1, \dots, d, \quad y \in \mathbb{R}^d, \quad (8.15)$$

where the threshold $\tau \in \mathbb{R}$ is the unique solution of $\sum_{i=1}^d \max\{y_i - \tau, 0\} = 1$.

Proof. By [Example 8.9](#) applied to $f(x) = \frac{1}{2} \|x - y\|_2^2$, x is the projection exactly when $x \in \Delta$ and $x - y + \nu \mathbf{1} - \lambda = 0$ with $\lambda \geq 0$, $\lambda_i x_i = 0$. If $x_i > 0$ then $\lambda_i = 0$ and $x_i = y_i - \nu$. If $x_i = 0$ then $y_i - \nu = -\lambda_i \leq 0$. In both cases $x_i = \max\{y_i - \nu, 0\}$, so $\tau = \nu$, and $\mathbf{1}^\top x = 1$ is the equation for τ . Its left-hand side $\sum_i \max\{y_i - \tau, 0\}$ is continuous in τ , tends to $+\infty$ as $\tau \rightarrow -\infty$, vanishes for $\tau \geq \max_i y_i$ and is strictly decreasing where it is positive, so it equals 1 exactly once. $\square$

The threshold τ is the multiplier of the budget constraint: every coordinate is lowered by the same amount, those that would become negative are set to zero, and τ is tuned so that what remains sums to one. If the projection has exactly j positive coordinates, they must be the j

Table 8.1: Closed-form projections onto simple sets and their cost in dimension d (p rows of A , with $AA^\top$ factorised once; τ from Algorithm 8.1).

Set	Projection of y	Main idea	Cost
interval, box $\{\ell \leq x \leq u\}$	$\min\{\max\{y_i, \ell_i\}, u_i\}$	clip to the bounds	$O(d)$
orthant $\mathbb{R}_+^d$	$(y)_+ = \max\{y, 0\}$, coordinate-wise	zero out negatives	$O(d)$
ball $\{\ x\ _2 \leq r\}$	$\min\{1, r/\ y\ _2\} y$	radial rescaling	$O(d)$
hyperplane $\{a^\top x = b\}$	$y - ((a^\top y - b)/\ a\ _2^2) a$	orthogonal correction	$O(d)$
affine set $\{Ax = b\}$	$y - A^\top(AA^\top)^{-1}(Ay - b)$	correct along row normals	$O(pd)$
simplex Δ	$\max\{y_i - \tau, 0\}$	shift by τ , then clip at zero	$O(d \log d)$

largest entries of y , and $\sum(y_i - \tau) = 1$ over those entries fixes τ ; Algorithm 8.1 tries $j = 1, 2, \dots$ in sorted order and keeps the largest j for which the candidate threshold still lies below the j th largest entry. The sort dominates the cost, $O(d \log d)$.

Algorithm 8.1 Projection onto the simplex by sorting

Require: $y \in \mathbb{R}^d$ **Ensure:** $\Pi_\Delta(y)$

- 1: sort the entries of y in decreasing order: $s_1 \geq s_2 \geq \dots \geq s_d$
 - 2: $\rho \leftarrow \max\{j \in \{1, \dots, d\} : s_j - \frac{1}{j}(\sum_{i=1}^j s_i - 1) > 0\}$
 - 3: $\tau \leftarrow \frac{1}{\rho}(\sum_{i=1}^\rho s_i - 1)$
 - 4: **return** the vector with entries $\max\{y_i - \tau, 0\}$
-

Example 8.14 (A simplex projection by hand). Project $y = (0.6, 0.9, -0.1)$ onto $\Delta \subset \mathbb{R}^3$. Sorted, $s = (0.9, 0.6, -0.1)$. The tests of Algorithm 8.1 are: $j = 1$: $0.9 - (0.9 - 1) = 1 > 0$; $j = 2$: $0.6 - \frac{1}{2}(1.5 - 1) = 0.35 > 0$; $j = 3$: $-0.1 - \frac{1}{3}(1.4 - 1) < 0$. So $\rho = 2$, $\tau = \frac{1}{2}(1.5 - 1) = 0.25$ and $z = \Pi_\Delta(y) = (\max\{0.35, 0\}, \max\{0.65, 0\}, \max\{-0.35, 0\}) = (0.35, 0.65, 0)$. The projection theorem certifies it: the residual is $y - z = (0.25, 0.25, -0.1)$ and, for any $x \in \Delta$, using $x_1 + x_2 = 1 - x_3$, we get $(y - z)^\top(x - z) = 0.25(1 - x_3) - 0.1x_3 - 0.25(0.35 + 0.65) = -0.35x_3 \leq 0$. The residual also has the form (8.9), with $\nu = \tau = 0.25$ and $\lambda = (0, 0, 0.35)$.

Table 8.1 collects the projections and their costs: $O(d)$ arithmetic operations each, except the simplex (a sort) and the general affine set (a linear solve with the small matrix $AA^\top$). If you can write the projection, you can usually write the constrained first-order algorithm; a general polyhedron $\{x : Ax \leq b\}$ has no closed-form projection, the subject of Week 13.

8.4 The projected-gradient method

8.4.1 Optimality as a fixed point

The optimality condition and the projection theorem have the same shape. Putting them together turns optimality into a statement about one gradient step followed by one projection.

Proposition 8.15 (Fixed-point form of optimality). *Let f be convex and differentiable and C nonempty, closed and convex. For every $\alpha > 0$,*

$$x^* \text{ solves } \min_{x \in C} f(x) \iff x^* = \Pi_C(x^* - \alpha \nabla f(x^*)).$$

Proof. By Theorem 8.7, x^* is optimal exactly when $-\nabla f(x^*) \in N_C(x^*)$. The normal cone is closed under multiplication by positive numbers, so this holds exactly when $-\alpha \nabla f(x^*) \in N_C(x^*)$, that is, when $(x^* - \alpha \nabla f(x^*)) - x^* \in N_C(x^*)$. This is the condition $y - z \in N_C(z)$ of Theorem 8.11 with $y = x^* - \alpha \nabla f(x^*)$ and $z = x^*$, so it says $x^* = \Pi_C(y)$. $\square$

At a solution a gradient step may well leave C , but the projection brings it back exactly to where it started. The fixed-point form matters for three reasons: it is the constrained analogue of the stationarity test $\nabla f(x^*) = 0$ of [Week 6](#), to which it reduces when $C = \mathbb{R}^d$; it can be checked with the two operations we can compute, a gradient and a projection, which gives a practical residual; and it suggests the algorithm at once: iterate the right-hand side.

8.4.2 The projected-gradient mapping

Definition 8.16 (Projected-gradient mapping). For $\alpha > 0$, the **projected-gradient mapping** is

$$G_\alpha(x) = \frac{1}{\alpha} \left(x - \Pi_C(x - \alpha \nabla f(x)) \right).$$

By [Proposition 8.15](#) and [Theorem 8.7](#), for $x \in C$,

$$G_\alpha(x) = 0 \iff x = \Pi_C(x - \alpha \nabla f(x)) \iff 0 \in \nabla f(x) + N_C(x), \quad (8.16)$$

so G_α vanishes exactly at the solutions, whatever the value of α , and plays the role of the gradient in the constrained problem. When $C = \mathbb{R}^d$ the projection is the identity and $G_\alpha = \nabla f$; when the solution lies on the boundary of C , $\nabla f(x^*)$ is typically nonzero while $G_\alpha(x^*) = 0$, because the mapping discards the part of the gradient that the constraint resists. In [Week 6](#) we monitored $\|\nabla f(x^k)\|_2$; this week the natural diagnostic is $\|G_\alpha(x^k)\|_2$. (The symbol G_α is unrelated to the bound G on subgradient norms of [Week 7](#).)

8.4.3 Deriving the update from the local model

In [Week 6](#) the gradient step minimised the stabilised local model (6.3): the tangent approximation of f at x^k plus a quadratic term that keeps the next point close. Keep the same model (without its constant $f(x^k)$, which does not affect the minimiser) and add the indicator:

$$x^{k+1} = \operatorname{argmin}_{z \in \mathbb{R}^d} \left\{ \nabla f(x^k)^\top (z - x^k) + \frac{1}{2\alpha_k} \|z - x^k\|_2^2 + I_C(z) \right\}. \quad (8.17)$$

Completing the square in the first two terms,

$$\nabla f(x^k)^\top (z - x^k) + \frac{1}{2\alpha_k} \|z - x^k\|_2^2 = \frac{1}{2\alpha_k} \|z - (x^k - \alpha_k \nabla f(x^k))\|_2^2 - \frac{\alpha_k}{2} \|\nabla f(x^k)\|_2^2.$$

The last term does not depend on z , and a positive factor $1/\alpha_k$ does not change a minimiser, so (8.17) is the projection problem (8.10) for $y = x^k - \alpha_k \nabla f(x^k)$:

$$x^{k+1} = \Pi_C(x^k - \alpha_k \nabla f(x^k)). \quad (8.18)$$

Step one moves in the unconstrained descent direction, step two restores feasibility by projection, and [Algorithm 8.2](#) is [Algorithm 6.1](#) with exactly one ingredient changed. In terms of the mapping, $x^{k+1} = x^k - \alpha_k G_{\alpha_k}(x^k)$: the method is gradient descent with G_{α_k} in place of ∇f , and the residual comes for free as $(x^k - x^{k+1})/\alpha_k$.

Each iteration costs one gradient, $O(Nd)$ for least squares, plus one projection, $O(d)$ or $O(d \log d)$ by [Table 8.1](#); the constraint adds almost nothing to the cost. Every iterate is feasible by construction, so the method can be stopped at any time with a usable decision.

8.4.4 Convergence: one extra inequality from projection

Fix a step $\alpha > 0$ and write the update as $x^{k+1} = T_\alpha(x^k)$ with the *projected-gradient map* (the update itself, not to be confused with the residual G_α)

$$T_\alpha(x) = \Pi_C(x - \alpha \nabla f(x)), \quad \text{so that } G_\alpha(x) = (x - T_\alpha(x))/\alpha.$$

Algorithm 8.2 Projected gradient

Require: convex differentiable f ; projection Π_C onto a simple set C ; starting point $x^0 \in C$; step-size rule; tolerance $\varepsilon > 0$; iteration cap K

Ensure: an approximate solution of $\min_{x \in C} f(x)$

```

1: for  $k = 0, 1, \dots, K - 1$  do
2:   choose  $\alpha_k > 0$  (fixed  $\alpha_k = \alpha \leq 1/L$ , or backtracking on (8.20))
3:    $y^k \leftarrow x^k - \alpha_k \nabla f(x^k)$  ▷ gradient step; may leave  $C$ 
4:    $x^{k+1} \leftarrow \Pi_C(y^k)$  ▷ return to  $C$ 
5:   if  $\|G_{\alpha_k}(x^k)\|_2 = \|x^k - x^{k+1}\|_2 / \alpha_k \leq \varepsilon$  then return  $x^{k+1}$ 
6: end for
7: return  $x^K$ 

```

If x^* is optimal then $x^* = T_\alpha(x^*)$ by [Proposition 8.15](#), and non-expansiveness ([Proposition 8.12](#), with $u = x^k - \alpha \nabla f(x^k)$ and $v = x^* - \alpha \nabla f(x^*)$) gives

$$\|x^{k+1} - x^*\|_2 = \|T_\alpha(x^k) - T_\alpha(x^*)\|_2 \leq \|(x^k - x^*) - \alpha(\nabla f(x^k) - \nabla f(x^*))\|_2. \quad (8.19)$$

The left-hand side is the error of the constrained iterate. The right-hand side contains no projection: it compares two plain gradient steps, and how much such a step shrinks the distance between two points depends only on the curvature, which [Definitions 6.8](#) and [6.12](#) cap above by L and below by μ .

Lemma 8.17 (Gradient-step contraction; quoted). *If f is μ -strongly convex and L -smooth, then*

$$\|(x - y) - \frac{1}{L}(\nabla f(x) - \nabla f(y))\|_2 \leq \left(1 - \frac{\mu}{L}\right) \|x - y\|_2 \quad \text{for all } x, y \in \mathbb{R}^d.$$

We quote this lemma without a general proof. The case that shows why it is true is a separable quadratic $f(x) = \frac{1}{2} \sum_i h_i x_i^2 - c^\top x$ with $\mu \leq h_i \leq L$: there $\nabla_i f(x) = h_i x_i - c_i$, so the i th component of the vector on the left is $(1 - h_i/L)(x_i - y_i)$ with $0 \leq 1 - h_i/L \leq 1 - \mu/L$, the coordinate factors of [Example 6.14](#); a general quadratic needs the eigenvalue tools of [Week 10](#), and the companion notebook `week08_projected_gradient.ipynb` checks the inequality on random pairs of points. With a smaller step $\alpha < 1/L$ in place of $1/L$ the factor becomes $1 - \alpha\mu$ (coordinate factors $1 - \alpha h_i$): among the safe steps $\alpha \leq 1/L$ the largest gives the best guarantee.

Theorem 8.18 (Linear convergence of projected gradient). *Let f be μ -strongly convex and L -smooth, let C be nonempty, closed and convex, and let x^* be the solution of $\min_{x \in C} f(x)$. The iterates of [Algorithm 8.2](#) with the fixed step $\alpha_k = 1/L$ satisfy*

$$\|x^{k+1} - x^*\|_2 \leq \left(1 - \frac{\mu}{L}\right) \|x^k - x^*\|_2, \quad \text{hence} \quad \|x^k - x^*\|_2 \leq \left(1 - \frac{\mu}{L}\right)^k \|x^0 - x^*\|_2.$$

Proof. Combine (8.19) for $\alpha = 1/L$ with [Lemma 8.17](#) for $x = x^k$, $y = x^*$, and iterate. (A solution exists because C is closed and f is strongly convex, and it is unique by strict convexity.) $\square$

Note on the lecture slides

The slides say that the “Week 6 convergence logic” is recycled (frames 8.2, 8.19, 8.31–8.32). What is inherited from [Week 6](#) is the pair of assumptions, the step $1/L$ and the factor $1 - \mu/L$. The statement is different: [Theorem 6.13](#) bounds the objective gap $f(x^k) - f^*$, whereas [Theorem 8.18](#) bounds the distance $\|x^k - x^*\|_2$, and it rests on [Lemma 8.17](#), which does not appear in [Week 6](#); frame 8.32 quotes it and the week’s notebook checks it numerically.

Since $1 - s \leq e^{-s}$, the distance after k iterations is at most $e^{-k\mu/L} \|x^0 - x^*\|_2$, which is below ε once $k \geq \frac{L}{\mu} \ln \frac{\|x^0 - x^*\|_2}{\varepsilon}$. The count is governed by L/μ (the condition number of [Week 10](#)),

as in [Week 6](#): an easy feasible set does not remove the curvature difficulty of the objective. If f is only convex and L -smooth ($\mu = 0$) the linear rate is lost, and the guarantee degrades to $f(x^k) - f^* = O(1/k)$, the same boundary as in [Theorem 6.11](#); this is the case $h = I_C$ of [Theorem 9.18](#), proved next week, which gives $f(x^k) - f^* \leq \|x^0 - x^*\|_2^2 / (2\alpha k)$ for any fixed step $0 < \alpha \leq 1/L$. The new benefit, in both regimes, is that every iterate is feasible.

Key idea

Projected gradient inherits the convergence behaviour of gradient descent because projection is non-expansive: the feasibility step never moves the iterate farther from the solution.

8.4.5 Step sizes and stopping in practice

The step-size ideas of [Week 6](#) still apply: projection changes the iterate, but the step size still controls stability and speed. A constant step $\alpha \leq 1/L$ is the simplest to implement and gives the cleanest theory, but it needs L (for a quadratic objective, the largest directional curvature of its Hessian) or some tuning. Backtracking is more robust and more expensive per iteration, but the test of [Algorithm 6.2](#) (whose parameter α is not this week's step) cannot be reused as it stands: it asks for a decrease proportional to $\|\nabla f(x^k)\|_2^2$, which is unattainable near a solution on the boundary, where the gradient is not small but the projected step hardly moves. Instead, from a trial step α compute $x^+ = \Pi_C(x^k - \alpha \nabla f(x^k))$ and halve α until

$$f(x^+) \leq f(x^k) + \nabla f(x^k)^\top (x^+ - x^k) + \frac{1}{2\alpha} \|x^+ - x^k\|_2^2. \quad (8.20)$$

For an L -smooth f this holds for every $\alpha \leq 1/L$ ([Proposition 6.9](#)), so the loop terminates; each trial costs a projection and a function value, and the same test returns in [Week 9](#). The test also guarantees descent: the projection theorem at the feasible point $x = x^k$ gives $\nabla f(x^k)^\top (x^+ - x^k) \leq -\|x^+ - x^k\|_2^2 / \alpha$, so every accepted step, and every fixed step $\alpha \leq 1/L$, has $f(x^+) \leq f(x^k) - \frac{\alpha}{2} \|G_\alpha(x^k)\|_2^2$. In a run we therefore track the objective $f(x^k)$, which must decrease; feasibility, which the projection guarantees up to rounding; and the residual $\|G_\alpha(x^k)\|_2$, the stopping test, which depends on α and should be compared only between runs with the same step.

8.5 Worked examples and extensions

8.5.1 A box-constrained quadratic, step by step

Example 8.19 (Projected gradient on a box). Take the quadratic $f(x) = 2x_1^2 + x_2^2 - 4x_1 - 2x_2 + 5$ of [Weeks 5](#) and [6](#), whose unconstrained minimiser is $(1, 1)$, and minimise it over the box $C = [0, 2] \times [0, 0.6]$, which excludes $(1, 1)$. The gradient is $\nabla f(x) = (4x_1 - 4, 2x_2 - 2)$ and the Hessian is $\text{diag}(4, 2)$, so $L = 4$ and $\mu = 2$. We run [Algorithm 8.2](#) from $x^0 = (1.9, 0.05)$ with $\alpha = 1/L = 0.25$; the projection is the clipping rule [\(8.12\)](#).

Iterations. From $\nabla f(x^0) = (3.6, -1.9)$ the trial point $y^0 = x^0 - 0.25(3.6, -1.9) = (1, 0.525)$ is feasible, so the projection does nothing, $x^1 = y^0$, the objective falls from $f(x^0) = 4.5225$ to $f(x^1) = 2.2256$, and $G_\alpha(x^0) = \nabla f(x^0)$. Next, $\nabla f(x^1) = (0, -0.95)$ gives $y^1 = (1, 0.7625)$, which violates $x_2 \leq 0.6$; clipping gives $x^2 = (1, 0.6)$ with $f(x^2) = 2.16$, and the residual $G_\alpha(x^1) = (x^1 - x^2)/0.25 = (0, -0.3)$ is shorter than the gradient because part of the step was cut off by the constraint. Then $\nabla f(x^2) = (0, -0.8)$ and $y^2 = (1, 0.8)$, again infeasible, and clipping returns $x^3 = (1, 0.6) = x^2$: the iterate is a fixed point, $G_\alpha(x^2) = 0$, and by [Proposition 8.15](#) it is the solution, $x^* = (1, 0.6)$ with $f^* = 2.16$.

Certificate. The gradient $\nabla f(x^*) = (0, -0.8)$ is not zero, but on the top edge of the box the normal cone is $N_C(x^*) = \{(0, t) : t \geq 0\}$, which contains $-\nabla f(x^*) = (0, 0.8)$; the variational inequality reads $-0.8(y_2 - 0.6) \geq 0$, true for every feasible y since $y_2 \leq 0.6$, and in KKT language 0.8 is the multiplier of $x_2 \leq 0.6$. [Figure 8.4](#) (left) shows the path.

8.5.2 Nonnegative least squares

For [Model 8.1](#), $\nabla f(x) = A^\top(Ax - b)$ and the projection onto $\mathbb{R}_+^d$ is $\max\{\cdot, 0\}$, so (8.18) becomes

$$x^{k+1} = \max\{x^k - \alpha_k A^\top(Ax^k - b), 0\} \quad \text{coordinate-wise :} \quad (8.21)$$

gradient descent followed by clipping at zero. As in [Model 6.16](#), the Hessian is $A^\top A$, so L and μ are the largest and smallest values of the directional curvature $u^\top A^\top A u = \|Au\|_2^2$ over unit vectors u (with $\mu > 0$ when the columns of A are linearly independent). These are the largest and smallest *eigenvalues* of $A^\top A$, a notion developed in [Week 10](#); in code, `np.linalg.eigvalsh`.

Example 8.20 (Projected gradient on a tiny NNLS problem). Let $A = \text{diag}(1, 2)$ and $b = (1, -2)$, so that $f(x) = \frac{1}{2}(x_1 - 1)^2 + \frac{1}{2}(2x_2 + 2)^2$, $\nabla f(x) = (x_1 - 1, 4x_2 + 4)$, $L = 4$ and $\mu = 1$, and take $\alpha = \frac{1}{4}$, $x^0 = (0, 1)$. Then $\nabla f(x^0) = (-1, 8)$, $y^0 = (0.25, -1)$ and $x^1 = (0.25, 0)$: the second coordinate is clipped and stays at zero from then on, while the first obeys $x_1^{k+1} - 1 = \frac{3}{4}(x_1^k - 1)$. The iterates converge to $x^* = (1, 0)$ with the factor $1 - \mu/L = \frac{3}{4}$ attained exactly, so the bound of [Theorem 8.18](#) cannot be improved in general. At the solution $\nabla f(x^*) = (0, 4)$ satisfies (8.7), being zero where $x_1^* > 0$ and positive where $x_2^* = 0$, and $G_\alpha(x^*) = 0$: the constraint $x_2 \geq 0$ is active, and the residual, not the gradient, reports optimality.

8.5.3 The simplex-constrained quadratic

For [Model 8.2](#), $\nabla f(x) = \Sigma x - c$ and

$$x^{k+1} = \Pi_\Delta(x^k - \alpha_k(\Sigma x^k - c)). \quad (8.22)$$

The objective is smooth and familiar; the algorithmic work is entirely in the simplex projection.

Example 8.21 (Projected gradient on the simplex). Take $\Sigma = \text{diag}(2, 1, 1)$ and $c = (1, 1, -1)$, so that $L = 2$ and $\mu = 1$, the step $\alpha = \frac{1}{2}$, and start from equal shares $x^0 = (\frac{1}{3}, \frac{1}{3}, \frac{1}{3})$ with $f(x^0) = -\frac{1}{9}$. Then $\nabla f(x^0) = (-\frac{1}{3}, -\frac{2}{3}, \frac{4}{3})$ and $y^0 = (\frac{1}{2}, \frac{2}{3}, -\frac{1}{3})$. [Algorithm 8.1](#) sorts to $(\frac{2}{3}, \frac{1}{2}, -\frac{1}{3})$, accepts $j = 1$ and $j = 2$ ($\frac{1}{2} - \frac{1}{2}(\frac{7}{6} - 1) = \frac{5}{12} > 0$) and rejects $j = 3$, so $\tau = \frac{1}{12}$ and $x^1 = (\frac{5}{12}, \frac{7}{12}, 0)$ with $f(x^1) = -0.65625$. The third option has left the portfolio after one step. By [Theorem 8.18](#) the iterates converge, to $x^* = (\frac{1}{3}, \frac{2}{3}, 0)$ with $f^* = -\frac{2}{3}$, where $\nabla f(x^*) = (-\frac{1}{3}, -\frac{1}{3}, 1)$: in (8.8), $\nu = \frac{1}{3}$ and $\lambda = (0, 0, \frac{4}{3})$. The two held options share the partial derivative $-\frac{1}{3}$ and the excluded one has a larger value, exactly the pattern of [Example 8.9](#).

8.5.4 Two implementation lanes

As in [Weeks 6](#) and [7](#) we solve the same model twice. In the *RSOME lane* ([Section 4.5](#)) the constraints are stated declaratively and a solver returns a benchmark solution and a status; in the *NumPy lane* we code the projection and the loop ourselves and obtain the iterates, the objective history and the residual $\|G_\alpha(x^k)\|_2$. [Listing 8.1](#) runs both lanes on an NNLS instance with $N = 60$, $d = 25$.

The function `projected_gradient` is [Algorithm 8.2](#) with a fixed step and takes the projection as an argument, so the same loop serves all three models of [Section 8.1](#) (`np.clip` for a box; [Algorithm 8.1](#) is a few lines of `np.sort` and `np.cumsum`, left to the exercises). The run prints

```
L = 142.3792, mu = 7.2211, 1 - mu/L = 0.949283
490 iterations, residual 9.64e-11, objective 14.09059699
status 2 objective 14.09059903 max |x_pg - x_rsome| = 2.37e-05
```

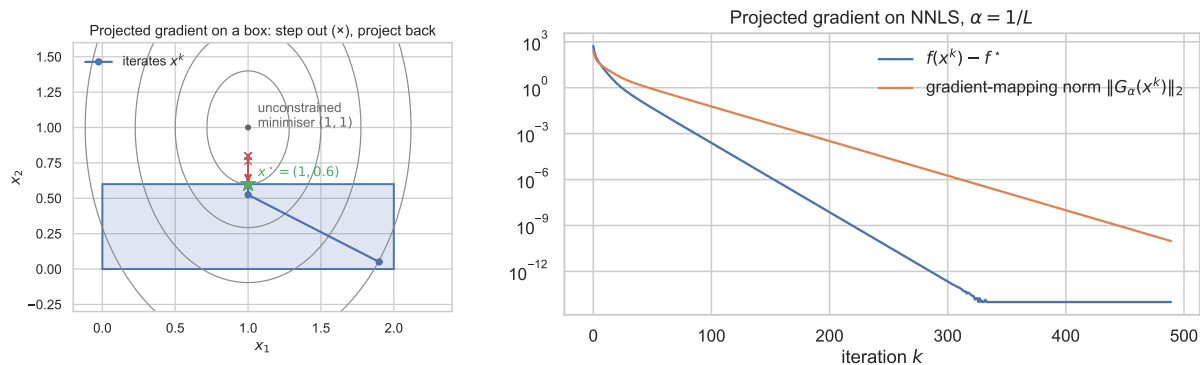


Figure 8.4: Left: the measured path of [Example 8.19](#); the crosses are the infeasible trial points $(1, 0.7625)$ and $(1, 0.8)$, projected back to the top edge of the box, and the star is $x^* = (1, 0.6)$. Right: the NNLS run of [Listing 8.1](#) with $\alpha = 1/L$: objective gap $f(x^k) - f^*$ (floored at 10^{-14}) and residual $\|G_\alpha(x^k)\|_2$.

Gurobi reports status 2 (optimal), the two lanes agree to about 10^{-5} , the solver’s tolerance, and SciPy’s `nnls`, an external reference, agrees with the projected-gradient solution to about 7×10^{-12} . In the right panel of [Figure 8.4](#) the objective gap and the residual are straight lines on a logarithmic scale, the signature of linear convergence. The measured distance ratio $\|x^{k+1} - x^*\|_2 / \|x^k - x^*\|_2$ settles at $0.9493 = 1 - \mu/L$, so the bound of [Theorem 8.18](#) is attained; the theorem promises a 10^{-6} reduction of the distance within $(L/\mu) \ln 10^6 \approx 272$ iterations, and the run needs 241. One caution: here the NNLS solution is strictly positive (it equals the unconstrained least-squares solution) and clipping occurs only in the first 22 iterations: the run shows feasibility being repaired early, not a constraint active at the solution (for that, see [Examples 8.19](#) and [8.20](#)).

8.5.5 Extension: projected subgradient

If the objective is convex but not differentiable, replace the gradient by a subgradient as in [Algorithm 7.1](#):

$$x^{k+1} = \Pi_C(x^k - \alpha_k g^k), \quad g^k \in \partial f(x^k). \quad (8.23)$$

This is the subgradient step of [Week 7](#) plus this week’s projection, and the analysis of [Section 7.4](#) carries over in one line: a solution $x^* \in C$ satisfies $x^* = \Pi_C(x^*)$, so by [Proposition 8.12](#) $\|x^{k+1} - x^*\|_2 \leq \|x^k - \alpha_k g^k - x^*\|_2$, after which the one-step inequality of [Lemma 7.13](#) follows exactly as before (its proof used only the subgradient inequality at $y = x^*$, so x^* may be the solution over C and f^* the optimal value over C ; G still bounds the subgradients along the iterates, not the mapping G_α). The same diminishing step-size rules and best-iterate bounds ([Theorem 7.14](#) and [Corollary 7.15](#)) therefore hold for the projected iterates, with the same $O(1/\sqrt{k})$ rate and non-descent behaviour. The method serves, for instance, for box-constrained least-absolute-deviations regression ([Model 7.6](#)) and constrained max-affine models ([Proposition 7.3](#)). What does not carry over is this week’s stopping test: with a subgradient in place of the gradient, $(x^k - x^{k+1})/\alpha_k$ need not become small near a solution (on $|x|$ over $[-1, 1]$ it equals $\text{sign}(x^k)$ at every $x^k \neq 0$ from which the step stays inside C), so the method is stopped as in [Section 7.5](#), by the best-so-far value, a budget and a benchmark.

8.6 Summary and looking ahead

Folding the constraint into the objective as an indicator function, and computing its subdifferential, the normal cone, compresses the KKT conditions of [Week 5](#) into $0 \in \nabla f(x^*) + N_C(x^*)$, equivalently the variational inequality: no feasible direction points downhill. The one new computational

Listing 8.1: Projected gradient in NumPy and the RSOME benchmark lane on an NNLS instance.

```

import numpy as np
import rsome as rso
from rsome import ro, grb_solver as grb
def projected_gradient(grad, proj, x0, alpha, tol=1e-10, maxit=5000):
    x = proj(x0)
    for k in range(maxit):
        x_new = proj(x - alpha * grad(x))      # gradient step, then project
        G = (x - x_new) / alpha                # gradient mapping G_alpha(x^k)
        x = x_new
        if np.linalg.norm(G) <= tol:
            break
    return x, k + 1, np.linalg.norm(G)
rng = np.random.default_rng(0)
N, d = 60, 25
A = rng.normal(size=(N, d))
b = rng.normal(size=N) + A @ np.abs(rng.normal(size=d))
mu, L = np.linalg.eigvalsh(A.T @ A)[[0, -1]] # smallest and largest eigenvalue
x_pg, its, res = projected_gradient(lambda x: A.T @ (A @ x - b),
                                   lambda y: np.maximum(y, 0.0), np.zeros(d), 1.0 / L)
print(f"L = {L:.4f}, mu = {mu:.4f}, 1 - mu/L = {1 - mu / L:.6f}")
print(f"{its} iterations, residual {res:.2e}, "
      f"objective {0.5 * np.sum((A @ x_pg - b) ** 2):.8f}")
mod = ro.Model()                                # the declarative lane
xr = mod.dvar(d)
mod.min(0.5 * rso.sumsqr(A @ xr - b))
mod.st(xr >= 0)
mod.solve(grb, display=False)
print("status", mod.solution.status, f"objective {mod.get():.8f}",
      f"max |x_pg - x_rsomed| = {np.abs(x_pg - xr.get()).max():.2e}")

```

primitive, projection, is characterised by the same normal-cone pattern, is non-expansive, and has a closed form for the simple sets of Table 8.1. The two combine into the fixed-point form of optimality, the residual G_α and the method $x^{k+1} = \Pi_C(x^k - \alpha_k \nabla f(x^k))$. With a subgradient in place of the gradient the same projection gives the projected subgradient method, which keeps the step-size rules and $O(1/\sqrt{k})$ guarantee of Week 7.

What the method does is take a gradient step and project back. *Why* it works: the update minimises the local model of Week 6 over C , its fixed points are exactly the solutions, and non-expansiveness lets the contraction of the gradient step pass through the projection, giving the factor $1 - \mu/L$ per iteration. *When* it applies: smooth convex objectives over sets with a cheap projection. Its *limitations*: the speed is still governed by L/μ , which Week 10 improves by acceleration; each iteration needs a full gradient, which Week 11 replaces by a sampled one (projection, being non-expansive, does not amplify the sampling noise); and a set without a cheap projection, such as a general polyhedron, stalls the method, the starting point of Week 13.

Next week hard constraints give way to penalties: the indicator I_C is replaced by a general convex function h , such as the ℓ_1 norm that produces sparse solutions, and projection by the proximal operator. The fixed-point form of optimality, the residual G_α , non-expansiveness and the convergence argument all transfer when Π_C is replaced by $\text{prox}_{\alpha h}$ (and N_C by ∂h), and projected gradient becomes the case $h = I_C$ of proximal gradient.

8.7 Exercises

Exercise 8.1 (From a business description to a model). Write each situation as one of Models 8.1 to 8.3, naming the decision variable, the objective, the feasible set C and its projection: (a) a retailer regresses weekly sales on advertising spend in d media channels and requires every

response coefficient to be nonnegative; (b) a marketing budget is split into shares across d channels so as to minimise a risk-minus-return score $\frac{1}{2}x^\top \Sigma x - c^\top x$; (c) d prices are chosen to minimise a smooth convex cost f , each price lying between a regulatory floor and a cap.

Exercise 8.2 (KKT versus indicator-function optimality). For the NNLS problem (8.1): (a) write the KKT conditions of Definition 5.14; (b) write the condition $0 \in A^\top(Ax^* - b) + N_{\mathbb{R}_+^d}(x^*)$ explicitly; (c) explain, coordinate by coordinate, why the two descriptions are equivalent.

Exercise 8.3 (Projections by hand). Let $y = (2, -1, 0.5)$. Project y onto (a) the box $[0, 1]^3$, (b) the orthant $\mathbb{R}_+^3$ and (c) the hyperplane $\{x : x_1 + 2x_2 - x_3 = 1\}$; then project (d) 1.4 onto $[-1, 1]$, (e) $(3, -4, 0)$ onto the ball $\{x : \|x\|_2 \leq 2\}$ and (f) $(0.8, 0.6, 0.1)$ onto the simplex $\Delta \subset \mathbb{R}^3$ by Algorithm 8.1. For one of these, verify $(y - z)^\top(x - z) \leq 0$ for all $x \in C$.

Exercise 8.4 (Non-expansiveness and convergence). Let $p = \Pi_C(u)$ and $q = \Pi_C(v)$. (a) Use the projection theorem twice to show $(p - q)^\top(u - v) \geq \|p - q\|_2^2$; (b) conclude that $\|\Pi_C(u) - \Pi_C(v)\|_2 \leq \|u - v\|_2$; (c) with $T_\alpha(x) = \Pi_C(x - \alpha \nabla f(x))$ and $x^* = T_\alpha(x^*)$, show $\|T_\alpha(x) - x^*\|_2 \leq \|(x - x^*) - \alpha(\nabla f(x) - \nabla f(x^*))\|_2$; (d) explain in one sentence why this reduces the convergence of projected gradient to the analysis of a plain gradient step.

Exercise 8.5 (Derive and apply projected gradient). For NNLS, derive the update (8.21) from (8.18). Then take $A = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}$, $b = (2, -1)$, $\alpha = \frac{1}{3}$ and $x^0 = (1, 1)$: (a) perform two iterations by hand; (b) compute the projected-gradient mapping $G_\alpha(x^0)$ and $G_\alpha(x^1)$; (c) verify that $x^* = (\frac{1}{2}, 0)$ satisfies (8.7) and interpret a zero residual.

Exercise 8.6 (Simplex-constrained quadratic and coding lab). Consider Model 8.2 with $\Sigma = \begin{bmatrix} 2 & 0.5 & 0 \\ 0.5 & 1 & 0 \\ 0 & 0 & 1.5 \end{bmatrix}$ and $c = (1, 0.5, -0.5)$. (a) Write the gradient and the projected-gradient update. (b) Implement the simplex projection and projected gradient in NumPy and run them with $\alpha = 0.4$. (c) Solve the same instance with RSOME. (d) Compare objective value, feasibility and $\|G_\alpha(x^k)\|_2$ over the iterations, and check complementary slackness at the solution: which coordinates are zero, and what are the partial derivatives there? (e) Optional: replace the smooth objective by a nonsmooth one and carry out one projected-subgradient step (8.23).

Week 9 Regularisation, Proximal Operators and the Proximal Gradient Method

Can an objective that is a smooth part plus a simple penalty be minimised at the rate of a smooth one? Many prediction and decision models add a penalty to a smooth loss, and the most useful penalties, such as the ℓ_1 norm, have kinks. [Week 7](#) already tells us how to minimise such an objective: treat the whole of it as one nonsmooth convex function and run the subgradient method ([Algorithm 7.1](#)). The price was high: no guarantee of descent, a rate of only $O(1/\sqrt{k})$, and iterates that are small but never exactly zero. This week asks whether that price must be paid when only *part* of the objective is nonsmooth, and the answer is no, provided the nonsmooth part is simple in a sense made precise below.

Three earlier weeks supply the ingredients. [Week 6](#) gave gradient descent as the minimiser of a stabilised local model ([Equation \(6.3\)](#)), the notion of L -smoothness with its descent lemma, and the rates $O(1/k)$ and $(1 - \mu/L)^k$. [Week 7](#) gave subgradients, the subdifferential of the ℓ_1 norm and the certificate $0 \in \partial f(x^*)$. [Week 8](#) gave the indicator I_C of a constraint set, the projection Π_C , its non-expansiveness, and the projected-gradient method. This week adds a single new object, the *proximal operator*, which does for a soft penalty what projection does for a hard constraint. With it the gradient step of [Week 6](#) and the projected step of [Week 8](#) become two cases of one update, and every convergence result of [Week 8](#) carries over.

[Section 9.1](#) explains why penalties are added and why the subgradient method is the wrong tool for them; [Section 9.2](#) sets up the composite model $\min g + h$, its main instances and its optimality condition; [Section 9.3](#) defines the proximal operator, shows that projection is a special case, proves non-expansiveness and derives the closed forms we need; [Section 9.4](#) derives the proximal gradient method, its stopping residual and its step-size rules; and [Section 9.5](#) covers convergence, the role of the regularisation strength and implementation in NumPy with an RSOME benchmark.

Two conventions should be fixed at once. The step size is written α or α_k , as in [Weeks 7](#) and [8](#); [Week 6](#) wrote t_k for the same quantity. And the letter g now denotes the *smooth part of the objective*, the function called f in [Weeks 6](#) and [8](#); it is no longer a subgradient as in [Week 7](#), nor a constraint function g_i as in the KKT recall of [Week 8](#). When a subgradient needs a name this week it is called s .

9.1 Why penalise, and why not the subgradient method

9.1.1 Regularisation as a modelling choice

A *penalty*, or *regulariser*, is a term added to an objective to express a preference that the main criterion ignores. In prediction, pure fit over-reacts to noise: a least-squares fit with many features will use all of them to chase random fluctuations in the sample. In decisions, pure profit ignores the cost of implementation: a portfolio optimiser re-run each month will happily trade every position for a negligible gain. Four motives recur. *Shrinkage* pulls estimates towards zero to stabilise them against noise. *Sparsity* keeps only a small number of active features or decisions, which makes a model easier to interpret and cheaper to run. *Turnover control* discourages unnecessary movement away from the current decision. Finally, a penalty can improve *numerical stability*: it can make a badly conditioned problem better conditioned and its solution less sensitive to the data.

Three models, studied in detail in [Section 9.2](#), illustrate the motives. *Ridge* regression adds $\frac{\lambda}{2} \|\beta\|_2^2$ to the least-squares loss of a coefficient vector β and shrinks all coefficients smoothly. The *lasso* adds $\lambda \|\beta\|_1$ instead and can set some coefficients exactly to zero. A *turnover penalty* $\lambda \|x - x^{\text{old}}\|_1$, added to a portfolio objective, keeps the portfolio x near its current holdings x^{old} .

In each, the parameter $\lambda \geq 0$ is the **regularisation strength**: it sets the rate of exchange between fit (or profit) and simplicity. Regularisation is first a modelling decision about which solutions we prefer; the algorithmic payoff, which occupies most of this week, comes afterwards.

9.1.2 Penalised and constrained forms of one trade-off

A preference for simple solutions can be written in two ways. Let g be the loss and let r measure complexity (a norm of the coefficients, or the amount of trading). The *penalised* and the *constrained* formulations are

$$\min_x g(x) + \lambda r(x) \quad \text{and} \quad \begin{array}{ll} \min_x & g(x) \\ \text{s.t.} & r(x) \leq \tau. \end{array} \quad (9.1)$$

The constrained form fixes a budget τ for complexity or turnover; the penalised form folds the trade-off into the objective. They describe the same family of solutions.

Proposition 9.1 (Penalised solutions solve a constrained problem). *Let $g, r : \mathbb{R}^d \rightarrow \mathbb{R}$ and $\lambda \geq 0$. If x_λ minimises $g + \lambda r$, then x_λ solves the constrained problem in (9.1) with budget $\tau = r(x_\lambda)$.*

Proof. Let x satisfy $r(x) \leq \tau$. Optimality of x_λ gives $g(x) + \lambda r(x) \geq g(x_\lambda) + \lambda \tau$, hence $g(x) \geq g(x_\lambda) + \lambda(\tau - r(x)) \geq g(x_\lambda)$. $\square$

No convexity was needed. The converse needs it: when g and r are convex and the budget constraint can be strictly satisfied (Slater's condition, [Theorem 5.13](#)), a solution of the constrained problem minimises $g + \lambda r$ with λ the optimal multiplier of the constraint $r(x) \leq \tau$. The regularisation strength *is* the shadow price ([Definition 5.21](#)) of the complexity budget, which reconciles the two uses of the letter λ in this book. A small λ matches a generous budget, and fit dominates; a large λ matches a tight budget, and simplicity dominates. We work with the penalised form because it leads to this week's algorithms. The budget symbol τ is used only here; from [Section 9.3](#) on, τ denotes a threshold level.

9.1.3 Exploiting structure instead of a generic subgradient

Write $F = g + h$ for the whole objective, with g the smooth loss and h the penalty. The generic nonsmooth viewpoint of [Week 7](#) treats F as one convex function with kinks and steps along $-s^k$ for some $s^k \in \partial F(x^k)$. That works very generally, but it makes no distinction between loss and penalty and discards everything we know about g . The *composite* viewpoint keeps the two parts separate: it uses the gradient $\nabla g(x^k)$ for the smooth part, exactly as in [Week 6](#), and handles h through a correction map that depends only on h , the proximal operator of [Section 9.3](#). This uses more of the problem's structure, is faster in theory and in practice, and keeps the modelling interpretation of the two parts visible in the algorithm.

The resulting update is not a new philosophy but the third member of a family:

$$\underbrace{x^+ = x - \alpha \nabla g(x)}_{h=0: \text{Week 6}} \quad \underbrace{x^+ = \Pi_C(x - \alpha \nabla g(x))}_{h=I_C: \text{Week 8}} \quad \underbrace{x^+ = \text{prox}_{\alpha h}(x - \alpha \nabla g(x))}_{\text{soft penalty } h: \text{this week}} \quad (9.2)$$

Here x^+ denotes the next iterate. The only new ingredient is the correction map $\text{prox}_{\alpha h}$ for a soft penalty, the proximal operator defined in [Section 9.3](#).

Key idea

When the loss is smooth and the regulariser has a cheap proximal operator, we can do much better than treating the whole objective as generic nonsmooth. The week is about exploiting structure.

9.2 The composite model and its optimality condition

9.2.1 The template and its instances

All the models of [Section 9.1](#) share one shape, which we now fix.

Model 9.2 (Composite problem).

$$\min_{x \in \mathbb{R}^d} F(x) = g(x) + h(x), \quad (9.3)$$

where $g : \mathbb{R}^d \rightarrow \mathbb{R}$ is the *smooth convex loss* and $h : \mathbb{R}^d \rightarrow \mathbb{R} \cup \{+\infty\}$ is a *simple convex regulariser*.

The decision variable is $x \in \mathbb{R}^d$ and there are no explicit constraints. The loss g is assumed convex and differentiable with an L -Lipschitz gradient ([Definition 6.8](#)). The regulariser h is convex; it may have kinks, and it may take the value $+\infty$, so that a hard constraint $x \in C$ can be included as $h = I_C$ ([Definition 8.4](#)). For such an h the subgradient inequality of [Definition 7.1](#) is used unchanged: at a point x with $h(x) < +\infty$, $s \in \partial h(x)$ means $h(y) \geq h(x) + s^\top(y - x)$ for all y ; at a point with $h(x) = +\infty$ we set $\partial h(x) = \emptyset$, so that for $h = I_C$ the subdifferential is exactly the normal cone of [Definition 8.5](#), empty outside C . The word **simple** does not mean small. It means that the proximal operator of h , defined in [Section 9.3](#), is cheap to evaluate, typically by a closed formula costing $O(d)$ operations. This extends the simple sets of [Week 8](#): since the prox of I_C is Π_C ([Proposition 9.8](#)), a set C is simple in the sense of [Week 8](#) exactly when its indicator is simple in the present sense. We assume throughout that a minimiser x^* exists and write $F^* = F(x^*)$. The point of the split is that we never differentiate the whole objective: g is handled through its gradient and h through its proximal operator.

The week's examples all fit the template. In the data-fitting models $A \in \mathbb{R}^{N \times d}$ holds N observations of d features (rows $a_i^\top$), $b \in \mathbb{R}^N$ the responses, and the decision variable is the coefficient vector, written $\beta \in \mathbb{R}^d$ as in [Model 1.3](#) ([Weeks 6](#) and [8](#) wrote the coefficients as x , and [Weeks 10](#) and [12](#) return to x).

Model 9.3 (Lasso).

$$\min_{\beta \in \mathbb{R}^d} \frac{1}{2} \|A\beta - b\|_2^2 + \lambda \|\beta\|_1, \quad \lambda > 0. \quad (9.4)$$

Here $g(\beta) = \frac{1}{2} \|A\beta - b\|_2^2$ is the least-squares loss of [Model 6.16](#), with $\nabla g(\beta) = A^\top(A\beta - b)$ by the chain rule ([Proposition 1.17](#)), and $h(\beta) = \lambda \|\beta\|_1 = \lambda \sum_j |\beta_j|$ charges every unit of every coefficient at the same rate λ . Because the charge does not fade as $\beta_j \rightarrow 0$ (the slope of $|\beta_j|$ stays ± 1 right up to the kink), a feature must earn its place: a coefficient is nonzero only if it improves the fit by more than λ per unit. A solution is therefore typically *sparse*, and its nonzero pattern is read as the set of selected features. *Ridge* regression replaces the penalty by $h(\beta) = \frac{\lambda}{2} \|\beta\|_2^2$, whose slope $\lambda\beta_j$ vanishes at zero: it shrinks all coefficients but selects none. Ridge is smooth; the lasso is the first example for which new machinery is essential. The *logistic-lasso* is the classification analogue,

$$\min_{\beta \in \mathbb{R}^d} \sum_{i=1}^N \log(1 + \exp(-b_i a_i^\top \beta)) + \lambda \|\beta\|_1, \quad b_i \in \{-1, 1\}, \quad (9.5)$$

with the smooth logistic loss of [Example 6.17](#) as g and the same h . The fourth instance is a decision model rather than a learning model.

Model 9.4 (Turnover-penalised portfolio).

$$\min_{x \in \mathbb{R}^d} \frac{1}{2} x^\top \Sigma x - c^\top x + \lambda \|x - x^{\text{old}}\|_1. \quad (9.6)$$

The variable $x \in \mathbb{R}^d$ holds the positions in d assets. The smooth part $g(x) = \frac{1}{2} x^\top \Sigma x - c^\top x$ is the quadratic risk–return objective used in [Week 8](#), that is, the mean–variance objective of

Model 1.2 with the sign reversed and the letters of **Week 8**: c is the vector of expected returns and the risk-aversion weight is absorbed into the positive semidefinite matrix Σ . Its gradient is $\nabla g(x) = \Sigma x - c$. The vector x^{old} is the portfolio currently held, $\|x - x^{\text{old}}\|_1$ is the total amount traded, and λ acts as a proportional transaction cost. The regulariser is a *shifted* ℓ_1 norm, kinked at x^{old} rather than at zero, so we should expect solutions in which many positions are *not traded at all*. The model as stated has no budget or long-only constraint; **Week 13** returns to the portfolio with the turnover limit imposed as a hard constraint alongside others, a feasible set for which no cheap projection is available.

In summary, three methods are in play: those of **Week 6**, applied to the whole objective, for the smooth ridge term; the proximal gradient method of **Section 9.4** for the kinked ℓ_1 and turnover penalties; and, for an indicator I_C , the projected gradient method of **Week 8**, which will turn out to be a special case of proximal gradient.

9.2.2 Composite first-order optimality

How do we recognise a minimiser of $g + h$ when h has kinks, so that ∇F need not exist? The answer combines the gradient of g with the subdifferential of h .

Theorem 9.5 (Composite optimality condition). *Let $g : \mathbb{R}^d \rightarrow \mathbb{R}$ be convex and differentiable and let $h : \mathbb{R}^d \rightarrow \mathbb{R} \cup \{+\infty\}$ be convex. A point x^* with $h(x^*) < +\infty$ minimises $g + h$ if and only if*

$$0 \in \nabla g(x^*) + \partial h(x^*), \quad \text{that is,} \quad -\nabla g(x^*) \in \partial h(x^*). \quad (9.7)$$

Proof. ($\Leftarrow$) If $-\nabla g(x^*) \in \partial h(x^*)$, then $h(y) \geq h(x^*) - \nabla g(x^*)^\top (y - x^*)$ for all y , while convexity of g gives $g(y) \geq g(x^*) + \nabla g(x^*)^\top (y - x^*)$ (**Theorem 3.8**). Adding the two, $F(y) \geq F(x^*)$. ($\Rightarrow$) Let x^* be a minimiser, fix y with $h(y) < +\infty$ and $t \in (0, 1]$, and set $z_t = x^* + t(y - x^*)$. By convexity of h , $0 \leq F(z_t) - F(x^*) \leq g(z_t) - g(x^*) + t(h(y) - h(x^*))$. Dividing by t and letting $t \downarrow 0$ turns the first term into the directional derivative $\nabla g(x^*)^\top (y - x^*)$ and leaves $h(y) \geq h(x^*) - \nabla g(x^*)^\top (y - x^*)$, the subgradient inequality for $-\nabla g(x^*)$. $\square$

The gradient comes from the smooth loss and the subdifferential from the regulariser. The condition contains the earlier certificates. With $h = 0$ it is the stationarity condition $\nabla g(x^*) = 0$ that gradient descent aims at. In general it is a structured version of $0 \in \partial F(x^*)$ (**Theorem 7.7**), the structure being supplied by the sum rule of **Proposition 7.4**. That rule was stated for finite-valued functions; the direct proof above does not need it, so the theorem also covers an h that takes the value $+\infty$. In particular, with $h = I_C$, whose subdifferential is the normal cone $N_C(x^*)$ (**Definition 8.5** and the proposition that follows it), the condition is $0 \in \nabla g(x^*) + N_C(x^*)$, the constrained condition of **Theorem 8.7**.

9.2.3 Lasso optimality, coordinate by coordinate

For the lasso, (9.7) reads $0 \in A^\top(A\beta^* - b) + \lambda \partial \|\beta^*\|_1$, the condition previewed in **Section 5.4**. By **Example 7.5**, the subdifferential of the ℓ_1 norm is taken coordinate by coordinate, $\{\text{sign}(\beta_j)\}$ where $\beta_j \neq 0$ and $[-1, 1]$ where $\beta_j = 0$. Coordinate j of the condition therefore says

$$(A^\top(A\beta^* - b))_j = \begin{cases} -\lambda, & \beta_j^* > 0, \\ \text{some value in } [-\lambda, \lambda], & \beta_j^* = 0, \\ +\lambda, & \beta_j^* < 0. \end{cases} \quad (9.8)$$

The number $(A^\top(A\beta - b))_j = \partial g / \partial \beta_j$ is the rate at which the fit changes with coefficient j . At an active coefficient this rate balances the penalty rate λ exactly. A coefficient can stay exactly at zero as long as the smooth gradient lies inside the *threshold band* $[-\lambda, \lambda]$: moving it off zero would improve the fit at a rate of at most λ and cost exactly λ . This band is the source of

sparsity, and it quantifies the effect of λ : putting $\beta^* = 0$ in (9.8) shows that the zero vector is optimal if and only if $\lambda \geq \|A^\top b\|_\infty = \max_j |(A^\top b)_j|$. A smooth penalty has no band: the ridge condition $A^\top(A\beta^* - b) + \lambda\beta^* = 0$ forces $\beta_j^* = 0$ only when the smooth gradient vanishes exactly.

9.3 The proximal operator

9.3.1 Definition and interpretation

Projection answers the question “which point of C is closest to v ?”. For a soft penalty there is no set to land in, only a cost $h(z)$ to be weighed against the distance moved.

Definition 9.6 (Proximal operator). Let $h : \mathbb{R}^d \rightarrow \mathbb{R} \cup \{+\infty\}$ be convex, closed (its epigraph is a closed set) and finite somewhere, and let $\alpha > 0$. The **proximal operator** of αh maps $v \in \mathbb{R}^d$ to

$$\text{prox}_{\alpha h}(v) = \underset{z \in \mathbb{R}^d}{\operatorname{argmin}} \left\{ h(z) + \frac{1}{2\alpha} \|z - v\|_2^2 \right\}. \quad (9.9)$$

Two terms compete. The term $h(z)$ rewards points with a lower regularisation cost; the quadratic term keeps the output *local*, close to the input v . The parameter α sets the balance: for small α moving is expensive and $\text{prox}_{\alpha h}(v) \approx v$, while for large α the penalty dominates and the output moves further towards the minimisers of h . All regularisers in this book (norms, convex quadratics, indicators of closed convex sets) satisfy the hypotheses. The objective in (9.9) is a convex function plus the strongly convex quadratic $\frac{1}{2\alpha} \|z - v\|_2^2$ (Definition 6.12), so it has exactly one minimiser: $\text{prox}_{\alpha h}(v)$ is a single well-defined point, which is why (9.9) carries an equality sign. The next lemma characterises that point.

Lemma 9.7 (Prox optimality condition). For h and α as in Definition 9.6, $p = \text{prox}_{\alpha h}(v) \iff v - p \in \alpha \partial h(p)$.

Proof. Problem (9.9) is a composite problem whose smooth part $z \mapsto \frac{1}{2\alpha} \|z - v\|_2^2$ has gradient $(z - v)/\alpha$. By Theorem 9.5, p is its minimiser if and only if $0 \in (p - v)/\alpha + \partial h(p)$. $\square$

9.3.2 Projection as the special case of a hard constraint

Proposition 9.8 (The prox of an indicator is the projection). Let $C \subseteq \mathbb{R}^d$ be nonempty, closed and convex. Then $\text{prox}_{\alpha I_C}(v) = \Pi_C(v)$ for every $\alpha > 0$.

Proof. Since $I_C(z) = +\infty$ outside C and 0 on C ,

$$\text{prox}_{\alpha I_C}(v) = \underset{z}{\operatorname{argmin}} \left\{ I_C(z) + \frac{1}{2\alpha} \|z - v\|_2^2 \right\} = \underset{z \in C}{\operatorname{argmin}} \frac{1}{2\alpha} \|z - v\|_2^2 = \Pi_C(v),$$

the factor $1/(2\alpha)$ having no effect on the minimiser (Definition 8.10). $\square$

A hard constraint gives a projection; a soft penalty gives a proximal operator. The step size drops out of the first because $\alpha I_C = I_C$: an infinite cost is not made cheaper by scaling. A projection has a fixed target set: points outside C are pushed back into it, points inside are left alone, and once C is chosen there is no trade-off. A general prox has no target set: nearby points compete according to their penalty value, the output need not lie in any fixed set, and every input that is not already a minimiser of h is moved, by an amount that depends on α .

Example 9.9 (Clipping into a box). Let $C = \{x \in \mathbb{R}^3 : \ell \leq x \leq u\}$ with lower bounds $\ell = (0, -1, 0)$ and upper bounds $u = (2, 1, 1)$, and let $v = (2.6, -0.4, -0.7)$. By Proposition 9.8 and the box projector of Table 8.1, each coordinate is clipped into its own interval, so that $\text{prox}_{\alpha I_C}(v) = \Pi_C(v)$ with

$$\Pi_C(v) = (\min\{\max\{2.6, 0\}, 2\}, \min\{\max\{-0.4, -1\}, 1\}, \min\{\max\{-0.7, 0\}, 1\}) = (2, -0.4, 0)$$

for every $\alpha > 0$. The second coordinate is already inside its interval and passes through untouched; the other two land on their nearest bounds.

9.3.3 Non-expansiveness

The convergence proofs of [Week 8](#) rested on a single property of the projection: it never increases distances ([Proposition 8.12](#)). The proximal operator has the same property.

Proposition 9.10 (The proximal operator is non-expansive). *For h and α as in [Definition 9.6](#) and all $u, v \in \mathbb{R}^d$,*

$$\|\text{prox}_{\alpha h}(u) - \text{prox}_{\alpha h}(v)\|_2 \leq \|u - v\|_2.$$

Proof. Write $p = \text{prox}_{\alpha h}(u)$ and $q = \text{prox}_{\alpha h}(v)$. By [Lemma 9.7](#), $s_p = (u - p)/\alpha \in \partial h(p)$ and $s_q = (v - q)/\alpha \in \partial h(q)$. The two subgradient inequalities $h(q) \geq h(p) + s_p^\top(q - p)$ and $h(p) \geq h(q) + s_q^\top(p - q)$ add up to $(s_p - s_q)^\top(p - q) \geq 0$. Substituting and multiplying by α , $((u - p) - (v - q))^\top(p - q) \geq 0$, that is,

$$(u - v)^\top(p - q) \geq \|p - q\|_2^2.$$

The left-hand side is at most $\|u - v\|_2 \|p - q\|_2$ by the Cauchy–Schwarz inequality ([Proposition 1.11](#)); dividing by $\|p - q\|_2$ (the case $p = q$ is trivial) gives the claim. $\square$

This is the inequality that stood behind the non-expansiveness of the projection, with the projection theorem ([Theorem 8.11](#)) replaced by the prox optimality condition; with $h = I_C$ it reduces to [Proposition 8.12](#). Non-expansiveness is what allowed the correction step of [Week 8](#) to be applied without undoing the progress of the gradient step. It holds verbatim for the prox, so the whole analysis will transfer.

9.3.4 Closed-form proximal operators

The regulariser is “simple” when (9.9) can be solved by a formula. We work out the three cases that the week’s models need.

Proposition 9.11 (Soft-thresholding). *For a threshold level $\tau \geq 0$ define the **soft-thresholding function***

$$\text{soft}_\tau(v) = \begin{cases} v - \tau, & v > \tau, \\ 0, & |v| \leq \tau, \\ v + \tau, & v < -\tau, \end{cases} \quad \text{equivalently} \quad \text{soft}_\tau(v) = \text{sign}(v) \max\{|v| - \tau, 0\}, \quad (9.10)$$

applied coordinate by coordinate when v is a vector. Then, for $\lambda \geq 0$ and $\alpha > 0$, $\text{prox}_{\alpha\lambda|\cdot|}(v) = \text{soft}_{\alpha\lambda}(v)$ for $v \in \mathbb{R}$, and $\text{prox}_{\alpha\lambda\|\cdot\|_1}(v) = \text{soft}_{\alpha\lambda}(v)$ for $v \in \mathbb{R}^d$.

Proof. In one dimension the prox minimises $\lambda|z| + \frac{1}{2\alpha}(z - v)^2$, and by [Lemma 9.7](#) the minimiser z is characterised by $v - z \in \alpha\lambda \partial|z|$. The kink of $|z|$ at the origin forces a split into three cases. If $z > 0$, then $\partial|z| = \{1\}$, so $\lambda + \frac{1}{\alpha}(z - v) = 0$ and $z = v - \alpha\lambda$; this is consistent with $z > 0$ exactly when $v > \alpha\lambda$. If $z < 0$, then $\partial|z| = \{-1\}$ and $z = v + \alpha\lambda$, consistent exactly when $v < -\alpha\lambda$. If $z = 0$, then $\partial|z| = [-1, 1]$ and the condition reads $v \in [-\alpha\lambda, \alpha\lambda]$. The three conditions on v are mutually exclusive and cover the whole line, so they determine z , and the result is (9.10) with $\tau = \alpha\lambda$. In d dimensions the prox objective is $\sum_{j=1}^d (\lambda|z_j| + \frac{1}{2\alpha}(z_j - v_j)^2)$, a sum of terms each involving one coordinate only; it is minimised by minimising each term separately. $\square$

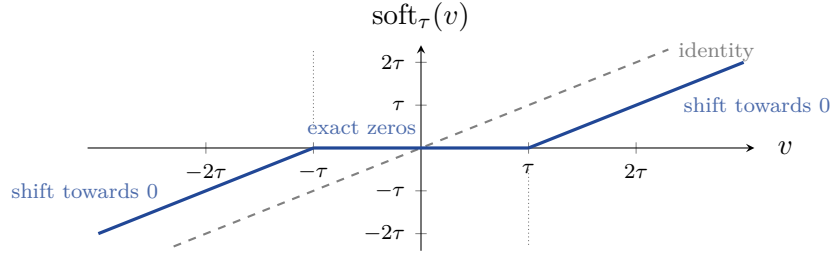


Figure 9.1: The soft-thresholding function soft_τ (solid) against the identity (dashed). Inputs in the dead zone $[-\tau, \tau]$ are mapped to exactly zero; every other input is moved towards zero by the fixed amount τ .

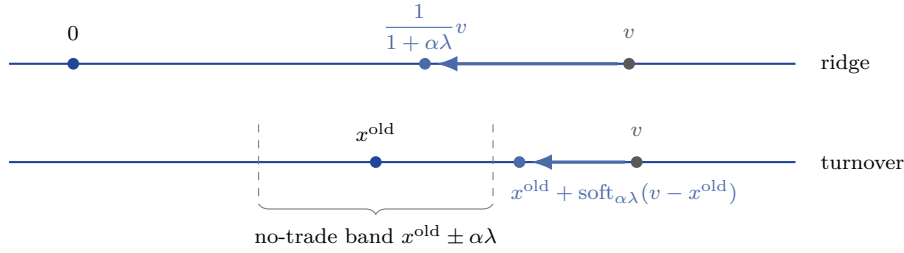


Figure 9.2: Two proximal operators on the real line: the ridge prox scales v towards the origin (top); the turnover prox moves v a distance $\alpha\lambda$ towards x^{old} and maps any input inside the dashed band to x^{old} itself (bottom).

Figure 9.1 shows the result: the optimal point is either shifted towards zero by the fixed amount $\tau = \alpha\lambda$ or thresholded to exactly zero. The dead zone $|v| \leq \alpha\lambda$ is the threshold band of (9.8) seen from the algorithm's side, and it is what will make the iterates of the method sparse.

For the ridge regulariser $h(x) = \frac{\lambda}{2} \|x\|_2^2$ no case split is needed. The prox objective $\frac{\lambda}{2} \|z\|_2^2 + \frac{1}{2\alpha} \|z - v\|_2^2$ is smooth, and setting its gradient to zero gives

$$\lambda z + \frac{1}{\alpha}(z - v) = 0 \quad \implies \quad \text{prox}_{\alpha h}(v) = \frac{1}{1 + \alpha\lambda} v. \quad (9.11)$$

Quadratic regularisation pulls the point smoothly towards the origin by a constant *factor*; a nonzero coordinate never becomes exactly zero, in line with the discussion after (9.8).

For the turnover penalty $h(x) = \lambda \|x - x^{\text{old}}\|_1$, substitute $u = z - x^{\text{old}}$, the trade, in (9.9). The objective becomes $\lambda \|u\|_1 + \frac{1}{2\alpha} \|u - (v - x^{\text{old}})\|_2^2$, whose minimiser is $u = \text{soft}_{\alpha\lambda}(v - x^{\text{old}})$ by Proposition 9.11. Hence

$$\text{prox}_{\alpha h}(v) = x^{\text{old}} + \text{soft}_{\alpha\lambda}(v - x^{\text{old}}), \quad (9.12)$$

with soft-thresholding applied coordinatewise. Thresholding happens around the current portfolio, not around zero (Figure 9.2).

Example 9.12 (A no-trade band). Let $x^{\text{old}} = (0.5, 0.3, 0.2)$, let $v = (0.62, 0.27, 0.05)$ be the portfolio proposed by a gradient step, and let $\alpha\lambda = 0.05$. The proposed trades $v - x^{\text{old}} = (0.12, -0.03, -0.15)$ are soft-thresholded at level 0.05 to $(0.07, 0, -0.10)$, so $\text{prox}_{\alpha h}(v) = (0.57, 0.30, 0.10)$. The trade in asset 2 is smaller than the threshold and is cancelled; the other two are shortened by 0.05 each. A turnover penalty pulls the portfolio towards the current holdings, not towards zero, and trades that are not worth their cost are not made at all.

Table 9.1 collects these formulas together with a few others that are obtained in the same way. The group penalty $\lambda \|x\|_2$ (the norm itself, not its square) shrinks the whole vector by the amount $\alpha\lambda$ and sets it to zero as a block when $\|v\|_2 \leq \alpha\lambda$. The general quadratic row follows from setting a gradient to zero as in (9.11), and costs a linear solve rather than $O(d)$; every other explicit formula in the table costs $O(d)$. A much longer catalogue is given by Parikh and Boyd, *Proximal Algorithms* (2014), Section 6.

Table 9.1: Common proximal operators $\text{prox}_{\alpha h}(v)$: projections (independent of α) and shrinkages. Here $(t)_+ = \max\{t, 0\}$, coordinatewise for vectors; in the halfspace row $a \neq 0$ and b is a scalar; in the group row the output is 0 when $v = 0$. The halfspace row is the hyperplane projection of Table 8.1 applied only when $a^\top v > b$; balls, affine sets and the simplex are in Table 8.1.

Regulariser $h(x)$	Proximal operator $\text{prox}_{\alpha h}(v)$
<i>Hard constraints, $h = I_C$ (prox = projection, Table 8.1)</i>	
feasible set C	$\Pi_C(v)$
interval or box $\ell \leq x \leq u$	clip each v_j into $[\ell_j, u_j]$
nonnegative orthant $x \geq 0$	$(v)_+$
halfspace $a^\top x \leq b$	$v - \frac{(a^\top v - b)_+}{\ a\ _2^2} a$
<i>Soft penalties (prox = shrinkage)</i>	
$\lambda x $ or $\lambda\ x\ _1$	$\text{soft}_{\alpha\lambda}(v)$, coordinatewise
$\lambda\ x\ _2$ (group penalty)	$(1 - \alpha\lambda/\ v\ _2)_+ v$
$\frac{\lambda}{2}\ x\ _2^2$ (ridge)	$v/(1 + \alpha\lambda)$
$\frac{1}{2}x^\top Px + q^\top x$, $P \succeq 0$	$(I + \alpha P)^{-1}(v - \alpha q)$
$\lambda\ x - x^{\text{old}}\ _1$ (turnover)	$x^{\text{old}} + \text{soft}_{\alpha\lambda}(v - x^{\text{old}})$

9.4 The proximal gradient method

9.4.1 Optimality as a fixed point

The optimality condition (9.7) is an inclusion; the prox turns it into an equation.

Proposition 9.13 (Fixed-point form of optimality). *Let g and h be as in Model 9.2, with h satisfying the hypotheses of Definition 9.6. For every $\alpha > 0$,*

$$0 \in \nabla g(x^*) + \partial h(x^*) \iff x^* = \text{prox}_{\alpha h}(x^* - \alpha \nabla g(x^*)).$$

Proof. Apply Lemma 9.7 with $v = x^* - \alpha \nabla g(x^*)$ and $p = x^*$: the right-hand statement holds if and only if $v - x^* = -\alpha \nabla g(x^*) \in \alpha \partial h(x^*)$, which is the left-hand statement. $\square$

An optimal point is one that the map “gradient step, then prox” leaves where it is, exactly as in Proposition 8.15 with Π_C in place of the prox. Two things follow: the natural algorithm is to iterate this map, and the natural measure of non-optimality is how far the map moves the current point.

9.4.2 Deriving the update from a local model

The update can be derived exactly as projected gradient was in Section 8.4, with the indicator I_C replaced by a general h . At the current iterate x^k , keep the regulariser exact and approximate only the smooth loss:

$$Q_{\alpha_k}(z; x^k) = g(x^k) + \nabla g(x^k)^\top (z - x^k) + \frac{1}{2\alpha_k} \|z - x^k\|_2^2 + h(z). \quad (9.13)$$

The first three terms are the stabilised local model of Equation (6.3): the tangent approximation of g plus a quadratic term that stops the step from leaving the region where the tangent can be trusted. The new choice is to leave $h(z)$ untouched, instead of linearising it or replacing it by a subgradient. The next iterate is the minimiser of the model, $x^{k+1} = \arg\min_z Q_{\alpha_k}(z; x^k)$. To compute it, write $v^k = x^k - \alpha_k \nabla g(x^k)$ for the gradient step and complete the square:

$$\nabla g(x^k)^\top (z - x^k) + \frac{1}{2\alpha_k} \|z - x^k\|_2^2 = \frac{1}{2\alpha_k} \|z - v^k\|_2^2 - \frac{\alpha_k}{2} \|\nabla g(x^k)\|_2^2$$

(the identity used for projected gradient in [Section 8.4](#)). Dropping the terms that do not depend on z leaves $x^{k+1} = \operatorname{argmin}_z \{h(z) + \frac{1}{2\alpha_k} \|z - v^k\|_2^2\}$, which is the definition of a prox. Therefore

$$x^{k+1} = \operatorname{prox}_{\alpha_k h}(x^k - \alpha_k \nabla g(x^k)). \quad (9.14)$$

The recipe, then, is to linearise the smooth part, keep the regulariser exact, and minimise the local model exactly, and the update is best read as two steps: first take the step suggested by the smooth loss, $v^k = x^k - \alpha_k \nabla g(x^k)$; then apply the correction dictated by the penalty, $x^{k+1} = \operatorname{prox}_{\alpha_k h}(v^k)$. [Algorithm 9.1](#) states the method in full. It collapses to earlier methods when h becomes simpler, which completes the family (9.2): with $h = 0$ the prox is the identity and the method is gradient descent ([Algorithm 6.1](#)); with $h = I_C$ the prox is Π_C and the method is projected gradient ([Algorithm 8.2](#)); and with $h = \lambda \|\cdot\|_1$ the prox is soft-thresholding and the update is

$$x^{k+1} = \operatorname{soft}_{\alpha\lambda}(x^k - \alpha \nabla g(x^k)), \quad \text{for the lasso} \quad \beta^{k+1} = \operatorname{soft}_{\alpha\lambda}(\beta^k - \alpha A^\top (A\beta^k - b)), \quad (9.15)$$

known as **ISTA**, the *iterative shrinkage-thresholding algorithm*.

Algorithm 9.1 Proximal gradient method (ISTA when $h = \lambda \|\cdot\|_1$)

Require: routines for ∇g and $\operatorname{prox}_{\alpha h}$; starting point x^0 ; step size $\alpha \in (0, 1/L]$ (or backtracking, [Remark 9.16](#)); tolerance $\varepsilon > 0$; iteration cap K

Ensure: an approximate minimiser of $F = g + h$

```

1: for  $k = 0, 1, \dots, K - 1$  do
2:    $v^k \leftarrow x^k - \alpha \nabla g(x^k)$                                  $\triangleright$  gradient step on the smooth loss
3:    $x^{k+1} \leftarrow \operatorname{prox}_{\alpha h}(v^k)$                          $\triangleright$  structured correction for the penalty
4:   if  $\|x^k - x^{k+1}\|_2 / \alpha \leq \varepsilon$  then return  $x^{k+1}$          $\triangleright \|G_\alpha(x^k)\|_2 \leq \varepsilon$ , Definition 9.15
5:   end if
6: end for
7: return  $x^K$ 

```

Each iteration costs one gradient of g plus one prox. For the lasso the gradient is a pair of matrix–vector products ($A\beta$, then $A^\top$ times the residual), $O(Nd)$ operations, and the prox is $O(d)$: the nonsmooth term adds almost nothing to the cost of a gradient step.

9.4.3 Worked instances

Example 9.14 (One proximal-gradient step for the lasso). Take the lasso with $A = I_3$, $b = (3.2, -0.6, 1.6)$ and $\lambda = 2$, so that $g(\beta) = \frac{1}{2} \|\beta - b\|_2^2$, $\nabla g(\beta) = \beta - b$ and $L = 1$. Choose $\alpha = \frac{1}{2} \leq 1/L$ and start from $\beta^k = (1, 0, 0)$, where $F(\beta^k) = 3.88 + 2 = 5.88$. *Gradient step:* $\nabla g(\beta^k) = (-2.2, 0.6, -1.6)$ and

$$v^k = \beta^k - \alpha \nabla g(\beta^k) = (1, 0, 0) - \frac{1}{2}(-2.2, 0.6, -1.6) = (2.1, -0.3, 0.8).$$

Prox step: the threshold is $\alpha\lambda = 1$, so $\beta^{k+1} = \operatorname{soft}_1((2.1, -0.3, 0.8)) = (1.1, 0, 0)$. The large coordinate is reduced by the threshold; the two small ones, which the gradient step had moved off zero, are set back to exactly zero. The objective falls to $F(\beta^{k+1}) = 3.665 + 2 = 5.865$. Repeating the step gives $(1.15, 0, 0)$, $(1.175, 0, 0)$, $\dots$, which converges to $\beta^* = \operatorname{soft}_2(b) = (1.2, 0, 0)$ with $F^* = 5.86$, the error halving at every step. At β^* the smooth gradient is $\beta^* - b = (-2, 0.6, -1.6)$: the first coordinate equals $-\lambda$ and the other two lie inside the band $[-2, 2]$, exactly as (9.8) requires.

For the logistic-lasso (9.5), nothing changes but the gradient. With $g(\beta) = \sum_{i=1}^N \log(1 + \exp(-b_i a_i^\top \beta))$ and $h(\beta) = \lambda \|\beta\|_1$, the chain rule gives (as in Example 6.17)

$$\nabla g(\beta) = - \sum_{i=1}^N \frac{b_i a_i}{1 + \exp(b_i a_i^\top \beta)}, \quad \beta^{k+1} = \text{soft}_{\alpha\lambda}(\beta^k - \alpha \nabla g(\beta^k)).$$

For the turnover model (9.6), $\nabla g(x) = \Sigma x - c$ and the prox is (9.12), so $x^{k+1} = x^{\text{old}} + \text{soft}_{\alpha\lambda}(x^k - \alpha(\Sigma x^k - c) - x^{\text{old}})$: every iterate leaves untraded the assets whose proposed trade falls inside the no-trade band. Once g and h are identified, the same template applies.

9.4.4 The proximal-gradient mapping and stopping

A practical method needs a computable quantity that is zero exactly at a solution and small near one. For gradient descent this was the gradient, which F does not have.

Definition 9.15 (Proximal-gradient mapping). For $\alpha > 0$ and g, h as in Proposition 9.13, the **proximal-gradient mapping** of the composite problem is

$$G_\alpha(x) = \frac{1}{\alpha} \left(x - \text{prox}_{\alpha h}(x - \alpha \nabla g(x)) \right).$$

By Proposition 9.13,

$$G_\alpha(x) = 0 \iff x = \text{prox}_{\alpha h}(x - \alpha \nabla g(x)) \iff 0 \in \nabla g(x) + \partial h(x),$$

so G_α vanishes exactly at the minimisers of F . The update (9.14) can be rewritten as $x^{k+1} = x^k - \alpha G_\alpha(x^k)$: the mapping plays the role that the gradient plays in gradient descent, and the natural stopping quantity is $\|G_\alpha(x^k)\|_2$. It costs nothing extra, since $\|G_\alpha(x^k)\|_2 = \|x^k - x^{k+1}\|_2 / \alpha$; the division by α matters, because it makes the test measure non-optimality rather than the size of the step. The residual of Week 6 was $\nabla f(x)$ and that of Week 8 the projected-gradient mapping (Definition 8.16); G_α reduces to the first when $h = 0$ and to the second when $h = I_C$. (The symbol is unrelated to the bound G on subgradient norms in Week 7.) In Example 9.14, $G_\alpha(\beta^k) = (\beta^k - \beta^{k+1})/\alpha = (-0.2, 0, 0)$, and the residual halves at each step along with the error.

9.4.5 Step sizes

If ∇g is L -Lipschitz, the standard safe rule is a fixed step $\alpha_k = \alpha \leq 1/L$, as for gradient descent. Only the smooth part enters: h sits inside the prox and has no influence on the admissible step. For the lasso, $\nabla^2 g = A^\top A$ and $L = \max_{\|u\|_2=1} \|Au\|_2^2$, the largest directional curvature of $\frac{1}{2} \|A\beta - b\|_2^2$. This number is the largest eigenvalue of $A^\top A$ (eigenvalues are introduced in Definition 10.2), and NumPy computes it in one line, `np.linalg.eigvalsh(A.T @ A).max()`; the bound $\|A\|_F^2$ of Model 6.16 (the sum of the squared entries of A) is a cruder but also safe overestimate. For the logistic loss the Hessian $\sum_i \sigma_i(1 - \sigma_i) a_i a_i^\top$, with $\sigma_i = 1/(1 + \exp(-b_i a_i^\top \beta))$ the modelled probability of the observed label b_i , has weights $\sigma_i(1 - \sigma_i) \leq \frac{1}{4}$ (Example 6.17), so a quarter of the least-squares constant for the same A is a safe value of L ; for the turnover model L is the largest eigenvalue of Σ .

Remark 9.16 (Backtracking). When L is unknown, the step is found by backtracking on the smooth part. Start from a trial α , compute the candidate $x^+ = \text{prox}_{\alpha h}(x^k - \alpha \nabla g(x^k))$, and shrink $\alpha \leftarrow \eta \alpha$ with a fixed $\eta \in (0, 1)$ (recomputing x^+ each time) until the smooth upper-model condition

$$g(x^+) \leq g(x^k) + \nabla g(x^k)^\top (x^+ - x^k) + \frac{1}{2\alpha} \|x^+ - x^k\|_2^2 \quad (9.16)$$

holds. By the descent lemma (Proposition 6.9) the test passes for every $\alpha \leq 1/L$, so the loop stops after finitely many shrinks, and it is the only property of the step that the proof of Theorem 9.18

uses (Theorem 9.19 is stated for $\alpha = 1/L$ only). The regulariser stays inside the prox: we never line-search on h separately. With $h = I_C$ it is the test used for projected gradient in Section 8.4, where $\eta = \frac{1}{2}$; for $h = 0$ the test is the Armijo condition of Algorithm 6.2 with parameter $\frac{1}{2}$ (that algorithm's letters α and β are both taken this week, hence η for the shrink factor).

Note on the lecture slides

Frame 9.31 says to backtrack “on the smooth part g exactly as in Week 6”. Read literally, that tests g at the gradient-step point $x^k - \alpha \nabla g(x^k)$, whereas the iterate taken is the prox point x^+ , and such a test can accept steps that *increase* F ; the test the frame's own diagram describes (“check smooth upper-model condition”) is (9.16) at x^+ , and that is the test the notes use, as they did for projected gradient in Section 8.4.

9.5 Convergence, tuning and implementation

9.5.1 Proximal gradient versus the subgradient method

The subgradient method on the full objective, $x^{k+1} = x^k - \alpha_k s^k$ with $s^k \in \partial(g + h)(x^k)$, throws away the smooth structure of g : its baseline rate is $O(1/\sqrt{k})$, it needs diminishing steps, it is not a descent method, and a step along $-s^k$ lands on a kink only by coincidence, so it produces no exact zeros. Proximal gradient uses the smooth structure directly: it runs with a constant step, decreases F at every iteration, attains the rates of gradient descent, and creates exact zeros through soft-thresholding. The next example measures the difference; the two theorems that follow explain it.

Example 9.17 (ISTA against the subgradient method on a lasso). A simulated lasso instance has $N = 80$ observations and $d = 40$ features: the entries of A are independent normal draws scaled by $1/\sqrt{N}$, the true coefficient vector has 6 nonzero entries drawn from a normal distribution with standard deviation 2, the responses carry Gaussian noise of standard deviation 0.05, and $\lambda = 0.08$. The code that generates the instance is printed after Listing 9.1, and the companion notebook `week09_proximal.ipynb` contains the same run; as in that code, the coefficient vector is written x in this example. Here $L = 2.784$, so ISTA uses $\alpha = 1/L = 0.359$; the subgradient method uses $s^k = A^\top(Ax^k - b) + \lambda \text{sign}(x^k)$ with the same step scale, $\alpha_k = \alpha/\sqrt{k+1}$, and reports its best value so far. Both start from $x^0 = 0$, where $F(x^0) - F^* = 27.46$ with $F^* = 1.11392903$. Figure 9.3 plots the gap. After 10, 50 and 100 iterations ISTA's gap is 0.305, 1.7×10^{-6} and 3×10^{-14} ; the subgradient method's is 0.942, 0.208 and 0.072, and it is still 1.1×10^{-3} after 400 iterations. On the logarithmic scale ISTA's curve is, from about $k = 30$, a straight line (the gap shrinks by the factor 0.70 per iteration) until it falls below 10^{-14} , where the plot is clipped: the convergence is *linear*, and the objective values decrease monotonically. ISTA's solution has exactly 12 nonzero coordinates (including the 6 true ones) and 28 exact zeros, whereas all 40 coordinates of the subgradient iterate are nonzero: they are merely small. At the solution the smooth gradient of every zero coordinate has absolute value at most $0.0747 \leq \lambda$, as (9.8) requires.

9.5.2 Descent and the $O(1/k)$ rate

Theorem 9.18 (Proximal gradient: descent and sublinear rate). *Let g be convex and L -smooth, let h be as in Definition 9.6, let $F = g + h$ have a minimiser x^* , and run (9.14) with a fixed step $0 < \alpha \leq 1/L$. Then $F(x^{k+1}) \leq F(x^k)$ for all k , and*

$$F(x^k) - F^* \leq \frac{\|x^0 - x^*\|_2^2}{2\alpha k} = O(1/k), \quad k \geq 1.$$

Proof sketch. Descent. For $\alpha \leq 1/L$ the descent lemma (Proposition 6.9) gives $g(z) \leq g(x^k) + \nabla g(x^k)^\top(z - x^k) + \frac{1}{2\alpha} \|z - x^k\|_2^2$ for all z . Adding $h(z)$ to both sides shows that the local model

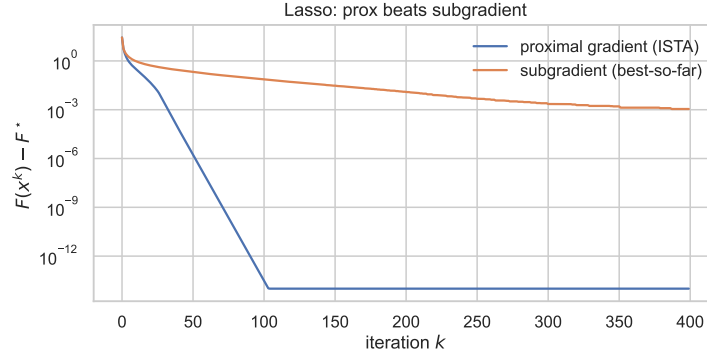


Figure 9.3: Measured objective gap $F(x^k) - F^*$ (logarithmic scale) on the lasso instance of [Example 9.17](#): ISTA with $\alpha = 1/L$ converges linearly and falls below 10^{-14} , the plotting floor, near $k = 100$; the subgradient method (best value so far) crawls.

(9.13) is an *upper model* of F , $Q_\alpha(z; x^k) \geq F(z)$, and it is tight at x^k : $Q_\alpha(x^k; x^k) = F(x^k)$. Because x^{k+1} minimises the model, $F(x^{k+1}) \leq Q_\alpha(x^{k+1}; x^k) \leq Q_\alpha(x^k; x^k) = F(x^k)$.

Rate. The three steps of the proof for gradient descent ([Theorem 6.11](#)) go through with $G_\alpha(x^k)$ in place of $\nabla f(x^k)$: the sufficient decrease $F(x^{k+1}) \leq F(x^k) - \frac{\alpha}{2} \|G_\alpha(x^k)\|_2^2$; the one-step bound $F(x^{k+1}) - F^* \leq \frac{1}{2\alpha} (\|x^k - x^*\|_2^2 - \|x^{k+1} - x^*\|_2^2)$; and a telescoping sum over the first k iterations, in which the last of the k gaps is the smallest because the values $F(x^i)$ decrease. The first two follow by adding three inequalities: the upper bound on $g(x^{k+1})$ just used, the convexity of g , and the subgradient inequality for h at x^{k+1} with the subgradient $G_\alpha(x^k) - \nabla g(x^k) \in \partial h(x^{k+1})$ that [Lemma 9.7](#) supplies. $\square$

The bound is word for word that of gradient descent ([Theorem 6.11](#)): the kinks of h cost nothing in the rate, because h is never approximated. To reach $F(x^k) - F^* \leq \varepsilon$ it suffices to take $k \geq \|x^0 - x^*\|_2^2 / (2\alpha\varepsilon)$ iterations, against the order of $1/\varepsilon^2$ iterations needed by the subgradient method ([Corollary 7.15](#)): for $\varepsilon = 10^{-3}$, thousands of iterations rather than millions. The statement also holds with the backtracking steps of [Remark 9.16](#), with α replaced by the smallest step used.

9.5.3 Linear convergence under strong convexity

Theorem 9.19 (Proximal gradient: linear rate). *In addition to the assumptions of [Theorem 9.18](#), let g be μ -strongly convex, that is, $g(y) \geq g(x) + \nabla g(x)^\top(y - x) + \frac{\mu}{2} \|y - x\|_2^2$ for all x, y ([Definition 6.12](#)), and let $\alpha = 1/L$. Then*

$$\|x^{k+1} - x^*\|_2 \leq \left(1 - \frac{\mu}{L}\right) \|x^k - x^*\|_2, \quad \text{hence} \quad \|x^k - x^*\|_2 \leq \left(1 - \frac{\mu}{L}\right)^k \|x^0 - x^*\|_2.$$

Proof. Define the proximal-gradient map (the update itself, not the residual G_α of [Definition 9.15](#)) $T_\alpha(x) = \text{prox}_{\alpha h}(x - \alpha \nabla g(x))$, so that $x^{k+1} = T_\alpha(x^k)$ and, by [Proposition 9.13](#), $x^* = T_\alpha(x^*)$. By non-expansiveness of the prox ([Proposition 9.10](#)),

$$\|x^{k+1} - x^*\|_2 = \|T_\alpha(x^k) - T_\alpha(x^*)\|_2 \leq \|(x^k - x^*) - \alpha(\nabla g(x^k) - \nabla g(x^*))\|_2.$$

The right-hand side involves only the gradient step on g . For a μ -strongly convex, L -smooth g and $\alpha = 1/L$ it is at most $(1 - \mu/L) \|x^k - x^*\|_2$; this is the gradient-step contraction that [Week 8](#) quotes as a lemma just before [Theorem 8.18](#) and uses in the same way. $\square$

Note on the lecture slides

Frame 9.35 calls the contraction $\left\| (x - y) - \frac{1}{L}(\nabla g(x) - \nabla g(y)) \right\|_2 \leq (1 - \frac{\mu}{L}) \|x - y\|_2$ “the Week 6 gradient-step bound”. Week 6 does not contain it (its linear rate is stated for function values). The inequality, which is correct, first appears in Week 8 as a lemma quoted without proof just before [Theorem 8.18](#), and the notes cite it from there.

This is the argument of [Week 8](#) line for line, with Π_C replaced by $\text{prox}_{\alpha h}$ and ∇f by ∇g : the correction now enforces a penalty instead of feasibility, and non-expansiveness lets either correction survive the estimate for the gradient step. For a separable quadratic g , with curvatures $D_i \in [\mu, L]$ as in [Example 6.14](#), the quoted contraction is one line: the gradient step multiplies coordinate i of $x^k - x^*$ by $1 - D_i/L \in [0, 1 - \mu/L]$; a general quadratic reduces to this case in the eigenvector coordinates of [Week 10](#), as [Section 8.4](#) noted. Reducing the distance to x^* by a factor ε takes about $(L/\mu) \ln(1/\varepsilon)$ iterations, so the method is fast when L/μ is moderate and slow when it is large; without strong convexity only [Theorem 9.18](#) remains. In [Example 9.17](#), A has full column rank, so g is strongly convex with $\mu = 0.0904$ (the smallest eigenvalue of $A^\top A$) and $1 - \mu/L = 0.9675$. The measured ratios $\|x^{k+1} - x^*\|_2 / \|x^k - x^*\|_2$ never exceed 0.908 and settle near 0.837: the theorem is a worst-case guarantee and the run does better, but the straight line in [Figure 9.3](#) is the behaviour it predicts.

Remark 9.20 (Where the strong convexity must sit). The theorem needs the strong convexity in the *smooth* part. A ridge-type term $\frac{\mu}{2} \|x\|_2^2$ added to supply it should therefore be counted in g (for instance $g = \frac{1}{2} \|A\beta - b\|_2^2 + \frac{\mu}{2} \|\beta\|_2^2$, $h = \lambda \|\beta\|_1$); if it is left in h and g alone is not strongly convex, the theorem as stated gives only the factor 1.

9.5.4 The role of the regularisation strength

The parameter λ is chosen by the modeller, not by the algorithm, and larger λ means more shrinkage, more sparsity, or less turnover. The solution $\beta(\lambda)$ traced as a function of λ is the *regularisation path*: in the typical picture the coefficients shrink as λ grows and leave the model one after another, although real lasso paths, which are piecewise linear in λ , need not be monotone. [Table 9.2](#) shows this on the instance of [Example 9.17](#): as λ grows, the solution becomes sparser and smaller and fits worse. This is [Proposition 9.1](#) at work: each solution solves the constrained problem with budget $\tau = \|\beta(\lambda)\|_1$, and the budget shrinks as λ grows. The solution is zero once $\lambda \geq \|A^\top b\|_\infty = 6.606$, as predicted in [Section 9.2](#). The modelling trade-off and the optimisation landscape change together: λ also sets the threshold $\alpha\lambda$ applied at every iteration.

Table 9.2: Effect of λ on the ISTA solution $\beta(\lambda)$ of the instance of [Example 9.17](#).

λ	0.01	0.08	0.5	2	4
nonzero coefficients (of 40)	36	12	5	3	2
complexity $\ \beta(\lambda)\ _1$	14.06	12.75	10.15	5.80	2.54
fit $\frac{1}{2} \ A\beta(\lambda) - b\ _2^2$	0.047	0.094	0.79	5.85	15.6

9.5.5 Implementation: NumPy ISTA and an RSOME benchmark

[Listing 9.1](#) implements ISTA for the lasso. Three patterns are worth copying: the gradient is computed with matrix–vector products rather than loops over observations; soft-thresholding is applied to the whole vector at once, in the $\text{sign}(v) \max\{|v| - \tau, 0\}$ form of [\(9.10\)](#); and both the objective value and the residual $\|G_\alpha(x^k)\|_2$ are monitored, the latter free from two consecutive iterates. (`hist[k]` is $F(x^{k+1})$, whereas [Figure 9.3](#) plots $F(x^k)$.)

Listing 9.1: ISTA for the lasso in NumPy, with the $\|G_\alpha\|_2$ stopping test.

```
import numpy as np

def soft_threshold(v, tau):
    return np.sign(v) * np.maximum(np.abs(v) - tau, 0.0)

def ista(A, b, lam, alpha, x0, nsteps, tol=1e-8):
    x = x0.copy()
    hist = []
    for _ in range(nsteps):
        grad = A.T @ (A @ x - b)          # gradient of the smooth part
        x_prev = x
        x = soft_threshold(x - alpha * grad, alpha * lam)  # prox step
        hist.append(0.5*np.linalg.norm(A @ x - b)**2
                    + lam*np.linalg.norm(x, 1))
        G = (x_prev - x) / alpha          # proximal-gradient mapping
        if np.linalg.norm(G) < tol: break
    return x, np.array(hist)
```

The following lines generate the instance of [Example 9.17](#) and run ISTA on it.

```
rng = np.random.default_rng(0)          # simulated instance
N, d, s = 80, 40, 6
A = rng.normal(size=(N, d)) / np.sqrt(N)
x_true = np.zeros(d); idx = rng.choice(d, s, replace=False)
x_true[idx] = rng.normal(scale=2.0, size=s)  # planted coefficients
b = A @ x_true + 0.05 * rng.normal(size=N)
lam = 0.08
alpha = 1.0 / np.linalg.eigvalsh(A.T @ A).max()  # alpha = 1/L
x_ista, hist = ista(A, b, lam, alpha, np.zeros(d), 400)
print(len(hist), hist[-1], np.count_nonzero(x_ista))
```

The last line prints 117 1.1139290255035426 12: the stopping test triggers after 117 iterations, at the objective value F^* to eight decimals, with 12 nonzero coefficients.

To check such a result against a general-purpose solver, the lasso must first be put in a form the solver accepts. Split the coefficients into positive and negative parts, $\beta = u - v$ with $u, v \geq 0$ (the split-variable device of [Week 4](#); these new decision variables are not the points u, v of [Section 9.3](#)).

Model 9.21 (Split-variable form of the lasso).

$$\begin{aligned} \min_{u,v \in \mathbb{R}^d} \quad & \frac{1}{2} \|A(u - v) - b\|_2^2 + \lambda \mathbf{1}^\top(u + v) \\ \text{s.t.} \quad & u \geq 0, \quad v \geq 0. \end{aligned} \tag{9.17}$$

This is a smooth convex quadratic programme in $2d$ variables. It is equivalent to (9.4): since $\lambda > 0$, lowering u_j and v_j by the same amount leaves the fit unchanged and reduces the penalty, so at an optimum $\min\{u_j, v_j\} = 0$ for every j , and then $\mathbf{1}^\top(u + v) = \|u - v\|_1$. The lasso solution is recovered as $\beta = u - v$. [Listing 9.2](#) solves (9.17) with the RSOME workflow of [Section 4.5](#).

Listing 9.2: RSOME benchmark for the lasso via split variables.

```
from rsome import ro, grb_solver as grb
import rsome as rso
F = lambda x: 0.5*np.linalg.norm(A @ x - b)**2 + lam*np.linalg.norm(x, 1)
model = ro.Model('lasso')
u = model.dvar(d)
v = model.dvar(d)
model.min(0.5 * rso.sumsqr(A @ (u - v) - b)
```

```

        + lam * (u.sum() + v.sum()))
model.st(u >= 0, v >= 0)
model.solve(grb)
print(model.solution.status)      # 2 = optimal (Gurobi's status code)
print(model.get(), F(x_ista))     # solver objective vs ISTA objective
beta = u.get() - v.get()

```

With Gurobi the status is 2 and the last `print` shows 1.11392922 against 1.11392903: the two objectives agree to about 2×10^{-7} , and the coefficient vectors to 2.9×10^{-5} . With MOSEK (`msk_solver`) the status prints as `Optimal` and the objective is 1.11392903. In both lanes the solver's small coefficients are only approximately zero, while ISTA's zeros are exact. The solver is a trustworthy benchmark on small instances; ISTA is the method of choice when the prox is cheap and the dimensions grow, since it needs only products with A and $A^\top$.

9.6 Summary and looking ahead

Regularisation creates composite structure: a smooth loss g plus a simple convex penalty h . The certificate for such a problem, $0 \in \nabla g(x^*) + \partial h(x^*)$, takes its gradient from the loss and its subdifferential from the penalty, and for the lasso it explains sparsity through the threshold band $[-\lambda, \lambda]$. The proximal operator is the soft-penalty analogue of projection, and projection is the prox of a hard constraint; for the penalties of interest it is a one-line formula. Minimising the local model of [Week 6](#) with h kept exact gives the proximal gradient method, which contains gradient descent and projected gradient as special cases. Because the prox is non-expansive and the local model majorises F when $\alpha \leq 1/L$, the method descends, stops on the residual $\|G_\alpha(x^k)\|_2$, and inherits the smooth rates: $O(1/k)$ in general and the factor $1 - \mu/L$ when g is strongly convex. The smooth rate survives a nonsmooth term whose prox is cheap.

The limitations are equally clear. Everything rests on a cheap prox: when h has none (the indicator of a general polyhedron, say) none of this applies, and that is the problem of [Week 13](#). The factor $1 - \mu/L$ is close to one when L/μ is large, and without strong convexity the rate is only $O(1/k)$. The prox itself returns with sampled gradients in [Week 11](#) and, one coordinate at a time and inside ADMM, in [Week 12](#). First, [Week 10](#) asks whether the same gradients and the same prox, used more cleverly, can do better on both counts. For smooth problems it names L/μ the condition number κ and shows that momentum improves the factor $1 - 1/\kappa$ to $1 - 1/\sqrt{\kappa}$; for the composite problem the answer is FISTA ([Algorithm 10.3](#)), which keeps the gradient and the prox of [Algorithm 9.1](#), adds one extrapolation step, and improves $O(1/k)$ to $O(1/k^2)$.

9.7 Exercises

Exercise 9.1 (Identify g , h and the right method). For each model identify the smooth part g , the simple part h and the most natural starting method, and explain whether the machinery of [Week 6](#), of [Week 8](#) or of this week is the correct baseline: (i) $\min_\beta \frac{1}{2} \|A\beta - b\|_2^2 + \frac{\lambda}{2} \|\beta\|_2^2$; (ii) $\min_\beta \frac{1}{2} \|A\beta - b\|_2^2 + \lambda \|\beta\|_1$; (iii) $\min_x \frac{1}{2} x^\top \Sigma x - c^\top x + \lambda \|x - x^{\text{old}}\|_1$; (iv) $\min_\beta \sum_{i=1}^N \log(1 + \exp(-b_i a_i^\top \beta)) + \lambda \|\beta\|_1$.

Exercise 9.2 (Penalised and constrained forms). Rewrite the lasso ([9.4](#)) and the turnover model ([9.6](#)) as constrained trade-off models with a budget τ , and say in words what τ limits in each. Using [Proposition 9.1](#) and the measured values in [Table 9.2](#), describe how the budget matched to λ moves as λ increases, and explain the roles of λ and τ to a non-specialist.

Exercise 9.3 (Proximal operators by hand). Verify the following formulas: (a) $\text{prox}_{\alpha I_C}(v) = \Pi_C(v)$; (b) $\text{prox}_{\alpha \lambda |\cdot|}(v) = \text{soft}_{\alpha \lambda}(v)$ in one dimension; (c) $\text{prox}_{\alpha \lambda \|\cdot\|_1}(v)$ is coordinatewise soft-thresholding; (d) $\text{prox}_{\alpha(\lambda/2)\|\cdot\|_2^2}(v) = \frac{1}{1+\alpha\lambda}v$; (e) $\text{prox}_{\alpha \lambda \|\cdot - x^{\text{old}}\|_1}(v) = x^{\text{old}} + \text{soft}_{\alpha \lambda}(v - x^{\text{old}})$. In

one derivation, explain clearly where the case split is needed. Then evaluate (c), (d) and (e) at $v = (1.4, -0.2, -0.9)$ with $\alpha\lambda = 0.5$ and $x^{\text{old}} = (0.5, 0, -1)$.

Exercise 9.4 (Derive ISTA and do one step). Starting from $\operatorname{argmin}_z \{g(x^k) + \nabla g(x^k)^\top (z - x^k) + \frac{1}{2\alpha_k} \|z - x^k\|_2^2 + h(z)\}$, derive $x^{k+1} = \operatorname{prox}_{\alpha_k h}(x^k - \alpha_k \nabla g(x^k))$. Then consider the lasso with $A = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}$, $b = (2, 1)$, $\lambda = 2$ and $\beta^0 = (0, 0)$. Check that $\alpha = \frac{1}{4}$ is a safe step (here $L = (3 + \sqrt{5})/2 \approx 2.62$), compute two ISTA steps by hand, and identify which coordinates are thresholded to zero. Verify with (9.8) that $\beta^* = (0, \frac{1}{2})$ is optimal.

Exercise 9.5 (Proximal-gradient mapping and stopping). Show that $G_\alpha(x) = \frac{1}{\alpha}(x - \operatorname{prox}_{\alpha h}(x - \alpha \nabla g(x))) = 0$ if and only if $0 \in \nabla g(x) + \partial h(x)$. Then answer: (a) why is $\|G_\alpha(x)\|_2$ a sensible stopping diagnostic? (b) how does it compare with the residual $\|\nabla f(x)\|_2$ of Week 6? (c) how does it compare with the projected-gradient mapping of Week 8? Finally, evaluate $\|G_\alpha(\beta^k)\|_2$ for the iterates of the previous exercise and describe its behaviour.

Exercise 9.6 (Coding lab). Implement ISTA for the lasso in NumPy. Compare it against (a) a subgradient implementation for the full objective and (b) an RSOME benchmark using split variables $u, v \geq 0$ with $\beta = u - v$. Report the final objective value, the sparsity pattern of the solution, the residual $\|G_\alpha(x^k)\|_2$, and the qualitative effect of changing λ .

Week 10 Accelerated First-Order Methods: Momentum, Nesterov's Method and FISTA

Can the same gradients, used more cleverly, get us there faster? [Weeks 6, 8 and 9](#) built three methods with one shape, a gradient step on the smooth part of the objective followed by nothing, by a projection or by a proximal operator, and one pair of guarantees: an optimality gap of order $O(1/k)$ for convex problems and, under strong convexity, the factor $(1 - \mu/L)^k$ (on the objective gap in [Week 6](#), on the distance to x^* in [Weeks 8 and 9](#)). This week starts from what goes wrong with them in practice. On an objective whose level sets form a long, thin valley, gradient descent crawls: the ratio L/μ , which [Example 6.14](#) called the curvature balance and which we now name the *condition number* κ , can run into the thousands, and the factor $1 - 1/\kappa$ is then painfully close to one. A longer step is no cure, because beyond $1/L$ the decrease guaranteed by [Proposition 6.9](#) shrinks, and at $2/L$ it vanishes. The cure is to use *history*: momentum adds a multiple $\beta(x^k - x^{k-1})$ of the previous move to the gradient step, and Nesterov's refinement evaluates the gradient at an *extrapolated point* $y^k = x^k + \beta(x^k - x^{k-1})$ instead of at x^k . Nothing else changes, neither the gradient routine nor the projection nor the prox, so the same idea accelerates the methods of [Weeks 8 and 9](#) as well. What changes is the rate: about $\sqrt{\kappa} \ln(1/\varepsilon)$ iterations instead of $\kappa \ln(1/\varepsilon)$ for strongly convex problems, $O(1/k^2)$ instead of $O(1/k)$ for convex ones, and neither order can be beaten by any method that uses only gradients.

[Section 10.1](#) shows the crawl, the momentum idea and Nesterov's look-ahead, and measures the difference on one small quadratic. [Section 10.2](#) explains it: on a quadratic the iteration splits into independent one-dimensional *modes*, the slowest mode is governed by κ , and momentum replaces κ by $\sqrt{\kappa}$. [Section 10.3](#) gives the momentum schedule for general convex problems and the $O(1/k^2)$ guarantee. [Section 10.4](#) accelerates projection and prox, with the lasso as the canonical example, and collects the first-order methods of the unit in one ladder. [Section 10.5](#) adds what the theory leaves out: non-monotonicity and restart, backtracking, stopping residuals and three measured benchmarks.

Three conventions apply throughout. The step size is α , as in [Weeks 7 to 9](#); [Week 6](#) wrote t_k . The letter β is the **momentum coefficient**, a number in $[0, 1)$ that scales the previous move (in [Week 9](#) β was the coefficient vector of the lasso, which this week writes as x ; in [Algorithm 6.2](#) α and β were the Armijo constant and the shrink factor). And the scalar θ_k of the momentum schedule in [Section 10.3](#) is *not* a step size, whatever its resemblance to [Week 6](#)'s t_k .

10.1 Why gradient descent crawls, and the momentum idea

10.1.1 The elongated valley

[Week 6](#) minimised $f(x) = \frac{1}{2}(x_1^2 + 25x_2^2)$ and found that the coordinate with large curvature settles at once while the one with small curvature creeps ([Example 6.14](#)). [Figure 10.1](#) shows the same phenomenon on a rotated version of that problem, whose level sets are ellipses five times ($\sqrt{25}$) longer than they are wide, run with an *exact line search*: at each iteration the step size, written t_k as in [Week 6](#), minimises f along the negative gradient. The iterates bounce from one side of the valley to the other and advance only slowly along it, and the bouncing is no accident. Write $\phi(t) = f(x^k - t\nabla f(x^k))$; by the chain rule $\phi'(t) = -\nabla f(x^k - t\nabla f(x^k))^\top \nabla f(x^k)$, and at the minimising step $\phi'(t_k) = 0$, that is,

$$\nabla f(x^{k+1})^\top \nabla f(x^k) = 0. \quad (10.1)$$

Under an exact line search every step is orthogonal to the one before it (the measured angle in [Figure 10.1](#) is 90.0°), and orthogonal steps cannot follow a diagonal valley; they can only zig-zag across it. The steep directions are dealt with almost immediately; the flat direction consumes the iterations. Nothing is wrong with the algorithm: the difficulty is the *geometry of the level sets*.

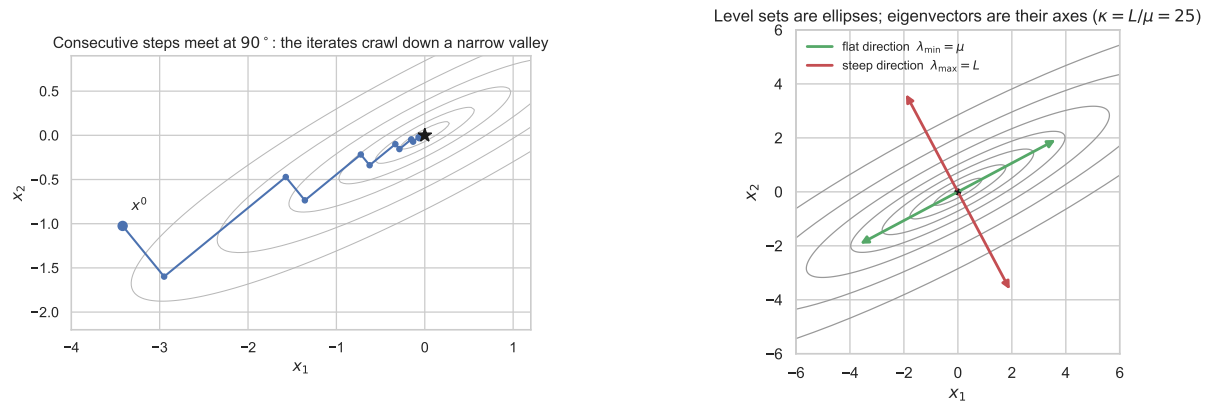


Figure 10.1: The quadratic of Example 10.1 ($\kappa = 25$). Left: gradient descent with an exact line search, twelve steps from x^0 ; consecutive steps meet at 90° , as (10.1) predicts, so the iterates cross the valley instead of travelling along it to the minimiser (star). Right: the level sets with the eigenvectors of Q (Definition 10.2), the axes of the ellipses; flat direction $\lambda_{\min} = \mu = 1$, steep direction $\lambda_{\max} = L = 25$.

Algorithm 10.1 Heavy-ball momentum

Require: a routine for ∇f ; starting point x^0 ; step size $\alpha > 0$; momentum coefficient $\beta \in [0, 1)$; iteration cap K

Ensure: an approximate minimiser of f

```

1:  $x^{-1} \leftarrow x^0$ 
2: for  $k = 0, 1, \dots, K - 1$  do
3:    $x^{k+1} \leftarrow x^k - \alpha \nabla f(x^k) + \beta (x^k - x^{k-1})$             $\triangleright$  gradient term plus momentum term
4: end for
5: return  $x^K$ 

```

Taking a larger step does not help. The descent lemma (Proposition 6.9) guarantees $f(x - \alpha \nabla f(x)) \leq f(x) - \alpha(1 - L\alpha/2) \|\nabla f(x)\|_2^2$: the guaranteed decrease grows with α only up to $\alpha = 1/L$, where the quadratic upper model is minimised, and for $\alpha \geq 2/L$ the lemma promises nothing; on the quadratic above a step $\alpha > 2/L$ makes the steep coordinate grow and the method diverges. Acceleration is not “just take a larger α ”. It keeps the safe step $\alpha = 1/L$ and uses the history of the iterates to build up speed along the flat direction, where progress is persistent, without amplifying the steep direction, where it is not.

Note on the lecture slides

The zig-zag picture on frame 10.2 uses an exact line search; every other run of the lecture and the mode analysis of frame 10.12 use the fixed step $\alpha = 1/L$, under which gradient descent on this quadratic does *not* zig-zag but creeps along the flat axis by 0.96 per step (Proposition 10.4). Zig-zag and crawl are two faces of the same ill-conditioning.

10.1.2 Heavy-ball momentum

The natural first idea is to carry some of the previous motion forward. Polyak’s **heavy-ball method** adds to the gradient step a multiple of the last displacement,

$$x^{k+1} = x^k - \alpha \nabla f(x^k) + \beta (x^k - x^{k-1}), \quad \alpha > 0, \beta \in [0, 1), x^{-1} = x^0 \quad (10.2)$$

(Algorithm 10.1). The *gradient term* moves downhill now; the *momentum term* reuses useful motion, like a ball rolling down the valley. An iteration costs one gradient, $O(d)$ arithmetic and one extra stored vector, and the method stops on the residual of Algorithm 6.1, $\|\nabla f(x^k)\|_2 \leq \varepsilon$.



Figure 10.2: Momentum as a filter on directions. Left: along the valley successive steps agree and accumulate. Right: across the valley they alternate and cancel; with too large a β the alternation is amplified instead.

Why does the extra term help? Momentum acts as a *filter on directions* (Figure 10.2). Along the valley successive gradient steps point the same way, so their contributions to the momentum term *accumulate*: the iterate picks up speed exactly where gradient descent was slow. Across the valley the steps alternate in sign, so they largely *cancel* and the oscillating component is damped, the right medicine for the zig-zag. But the filter has a dial: too much momentum amplifies the steep direction instead of damping it, so β must be designed, not guessed.

The heavy ball has a clear scope. Tuned for a quadratic it is excellent, faster in fact than Nesterov's method on the instance of Example 10.1, but on a general smooth convex function the curvature changes along the path, and parameters aggressive enough to be fast can oscillate or fail to converge: the heavy ball carries *no* universal guarantee. Its role this week is to motivate acceleration (Week 11 reuses (10.2) as stochastic gradient descent with momentum); the workhorse is Nesterov's method, which uses history more carefully and does keep a guarantee.

10.1.3 Nesterov's look-ahead: extrapolate, then correct

Nesterov's key idea is to change *where* the gradient is evaluated. Instead of taking the gradient at x^k and then adding momentum, first extrapolate along the recent motion to a look-ahead point, then take the gradient step there:

$$y^k = x^k + \beta_{k-1}(x^k - x^{k-1}), \quad x^{k+1} = y^k - \alpha \nabla f(y^k), \quad y^0 = x^0. \quad (10.3)$$

The point y^k is the **extrapolated point** (also the look-ahead or driving point) and $\beta_{k-1} \in [0, 1]$ the momentum coefficient attached to the last move $x^k - x^{k-1}$: either a constant β (Section 10.2), when the index is immaterial, or the schedule of Section 10.3, whose value β_k computed at iteration k multiplies the move $x^{k+1} - x^k$ just made (Algorithm 10.2), which is why it appears here with index $k - 1$. Compared with Algorithm 6.1 the only new ingredient is y^k ; compared with the heavy ball, the gradient is read at the look-ahead point, so the correction already knows where the momentum is taking us. With $\alpha = 1/L$ the step $x^{k+1} = y^k - \alpha \nabla f(y^k)$ is still the minimiser of the quadratic upper model of Week 6, now built at y^k , so the descent-lemma logic that justified gradient descent justifies every accelerated step. The method needs nothing beyond the same **gradient oracle**, the routine that returns ∇f at a queried point; it queries a different point.

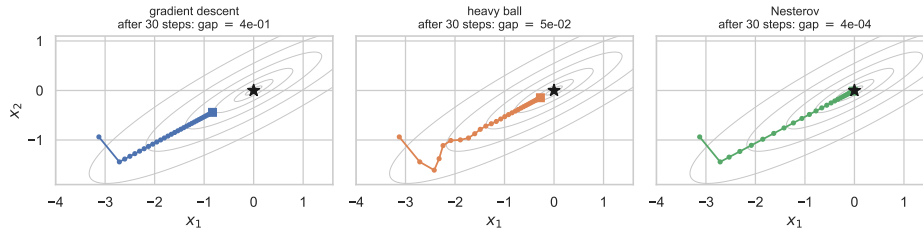
On the quadratic of Example 6.14 the extrapolation overshoots in the steep coordinate, but the gradient step at y^k removes that component at once, while in the flat coordinate it lengthens every step (the exercises work this through by hand); the effect compounds, as the next example shows.

10.1.4 Three methods on one problem

Example 10.1 (The quadratic laboratory: three methods, same start, same step). Let $f(x) = \frac{1}{2}x^\top Qx$ on $\mathbb{R}^2$ with $Q = R \operatorname{diag}(1, 25) R^\top$, R the rotation by 28° , so $Q \approx \begin{pmatrix} 6.290 & -9.948 \\ -9.948 & 19.710 \end{pmatrix}$, $\mu = 1$,

Table 10.1: Optimality gap $f(x^k) - f^*$ in [Example 10.1](#) (same start, same step $\alpha = 1/L$).

k	gradient descent	heavy ball ($\beta = 4/9$)	Nesterov ($\beta = 2/3$)	ratio GD/Nesterov
10	2.26	1.21	0.531	4.3
30	0.442	4.87×10^{-2}	3.85×10^{-4}	1.1×10^3
70	1.69×10^{-2}	7.89×10^{-5}	3.12×10^{-11}	5.4×10^8

Same start, same step $1/L$: momentum reaches the minimiser far sooner**Figure 10.3:** Thirty iterations of gradient descent, the heavy ball and Nesterov's method in [Example 10.1](#), same start, same step $1/L$; the final gaps above each panel are those of [Table 10.1](#).

$L = 25$, $\kappa = 25$, $x^* = 0$ and $f^* = 0$. All methods start from $x^0 \approx (-3.126, -0.937)$, where $f(x^0) = 10.24$, and use $\alpha = 1/L = 0.04$: gradient descent, the heavy ball with $\beta = 4/9$, and Nesterov's update (10.3) with the constant $\beta = 2/3$. [Figure 10.3](#) shows the paths and [Table 10.1](#) the gaps. Gradient descent removes the steep component at once and then creeps along the valley; momentum carries the iterates along it. The difference is orders of magnitude, not percentages: after thirty gradients Nesterov's method is about three digits more accurate than gradient descent, and after seventy more than eight.

The heavy-ball run uses the common step $\alpha = 1/L$ with $\beta = ((\sqrt{\kappa} - 1)/(\sqrt{\kappa} + 1))^2 = 4/9$, not Polyak's tuned pair $\alpha = 4/(\sqrt{L} + \sqrt{\mu})^2$ with the same β , under which every direction would contract by $(\sqrt{\kappa} - 1)/(\sqrt{\kappa} + 1) = 2/3$ and the heavy ball would beat Nesterov's method on this quadratic; that a method can be tuned to be very fast on a quadratic and still have no guarantee elsewhere is exactly why it is not the method of choice. The gaps in [Table 10.1](#) shrink asymptotically by about 0.92, 0.85 and 0.64 per iteration, numbers that [Section 10.2](#) explains.

Key idea

Acceleration keeps the same gradient oracle, projection and prox, and adds a principled use of history: extrapolate to y^k , then correct there. Momentum filters directions, accumulating persistent motion along the valley and cancelling alternating motion across it.

10.2 The quadratic laboratory: why momentum works

To see *why* the methods behave so differently we use a setting where the geometry can be computed exactly, the quadratic

$$f(x) = \frac{1}{2} x^\top Q x, \quad Q \succ 0 \text{ symmetric}, \quad \nabla f(x) = Qx, \quad x^* = 0, \quad f^* = 0, \quad (10.4)$$

where $Q \succ 0$ is positive definiteness in the sense of [Definition 3.11](#), so f is strictly convex ([Proposition 3.14](#)) with Hessian $\nabla^2 f(x) = Q$; near its minimiser any smooth strongly convex function looks like (10.4).

10.2.1 Eigenvalues, eigenvectors and the condition number

Definition 10.2 (Eigenvectors and eigenvalues of a symmetric positive definite matrix). Let $Q \in \mathbb{R}^{2 \times 2}$ be symmetric with $Q \succ 0$. There are two orthogonal unit vectors v_1, v_2 , the **eigenvectors**, along which Q only rescales:

$$Qv_i = \lambda_i v_i, \quad \lambda_i > 0, \quad v_i^\top v_j = 0 \ (i \neq j), \quad \|v_i\|_2 = 1,$$

with λ_i the **eigenvalues**. Every point decomposes as $x = \sum_i (v_i^\top x) v_i$, and the eigenvectors are the *axes* of the elliptical level sets $\{x : \frac{1}{2}x^\top Qx = c\}$, whose semi-axis along v_i has length $\sqrt{2c/\lambda_i}$. We write $\lambda_{\min} = \mu$ for the smaller eigenvalue (the *flat* direction) and $\lambda_{\max} = L$ for the larger (the *steep* direction).

The semi-axis formula follows by putting $x = sv_i$ into $\frac{1}{2}x^\top Qx = c$. The letters μ and L are those of [Week 6](#): since $\nabla^2 f = Q$, the smoothness constant of [Definition 6.8](#) and the strong-convexity constant of [Definition 6.12](#) are exactly $\lambda_{\max}$ and $\lambda_{\min}$, and $\mu I \preceq Q \preceq LI$. More generally, any symmetric $Q \in \mathbb{R}^{d \times d}$ has d orthonormal eigenvectors with real eigenvalues, all ≥ 0 when $Q \succeq 0$; a zero eigenvalue is a direction with no curvature, so $\mu = 0$. This is the sense in which [Week 9](#) and this week speak of the largest eigenvalue of $A^\top A$, the constant L of least squares, and of its smallest, which is zero whenever A has more columns than rows. In [Example 10.1](#) the eigenvectors are the rotated axes, $\lambda_1 = 1$ and $\lambda_2 = 25$ ([Figure 10.1](#), right). Within this subsection λ_i is an eigenvalue; from [Section 10.4](#) on, λ is again the regularisation strength of the lasso.

Definition 10.3 (Condition number). The **condition number** of the quadratic (10.4), and more generally of an L -smooth μ -strongly convex function, is $\kappa = L/\mu = \lambda_{\max}/\lambda_{\min} \geq 1$. It measures the elongation of the level sets: $\kappa = 1$ gives circles, a large κ a long thin valley. A problem with large κ is called **ill-conditioned** (or badly conditioned).

10.2.2 Gradient descent, one mode at a time

In the eigenvector basis the gradient iteration falls apart into independent scalar recursions, one per direction; each direction is called a **mode**.

Proposition 10.4 (Gradient descent on a quadratic decouples). *Let f be the quadratic (10.4) and write the iterates of gradient descent $x^{k+1} = x^k - \alpha \nabla f(x^k)$ in the eigenvector basis, $x^k = \sum_i c_i^k v_i$ with $c_i^k = v_i^\top x^k$. Then*

$$c_i^{k+1} = (1 - \alpha \lambda_i) c_i^k, \quad \text{hence} \quad c_i^k = (1 - \alpha \lambda_i)^k c_i^0. \quad (10.5)$$

Consequently: (i) the iteration converges for every x^0 if and only if $|1 - \alpha \lambda_i| < 1$ for all i , that is, if and only if $0 < \alpha < 2/L$; (ii) with $\alpha = 1/L$ the steep mode is removed in one step and the flat mode contracts by $1 - \mu/L = 1 - 1/\kappa$ per step.

Proof. Substituting the expansion into the update and using $Qv_i = \lambda_i v_i$,

$$x^{k+1} = x^k - \alpha Qx^k = \sum_i c_i^k v_i - \alpha \sum_i c_i^k \lambda_i v_i = \sum_i (1 - \alpha \lambda_i) c_i^k v_i,$$

and comparing coefficients gives (10.5). For (i), $(1 - \alpha \lambda_i)^k \rightarrow 0$ if and only if $|1 - \alpha \lambda_i| < 1$; as $\alpha \lambda_i > 0$ this means $\alpha \lambda_i < 2$ for every i , and the binding case is $\lambda_i = L$. For (ii) put $\alpha = 1/L$. $\square$

This explains the crawl. Each direction contracts by its own factor $|1 - \alpha \lambda_i|$, and the *worst* of them sets the overall speed: stability in the steep direction limits the step to $\alpha < 2/L$, the safe choice is $\alpha = 1/L$, and with it the flat direction contracts by only $1 - 1/\kappa$, painfully close to one for large κ ; this is the quadratic face of the rate $(1 - \mu/L)^k$ of [Theorem 6.13](#). In [Example 10.1](#) the factor is 0.96 per step, $0.96^2 = 0.92$ on the objective (the objective is quadratic in the coordinates, so it gains the factor twice), the asymptotic decay of the gradient-descent column of [Table 10.1](#).

Key idea

The condition number $\kappa = L/\mu$, not the number of variables, is what makes gradient descent slow: the step is limited by the steepest mode (L), progress by the flattest (μ).

The same decoupling shows why momentum works. Apply (10.3) with a constant β to the quadratic. In mode i the extrapolated coordinate is $c_i^k + \beta(c_i^k - c_i^{k-1})$ and the gradient step multiplies it by $1 - \alpha\lambda_i$, so

$$c_i^{k+1} = (1 - \alpha\lambda_i)[(1 + \beta)c_i^k - \beta c_i^{k-1}], \quad (10.6)$$

a two-term linear recursion whose solutions decay like r^k where r solves $r^2 - (1 - \alpha\lambda_i)(1 + \beta)r + (1 - \alpha\lambda_i)\beta = 0$.

Example 10.5 (Momentum mode by mode). In Example 10.1, $\alpha = 1/25$ and $\beta = 2/3$. For the flat mode $\lambda_i = 1$ the factor is 0.96 and the characteristic equation is $r^2 - 0.96 \cdot \frac{5}{3}r + 0.96 \cdot \frac{2}{3} = r^2 - 1.6r + 0.64 = (r - 0.8)^2$: a double root at $r = 0.8 = 1 - 1/\sqrt{\kappa}$. The flat mode now decays like 0.8^k (times a polynomial factor, as the root is double) instead of 0.96^k ; on the objective the factor is 0.64 against 0.92, the asymptotic decay of the Nesterov column of Table 10.1. The other modes are faster still (modulus 0.730 for $\lambda_i = 5$, 0.577 for $\lambda_i = 12.5$, and 0 for the steep mode). For the heavy ball (10.2) the same computation gives $r^2 - (1 - \alpha\lambda_i + \beta)r + \beta = 0$; with $\alpha = 1/L$ and $\beta = 4/9$ the slowest mode has modulus 0.923 (0.85 on the objective, the middle column of Table 10.1), while Polyak's tuned pair gives $2/3$ in every mode. Momentum changes only the recursion each mode obeys, and a well-chosen coefficient moves the slowest contraction from $1 - 1/\kappa$ to $1 - 1/\sqrt{\kappa}$.

10.2.3 Two geometric scales

Away from quadratics the constants L and μ still define two scales at every point y . Smoothness gives a cautious *upper* model of f with curvature L , minimised by the short step $y - \nabla f(y)/L$; strong convexity gives a *lower* model with curvature μ , whose minimiser $y - \nabla f(y)/\mu$ is a much longer target. For $f(x) = 0.15x^4 + 0.5x^2 + 0.1$ on $[-1.7, 1.7]$, where $f''(x) = 1.8x^2 + 1 \in [1, 6.2]$ so that $\mu = 1$ and $L = 7$ are valid constants, the point $y = 1$ ($f(y) = 0.75$, $f'(y) = 1.6$) has its upper model $0.75 + 1.6(x - 1) + 3.5(x - 1)^2$ minimised at $y - f'(y)/L = 0.771$, a quarter of the way to $x^* = 0$, and its lower model $0.75 + 1.6(x - 1) + 0.5(x - 1)^2$ minimised at $y - f'(y)/\mu = -0.6$, beyond x^* . Gradient descent takes the short step. Accelerated strongly convex methods can be read as mixing the two scales: the rate $1 - \sqrt{\mu/L}$ obtained below is what a step of the intermediate length $1/\sqrt{\mu L}$, the geometric mean of $1/L$ and $1/\mu$, would achieve on the flat mode, but such a step has $\alpha L = \sqrt{\kappa} > 2$ once $\kappa > 4$ and diverges in the steep mode (Proposition 10.4(i)), so the mixing has to be done through momentum, mode by mode, and not through a longer step.

10.2.4 The strongly convex rate and what the square root buys

The mode computation of Example 10.5 generalises from quadratics to all smooth strongly convex functions; the proof is beyond this unit, the statement is not.

Theorem 10.6 (Accelerated gradient, strongly convex case). *Let $f : \mathbb{R}^d \rightarrow \mathbb{R}$ be L -smooth and μ -strongly convex with $\mu > 0$, let $\kappa = L/\mu$, and run the accelerated update with $\alpha = 1/L$ and the constant momentum coefficient*

$$\beta = \frac{\sqrt{\kappa} - 1}{\sqrt{\kappa} + 1}, \quad y^0 = x^0, \quad x^{k+1} = y^k - \frac{1}{L}\nabla f(y^k), \quad y^{k+1} = x^{k+1} + \beta(x^{k+1} - x^k). \quad (10.7)$$

Then for every $k \geq 0$

$$f(x^k) - f^* \leq \left(1 - \frac{1}{\sqrt{\kappa}}\right)^k \left(f(x^0) - f^* + \frac{\mu}{2} \|x^0 - x^*\|_2^2\right) \leq 2\left(1 - \frac{1}{\sqrt{\kappa}}\right)^k (f(x^0) - f^*),$$

so $f(x^k) - f^* = O((1 - 1/\sqrt{\kappa})^k)$, against gradient descent's $f(x^k) - f^* \leq (1 - 1/\kappa)^k(f(x^0) - f^*)$ (Theorem 6.13).

The second inequality uses strong convexity at x^* , where $\nabla f(x^*) = 0$: $f(x^0) - f^* \geq \frac{\mu}{2} \|x^0 - x^*\|_2^2$. The gain is a condition-number improvement, κ becoming $\sqrt{\kappa}$ in the headline iteration count. To reduce the gap by a factor ε we need $(1 - 1/\kappa)^k \leq \varepsilon$ or, up to the constant 2 in the accelerated bound (worth only $\sqrt{\kappa} \ln 2$ extra iterations), $(1 - 1/\sqrt{\kappa})^k \leq \varepsilon$, and since $(1 - 1/\kappa)^k \leq e^{-k/\kappa}$ it suffices to take

$$k \geq \kappa \ln(1/\varepsilon) \quad (\text{gradient descent}), \quad k \geq \sqrt{\kappa} \ln(1/\varepsilon) \quad (\text{accelerated}). \quad (10.8)$$

Example 10.7 (What the square root buys). Take $\mu = 1$, $L = 100$, so $\kappa = 100$, and ask for six correct digits, $\varepsilon = 10^{-6}$, $\ln(1/\varepsilon) = 13.8$. Gradient descent's slowest mode contracts by $1 - 1/\kappa = 0.99$ per step and (10.8) asks for $\kappa \ln(1/\varepsilon) = 1382 \approx 1400$ iterations (solving $0.99^k = 10^{-6}$ exactly gives 1375). The accelerated factor is $1 - 1/\sqrt{\kappa} = 0.90$ and $\sqrt{\kappa} \ln(1/\varepsilon) = 138 \approx 140$ iterations suffice (exactly 132): about ten times fewer gradients for the same accuracy, the ratio of the counts being $\sqrt{\kappa} = 10$. The harder the problem, the bigger the prize.

Remark 10.8 (Acceleration is optimal, not merely clever). In the worst case over L -smooth μ -strongly convex functions (of dimension large relative to k), no method that uses only gradient evaluations can reduce the error by ε in fewer than a constant times $\sqrt{\kappa} \ln(1/\varepsilon)$ iterations, and over L -smooth convex functions none can beat the order $1/k^2$ of Theorem 10.11. This is Nesterov's lower bound, not proved here: further speed needs more structure, not more history.

Note on the lecture slides

Frames 10.14 and 10.20 state the accelerated rates without hypotheses, and frames 10.24 and 10.26 do not say where f or g must be smooth. Theorems 10.6, 10.11 and 10.14 state them in full: convexity or strong convexity, L -smoothness on all of $\mathbb{R}^d$ (since y^k can leave the feasible set, Remark 10.13), a minimiser, $\alpha = 1/L$ and the momentum rule.

10.3 Rates and schedules for general convex problems

The constant coefficient (10.7) needs $\mu > 0$ and needs it to be known. Least squares with more unknowns than observations, the lasso, and most problems of Week 9 are convex but not strongly convex, or have a μ nobody can estimate; for them the momentum coefficient follows a *schedule* that grows with k .

10.3.1 The momentum schedule

Definition 10.9 (Momentum schedule). The **momentum schedule** is the sequence $\theta_0 = 1$,

$$\theta_{k+1} = \frac{1 + \sqrt{1 + 4\theta_k^2}}{2}, \quad \beta_k = \frac{\theta_k - 1}{\theta_{k+1}}, \quad k = 0, 1, 2, \dots \quad (10.9)$$

Equivalently, θ_{k+1} is the positive root of $\theta_{k+1}^2 - \theta_{k+1} = \theta_k^2$.

The first values are $\theta_k = 1, 1.618, 2.194, 2.750, 3.295, 3.833, \dots$ with $\theta_{10} = 6.463$ and $\theta_{40} = 21.77$, and $\beta_k = 0, 0.282, 0.434, 0.531, 0.599, 0.649, \dots$ with $\beta_{10} = 0.782$, $\beta_{20} = 0.875$ and $\beta_{40} = 0.932$. The schedule starts cautiously, $\beta_0 = 0$, and then allows longer look-ahead: β_k rises towards 1 (roughly like $k/(k+3)$), trusting more of the recent direction of travel as the run settles. It changes the *extrapolation length*, never the gradient oracle.

Lemma 10.10 (Growth of the schedule). For the schedule (10.9), $\theta_k \geq (k+2)/2$ for all $k \geq 0$, hence $\theta_k^2 \geq (k+2)^2/4 \geq (k+1)^2/4$; moreover $0 \leq \beta_k < 1$.

Algorithm 10.2 Nesterov's accelerated gradient method (AGD)

Require: a routine for ∇f ; starting point x^0 ; step size $\alpha = 1/L$ (or backtracking, [Section 10.3.3](#)); tolerance $\varepsilon > 0$; iteration cap K

Ensure: an approximate minimiser of f

```

1:  $y^0 \leftarrow x^0$ ;  $\theta_0 \leftarrow 1$ 
2: for  $k = 0, 1, \dots, K-1$  do
3:    $x^{k+1} \leftarrow y^k - \alpha \nabla f(y^k)$   $\triangleright$  gradient step at the extrapolated point
4:    $\theta_{k+1} \leftarrow \frac{1}{2}(1 + \sqrt{1 + 4\theta_k^2})$ ;  $\beta_k \leftarrow (\theta_k - 1)/\theta_{k+1}$   $\triangleright$  or  $\beta_k \leftarrow \frac{\sqrt{k}-1}{\sqrt{k}+1}$  if  $\mu > 0$  is known
5:    $y^{k+1} \leftarrow x^{k+1} + \beta_k (x^{k+1} - x^k)$   $\triangleright$  extrapolate along the move just made
6:   if  $\|\nabla f(x^{k+1})\|_2 \leq \varepsilon$  then return  $x^{k+1}$   $\triangleright$  residual of Algorithm 6.1; costs a gradient, so
       test every few iterations
7:   end if
8: end for
9: return  $x^K$ 

```

Proof. Since $\sqrt{1 + 4\theta_k^2} \geq 2\theta_k$, the recursion gives $\theta_{k+1} \geq \frac{1}{2}(1 + 2\theta_k) = \theta_k + \frac{1}{2}$, and $\theta_0 = 1$ starts the induction. The same inequality gives $\theta_{k+1} > \theta_k - 1 \geq 0$, so $0 \leq \beta_k < 1$. $\square$

The convergence argument sketched below bounds the gap after k steps by a constant over θ_k^2 , which the lemma makes $O(1/k^2)$ (at $k = 40$, $\theta_k^2 = 473.8$ against the comparator 420.25); the schedule is not ad hoc tuning but what makes that argument close.

[Algorithm 10.2](#) assembles the method. Each iteration costs one gradient and $O(d)$ arithmetic, as in [Algorithm 6.1](#), plus one extra stored vector. The coefficient $\beta_k = (\theta_k - 1)/\theta_{k+1}$ is applied to the move $x^{k+1} - x^k$ just made, as in [\(10.3\)](#). With the schedule the first two gradient steps are plain ($\beta_0 = 0$, so $y^1 = x^1$) and momentum first acts in computing x^3 , through $y^2 = x^2 + 0.282(x^2 - x^1)$, whereas the constant coefficient of [Theorem 10.6](#) is present from the second step at full strength. In prose the index-free pattern $y = x + \beta(x - x^-)$, with x^- the previous iterate, is all one needs.

Note on the lecture slides

Frames 10.7, 10.16, 10.24 and 10.26 pair $\beta_k = (\theta_k - 1)/\theta_{k+1}$ with the display $y^k = x^k + \beta_k(x^k - x^{k-1})$, so that $\beta_1 = 0.282$ already multiplies $x^1 - x^0$. The companion notebook, which produced every figure of the lecture, and the published theorem quoted on frame 10.20 attach β_k to $x^{k+1} - x^k$, as [Algorithm 10.2](#) and [\(10.3\)](#) do. The variants differ by a one-step shift, both are $O(1/k^2)$, and their gaps on [Example 10.15](#) agree to within about ten per cent early on and a few per cent after $k = 100$; the notes use the notebook's form, for which the stated constants are the published ones.

10.3.2 The $O(1/k^2)$ guarantee

The baseline rate for structured first-order methods has been $O(1/k)$ ([Theorem 6.11](#), [Theorem 9.18](#)); acceleration keeps the cheap iterations and improves the headline.

Theorem 10.11 (Accelerated gradient, convex case). *Let $f : \mathbb{R}^d \rightarrow \mathbb{R}$ be convex and L -smooth, with a minimiser x^* . Run [Algorithm 10.2](#) with $\alpha = 1/L$ and the schedule [\(10.9\)](#). Then for every $k \geq 1$*

$$f(x^k) - f^* \leq \frac{2L\|x^0 - x^*\|_2^2}{(k+1)^2} = O(1/k^2). \quad (10.10)$$

The full proof is out of scope; the idea is not, and it explains the schedule. Writing $R = \|x^0 - x^*\|_2$, one shows that the weighted quantity $\theta_{k-1}^2(f(x^k) - f^*) + \frac{L}{2}\|u^k\|_2^2$, with

$u^k = \theta_{k-1}x^k - (\theta_{k-1} - 1)x^{k-1} - x^*$, never increases: the descent lemma at y^k and the convexity of f produce terms that cancel *because* $\theta_{k+1}^2 - \theta_k^2 = \theta_k^2$, which is the whole reason for (10.9). At $k = 1$ the quantity is at most $\frac{L}{2}R^2$, so $f(x^k) - f^* \leq LR^2/(2\theta_{k-1}^2)$, and Lemma 10.10 with $\theta_{k-1} \geq (k+1)/2$ gives (10.10).

In practice the bound is read as an iteration count. To guarantee $f(x^k) - f^* \leq \varepsilon$, Theorem 6.11 with step $1/L$ needs $k \geq LR^2/(2\varepsilon)$ iterations, (10.10) needs $k \geq \sqrt{2LR^2/\varepsilon} - 1$: the dependence on the accuracy improves from $1/\varepsilon$ to $1/\sqrt{\varepsilon}$ at the same cost per iteration. On the benchmark of Example 10.15, where $LR^2 \approx 3786$, the two counts for $\varepsilon = 10^{-2}$ are about 189,000 and 870; worst-case bounds are pessimistic on any particular instance, but the orders they predict are what the measured curves show.

10.3.3 Step sizes and unknown smoothness

Acceleration relies on the same smooth upper bound as Weeks 6 and 9. If L is known, use $\alpha = 1/L$. If L is uncertain, choose the step by *backtracking*: start from a guess of L , form the accelerated step, and check whether the quadratic upper model built at y^k is valid at the new point,

$$f(x^{k+1}) \leq f(y^k) + \nabla f(y^k)^\top (x^{k+1} - y^k) + \frac{1}{2\alpha} \|x^{k+1} - y^k\|_2^2, \quad x^{k+1} = y^k - \alpha \nabla f(y^k); \quad (10.11)$$

if it fails, increase the estimate of L (shrink α by a fixed factor) and retry, keeping the accepted estimate as the starting guess of the next iteration so that the step never grows. Since $x^{k+1} - y^k = -\alpha \nabla f(y^k)$, the right-hand side is $f(y^k) - \frac{\alpha}{2} \|\nabla f(y^k)\|_2^2$: the test asks for the decrease the descent lemma guarantees when $\alpha \leq 1/L$, so it accepts after finitely many shrinks. It is the Armijo-type test of Algorithm 6.2 with constant $\frac{1}{2}$, applied at y^k rather than at x^k , and with non-increasing steps the $O(1/k^2)$ guarantee survives with the accepted steps in place of $1/L$; the composite version is (10.17). The smooth-case story is now complete, and it is the moment to accelerate the methods of Weeks 8 and 9.

10.4 Accelerating projection and prox

10.4.1 Accelerated projected gradient

The projected-gradient method of Algorithm 8.2 minimises a smooth convex f over a closed convex set C with a cheap projector Π_C (Definition 8.10) by $x^{k+1} = \Pi_C(x^k - \alpha \nabla f(x^k))$: a gradient step, then the nearest feasible point. Acceleration changes only the point presented to the projector,

$$y^k = x^k + \beta_{k-1}(x^k - x^{k-1}), \quad x^{k+1} = \Pi_C(y^k - \alpha \nabla f(y^k)), \quad y^0 = x^0, \quad (10.12)$$

with β_{k-1} from the schedule (10.9), indexed as in (10.3) and Algorithm 10.2. The feasible set, the projection and the Week 8 geometry (the fixed-point characterisation of optimality, the projected-gradient mapping) are untouched; only the driving point changes. The **accelerated projected-gradient method** (APG) costs one gradient and one projection per iteration, like its parent. A recurring example is least squares over the simplex.

Model 10.12 (Simplex-constrained least squares).

$$\begin{aligned} \min_{x \in \mathbb{R}^d} \quad & \frac{1}{2} \|Ax - b\|_2^2 \\ \text{s.t.} \quad & \mathbf{1}^\top x = 1, \quad x \geq 0. \end{aligned} \quad (10.13)$$

The decision variable x is a vector of nonnegative shares summing to one, the feasible set is the simplex of Definition 2.12, the long-only portfolio set of Model 1.2 met again in Week 8, and

Algorithm 10.3 FISTA (accelerated proximal gradient); $h = I_C$ gives accelerated projected gradient, $h = 0$ gives [Algorithm 10.2](#)

Require: routines for ∇g and $\text{prox}_{\alpha h}$; starting point $x^0 \in \text{dom } h$; step size $\alpha = 1/L$ (or backtracking, [Section 10.5.2](#)); tolerance $\varepsilon > 0$; iteration cap K ; restart flag (optional)

Ensure: an approximate minimiser of $F = g + h$

```

1:  $y^0 \leftarrow x^0$ ;  $\theta_0 \leftarrow 1$ 
2: for  $k = 0, 1, \dots, K - 1$  do
3:    $x^{k+1} \leftarrow \text{prox}_{\alpha h}(y^k - \alpha \nabla g(y^k))$   $\triangleright$  gradient step at  $y^k$ , then the structured correction
4:   if restart and  $F(x^{k+1}) > F(x^k)$  then  $\triangleright$  objective restart, Section 10.5.1
5:      $\theta_k \leftarrow 1$ ;  $x^{k+1} \leftarrow \text{prox}_{\alpha h}(x^k - \alpha \nabla g(x^k))$   $\triangleright$  drop the momentum: a plain
       Algorithm 9.1 step
6:   end if
7:    $\theta_{k+1} \leftarrow \frac{1}{2}(1 + \sqrt{1 + 4\theta_k^2})$ ;  $\beta_k \leftarrow (\theta_k - 1)/\theta_{k+1}$ 
8:    $y^{k+1} \leftarrow x^{k+1} + \beta_k(x^{k+1} - x^k)$   $\triangleright$  extrapolate
9:   if  $\|\tilde{G}_\alpha(x^{k+1})\|_2 \leq \varepsilon$  then return  $x^{k+1}$   $\triangleright$  proximal-gradient mapping, Section 10.5.3
10:  end if
11: end for
12: return  $x^K$ 

```

the objective is the least-squares fit of [Model 6.16](#), read as tracking: weights whose combined return series Ax is as close as possible to a target b . This convex quadratic over a polyhedron is solved by (10.12) with $\nabla f(x) = A^\top(Ax - b)$, L the largest eigenvalue of $A^\top A$ and the sort-based projector of [Proposition 8.13](#); every iterate is feasible, and the solution typically has many exact zeros because the projection truncates ([Example 10.16](#)).

Remark 10.13 (The extrapolated point may be infeasible). The iterates x^k of (10.12) lie in C , but y^k need not: with $C = [0, 1]$, $x^{k-1} = 0.5$, $x^k = 0$ and a momentum coefficient of 0.5 we get $y^k = -0.25 \notin C$. Since the gradient is evaluated at y^k , f must be differentiable and L -smooth on all of $\mathbb{R}^d$ (or on a neighbourhood of C containing every y^k), not merely on C . This holds for the least-squares objectives used this week; it fails for a logarithmic barrier, which blows up at the boundary of C ([Week 13](#)). The same caveat applies to FISTA with $\text{dom } h$ for C .

10.4.2 FISTA: the accelerated proximal gradient method

Projection is one special proximal operator: by [Proposition 9.8](#), $\text{prox}_{\alpha I_C} = \Pi_C$ for the indicator I_C of [Definition 8.4](#). Reading (10.12) through that identity, the three smooth-part-plus-correction updates of the unit become three cases of one accelerated formula for the composite problem $\min F(x) = g(x) + h(x)$ of [Model 9.2](#), with g convex and L -smooth and h convex with a cheap prox ([Definition 9.6](#)):

$$y^k = x^k + \beta_{k-1}(x^k - x^{k-1}), \quad x^{k+1} = \text{prox}_{\alpha h}(y^k - \alpha \nabla g(y^k)), \quad y^0 = x^0. \quad (10.14)$$

With $h = 0$ this is [Algorithm 10.2](#); with $h = I_C$ it is (10.12); with a general convex h it is the accelerated analogue of the proximal gradient method [Algorithm 9.1](#), called **FISTA**, the *fast iterative shrinkage-thresholding algorithm*, after the soft-thresholding prox of its original application. [Algorithm 10.3](#) states the method, including the optional restart test of [Section 10.5.1](#) and a stopping test on the proximal-gradient mapping of [Definition 9.15](#), written $\tilde{G}_\alpha$ this week ([Section 10.5.3](#)). One formula unifies smooth unconstrained, simple-constraint and regularised composite convex optimisation.

10.4.3 The lasso as the canonical FISTA example

The lasso of [Model 9.3](#), written in x because β is now the momentum coefficient,

$$\min_{x \in \mathbb{R}^d} \frac{1}{2} \|Ax - b\|_2^2 + \lambda \|x\|_1, \quad (10.15)$$

has smooth part $g(x) = \frac{1}{2} \|Ax - b\|_2^2$ with $\nabla g(x) = A^\top(Ax - b)$ and L the largest eigenvalue of $A^\top A$, and nonsmooth part $h(x) = \lambda \|x\|_1$ whose prox is soft-thresholding, $\text{prox}_{\alpha h} = \text{soft}_{\alpha\lambda}$ ([Proposition 9.11](#)); here $\lambda > 0$ is the regularisation strength of [Week 9](#), not an eigenvalue. The FISTA step is

$$x^{k+1} = \text{soft}_{\alpha\lambda}(y^k - \alpha A^\top(Ay^k - b)), \quad (10.16)$$

ISTA's update of [Algorithm 9.1](#) with y^k in place of x^k : the same least-squares gradient, the same soft-thresholding, a look-ahead point in between, at the same cost of two matrix–vector products and an $O(d)$ threshold per iteration.

Theorem 10.14 (FISTA rate). *Let $g : \mathbb{R}^d \rightarrow \mathbb{R}$ be convex and L -smooth, let h be closed convex with a computable proximal operator, and suppose $F = g + h$ has a minimiser x^* . Run [Algorithm 10.3](#) without restart, with $\alpha = 1/L$ and the schedule (10.9). Then for every $k \geq 1$*

$$F(x^k) - F^* \leq \frac{2L \|x^0 - x^*\|_2^2}{(k+1)^2} = O(1/k^2),$$

whereas ISTA ([Theorem 9.18](#) with $\alpha = 1/L$) guarantees $F(x^k) - F^* \leq L \|x^0 - x^*\|_2^2 / (2k) = O(1/k)$. With $h = I_C$ the bound covers the accelerated projected-gradient method (10.12); $h = 0$ recovers [Theorem 10.11](#).

The constant is the same as in the smooth case and the proof is the same argument with the descent lemma applied to g and the prox handled through its optimality condition; we omit it. ISTA and FISTA use the same prox; the acceleration comes from the extrapolated point alone ([Example 10.17](#)).

10.4.4 One ladder, many methods

[Table 10.2](#) collects the first-order methods of [Weeks 6 to 9](#) and this week with their headline rates for convex problems. In the update column x^- and x^+ are the previous and next iterates; in the first row g denotes a subgradient, as in [Week 7](#), whereas in the proximal rows g is the smooth part of the objective.

Reading down the table, every gain comes from *structure*, not from a cleverer step: smoothness buys $O(1/k)$, and extrapolation on top of smoothness buys $O(1/k^2)$, whether the correction after the gradient step is nothing, a projection or a prox. The subgradient method is the exception that proves the rule: with no smooth part to extrapolate, momentum has nothing to work with, and by [Remark 7.17](#) the order $O(1/\sqrt{k})$ is already optimal for methods that see only subgradients; speed there is recovered by exposing composite structure, the route from [Week 9](#) to FISTA. [Week 13](#) extends this table.

Key idea

One formula, $x^+ = \text{prox}_{\alpha h}(y - \alpha \nabla g(y))$ with $y = x + \beta(x - x^-)$, accelerates smooth, constrained and composite convex problems alike: same oracle, same prox, one extra vector of memory, and $O(1/k^2)$ in place of $O(1/k)$.

Table 10.2: One ladder, many methods: the first-order methods of the unit and what buys their rate.

Method	Update	Structure exploited	Headline rate
Subgradient (Week 7)	$x^+ = x - \alpha_k g, g \in \partial f(x)$	none: general nonsmooth	$O(1/\sqrt{k})$, not accelerable
Gradient descent (Week 6)	$x^+ = x - \alpha \nabla f(x)$	smooth, unconstrained	$O(1/k)$
Projected gradient (Week 8)	$x^+ = \Pi_C(x - \alpha \nabla f(x))$	smooth + simple set	$O(1/k)$
Proximal gradient (Week 9)	$x^+ = \text{prox}_{\alpha h}(x - \alpha \nabla g(x))$	smooth + cheap prox	$O(1/k)$
Accelerated gradient	$x^+ = y - \alpha \nabla f(y)$ with $y = x + \beta(x - x^-)$	smooth, unconstrained	$O(1/k^2)$
Accelerated projected gradient	$x^+ = \Pi_C(y - \alpha \nabla f(y))$	smooth + simple set	$O(1/k^2)$
FISTA	$x^+ = \text{prox}_{\alpha h}(y - \alpha \nabla g(y))$	smooth + cheap prox	$O(1/k^2)$

10.5 Practice: restart, stopping and benchmarks

Acceleration matters most when gradients, projections or prox operators are cheap *relative to the number of iterations needed*, the large-scale regime in which first-order methods are attractive in the first place: large regression problems, simple portfolio constraints with the $O(d \log d)$ simplex projector, and sparse modelling with the $O(d)$ soft-threshold. Where a single gradient is itself the bottleneck (huge N) or the projection is expensive, other ideas take over (Weeks 11 and 13).

10.5.1 Acceleration is not monotone: restart

Gradient descent with $\alpha \leq 1/L$ and the proximal gradient method decrease the objective at every step (Proposition 6.9, Theorem 9.18). Accelerated methods do not: the guarantee of Theorems 10.11 and 10.14 is a faster *overall* decay, not monotone progress at every step. The extrapolation can overshoot, the objective briefly rises, and the gradient step at y^k then corrects it: the ripples in Figures 10.4 and 10.5. In Example 10.15 the gap rose at 13 of the first 150 iterations; in Example 10.17 plain FISTA rose at 89 of 400. (“Monotone” here means non-increasing objective values, not the monotone-gradient property of Week 3.)

The simplest stabiliser is **objective restart**, the optional test in Algorithm 10.3: after computing the trial point x^{k+1} , if $F(x^{k+1}) > F(x^k)$ then drop the momentum, reset $\theta_k \leftarrow 1$ so that the schedule begins again from $\beta = 0$, and recompute x^{k+1} as a plain proximal-gradient step from x^k , which cannot increase F when $\alpha \leq 1/L$; momentum then rebuilds on a fresh trajectory. Restart changes the path, not the problem, and on strongly convex problems it recovers most of the gap between the schedule, which does not know μ , and the constant coefficient of Theorem 10.6, which does. A more refined rule (O’Donoghue and Candès) restarts when the momentum direction opposes the step actually taken from y^k , $(y^k - x^{k+1})^\top (x^{k+1} - x^k) > 0$, which for $h = 0$ reads $\nabla g(y^k)^\top (x^{k+1} - x^k) > 0$; it needs no objective evaluations.

10.5.2 Backtracking for accelerated methods

When L is unknown, use the backtracking test of Section 9.4 with the extrapolated point in place of the iterate: with $x^{k+1} = \text{prox}_{\alpha h}(y^k - \alpha \nabla g(y^k))$, accept the current α if

$$g(x^{k+1}) \leq g(y^k) + \nabla g(y^k)^\top (x^{k+1} - y^k) + \frac{1}{2\alpha} \|x^{k+1} - y^k\|_2^2, \quad (10.17)$$

and otherwise shrink α , recompute and retry, carrying the accepted estimate forward as in Section 10.3.3. As in Week 9 the test passes once $\alpha \leq 1/L$, so the loop terminates, and the rate

of [Theorem 10.14](#) holds with the accepted, non-increasing steps. Backtracking is not a heuristic patch: it enforces the smoothness model behind the step.

10.5.3 How to stop an accelerated method

Acceleration changes the path to the solution, not the optimality condition that certifies it, so the stopping residuals are those of the parent methods, evaluated at the iterate x^k (the point we would return), not at y^k . For the unconstrained method the residual is $\|\nabla f(x^k)\|_2$, the first-order condition of [Theorem 5.1](#) monitored in [Algorithm 6.1](#); for the projected method it is the projected-gradient mapping $G_\alpha(x) = \frac{1}{\alpha}(x - \Pi_C(x - \alpha \nabla f(x)))$ of [Definition 8.16](#); for FISTA it is the proximal-gradient mapping $\tilde{G}_\alpha(x) = \frac{1}{\alpha}(x - \text{prox}_{\alpha h}(x - \alpha \nabla g(x)))$ of [Definition 9.15](#), which vanishes exactly when $0 \in \nabla g(x) + \partial h(x)$ ([Week 9](#) writes G_α for both; the tilde only separates the two). Each residual costs an extra gradient and prox at x^k : the identity $\|\tilde{G}_\alpha(x^k)\|_2 = \|x^k - x^{k+1}\|_2 / \alpha$ that made the test of [Algorithm 9.1](#) free holds only when the step is taken from x^k , and an accelerated step is taken from y^k . A small residual is the certificate; a small change $\|x^{k+1} - x^k\|_2$ is not, and for accelerated methods the objective is a poor progress monitor because it need not decrease.

10.5.4 Three benchmarks and the coding lane

The three examples use one data set: $A \in \mathbb{R}^{120 \times 60}$ with independent standard normal entries and column j scaled by $1/\sqrt{j}$, so that the columns have very different sizes, and $b \in \mathbb{R}^{120}$ standard normal (seed 0). For $g(x) = \frac{1}{2} \|Ax - b\|_2^2$ the eigenvalues of $A^\top A$ give $L = 132.1$ and $\mu = 0.315$, so $\kappa \approx 420$: ill-conditioned least squares. All runs use $\alpha = 1/L$ and the schedule (10.9); gaps are measured against reference solutions computed to high accuracy.

Example 10.15 (Benchmark 1: least squares). Minimise g without constraints from $x^0 = 0$ with gradient descent ([Algorithm 6.1](#)) and [Algorithm 10.2](#); [Figure 10.4](#) (left) shows the gaps. Here $f^* = 29.93$, $f(x^0) - f^* = 31.34$ and $R^2 = \|x^0 - x^*\|_2^2 = 28.65$. At $k = 100$ the gaps are 2.14 for gradient descent and 1.91×10^{-2} for the accelerated method, at $k = 299$ they are 0.319 and 3.32×10^{-4} . The worst-case bounds at $k = 100$ are $LR^2/(2k) = 18.9$ and $2LR^2/(k+1)^2 = 0.742$: both curves lie well below their bounds (the problem is strongly convex, so both eventually decay geometrically), but the ratio between the methods, two orders of magnitude at $k = 100$, is what the change of order from $1/k$ to $1/k^2$ predicts. The accelerated curve is not monotone, as [Section 10.5.1](#) explained.

Example 10.16 (Benchmark 2: simplex-constrained least squares). Solve [Model 10.12](#) with the same data by projected gradient ([Algorithm 8.2](#)) and by (10.12), both from $x^0 = \Pi_\Delta(0) = \mathbf{1}/60$ with the projector of [Proposition 8.13](#) ([Figure 10.4](#), right). The reference optimal value is $f^* = 56.51$ and the solution has 9 nonzero weights out of 60. At $k = 60$ the gaps are 2.8×10^{-3} for projected gradient and 2.9×10^{-5} for its accelerated version, at $k = 119$ they are 1.7×10^{-4} and 1.3×10^{-6} . The projection is identical in both methods; only the driving point changes: the cleanest bridge from [Week 8](#) to acceleration, on exactly the long-only portfolio set.

Example 10.17 (Benchmark 3: lasso with and without restart). Solve the lasso (10.15) with the same data and $\lambda = 0.02$, from $x^0 = 0$, by ISTA, by FISTA and by FISTA with objective restart. The reference value $F^* = 30.62715$ is confirmed by MOSEK through the split-variable RSOME model of [Week 9](#). [Figure 10.5](#) shows 400 iterations: at $k = 200$ the gaps are 0.634, 7.43×10^{-4} and 7.36×10^{-7} , at $k = 399$ they are 0.136, 1.33×10^{-5} and 3.26×10^{-12} . ISTA is slow; FISTA accelerates but oscillates (its gap rose at 89 iterations); FISTA with restart is both fast and stable, its gap never rose, and the restart fired only four times. Restart improves convergence, not the model: all three methods converge to the same minimiser.

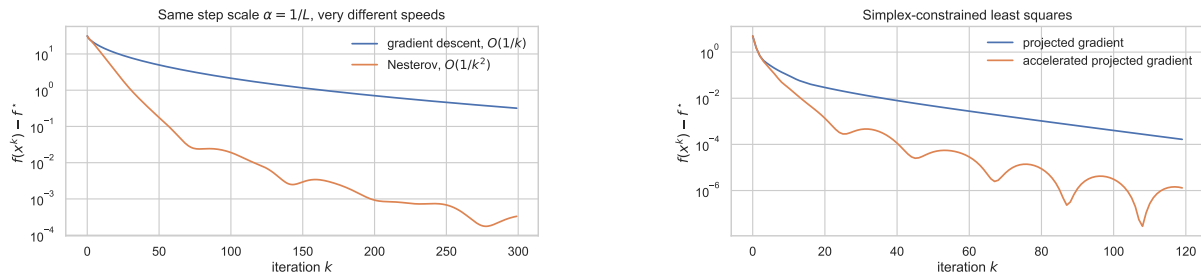


Figure 10.4: Measured optimality gap against iteration, log scale, same step $1/L$. Left: gradient descent and accelerated gradient on the least-squares instance of [Example 10.15](#); the orders $O(1/k)$ and $O(1/k^2)$ in the legend are the guaranteed worst-case rates, not slopes fitted to the curves, which decay faster. Right: projected gradient and its accelerated version on the simplex-constrained instance of [Example 10.16](#). The ripples are momentum overshooting and correcting.

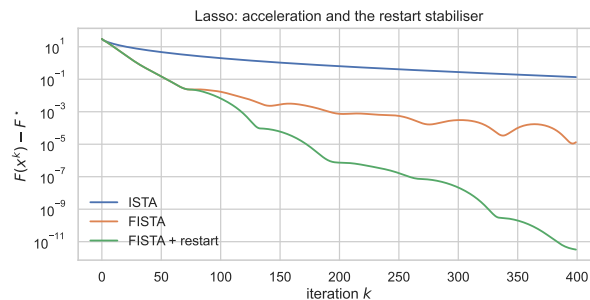


Figure 10.5: Benchmark 3 ([Example 10.17](#)): optimality gap $F(x^k) - F^*$ of ISTA, FISTA and FISTA with objective restart on the lasso, same prox, same step $1/L$ (log scale).

Note on the lecture slides

Frame 10.37 says that “FISTA keeps the same sparse structure as ISTA”. About the model this is right: all three methods converge to the same minimiser, whose sparsity is fixed by λ . On the plotted instance, however, nothing sparse is on display: with $\lambda = 0.02$ (against $\|A^\top b\|_\infty = 16.55$) all 60 coefficients are nonzero; $\lambda = 5$ would leave five. The run illustrates speed, not sparsity.

[Listing 10.1](#) implements [Algorithm 10.3](#) in NumPy for the lasso. Called on the benchmark data with $K = 400$, it returns points with $F = 30.62716$ without restart and $F = 30.62715$ with restart, against $F = 30.76206$ for 400 ISTA steps.

Listing 10.1: FISTA for the lasso with optional objective restart ([Algorithm 10.3](#)).

```
import numpy as np
soft = lambda z, tau: np.sign(z)*np.maximum(np.abs(z) - tau, 0)
def fista(A, b, lam, alpha, K, restart=False):
    F = lambda x: 0.5*np.sum((A@x - b)**2) + lam*np.abs(x).sum()
    grad = lambda x: A.T@(A@x - b) # gradient of the smooth part
    x = np.zeros(A.shape[1]); y = x.copy(); theta = 1.0
    for k in range(K):
        xn = soft(y - alpha*grad(y), alpha*lam) # prox-gradient step at y
        if restart and F(xn) > F(x): # objective rose: drop momentum,
            y, theta = x.copy(), 1.0 # redo the step from x itself
        xn = soft(y - alpha*grad(y), alpha*lam)
        theta_new = 0.5*(1 + np.sqrt(1 + 4*theta**2))
        y = xn + ((theta - 1)/theta_new)*(xn - x) # extrapolate with beta_k
        x, theta = xn, theta_new
```

```
return x
```

The function keeps three vectors (x , y and the trial point) and one scalar; the restart test costs up to two objective evaluations per iteration. The coding lanes are unchanged: NumPy is where algorithms are developed and their curves compared, RSOME with a conic solver (Week 4) is the small-instance benchmark that certifies the optimum; compare objective *and* residual curves against the iteration count, not merely final values.

10.6 Summary and looking ahead

What changed from Weeks 6 to 9? From Week 6 we kept the smooth gradient step and the step $\alpha = 1/L$, and added an extrapolated point y^k and a faster rate: $O(1/k^2)$ in place of $O(1/k)$, and $(1 - 1/\sqrt{\kappa})^k$ in place of $(1 - 1/\kappa)^k$. From Week 8 we kept the projection, from Week 9 the prox and the composite optimality condition, and now apply them after an extrapolated gradient step; the stopping residuals are unchanged. From Week 7 nothing changed: $O(1/\sqrt{k})$ is already optimal for general nonsmooth problems and momentum cannot improve that order. The answer to the opening question is a qualified yes. The same gradients, projections and proxes, applied at a look-ahead point with a designed momentum coefficient, converge at the best order any gradient-based method can achieve, and the improvement is largest exactly where gradient descent is worst, on ill-conditioned problems. Acceleration is what smooth structure buys, not a universal upgrade, and its limitations are as clear as its gains: it needs a smooth part and its L (or backtracking); it is not monotone, so restart is part of the practical method; the constant coefficient needs μ ; and the extrapolated point must stay where the smooth part is defined.

Acceleration is also not the only way to go faster at scale. Week 11 asks what to do when a single gradient is itself the bottleneck because N is huge: it samples the gradient, the clean $O(1/k^2)$ story does not carry over to noisy gradients, and momentum returns in the heavy-ball form, without a guarantee but with a job. Weeks 12 and 13 ask what to do when the structure is richer, with many coupled variables or constraints that have no cheap projection. The origins: Polyak (1964) for the heavy ball, Nesterov (1983, 2004) for the accelerated method and its optimality, Beck and Teboulle (2009) for FISTA, O’Donoghue and Candès (2015) for adaptive restart.

10.7 Exercises

Exercise 10.1 (Match the model to its method). For each model name the baseline first-order method of Weeks 6 to 9 and, where one exists, its accelerated analogue from Table 10.2, with both headline rates: (a) least squares; (b) least squares over the simplex (Model 10.12); (c) the lasso (10.15); (d) least absolute deviations $\min \|Ax - b\|_1$. Explain in one sentence why (d) has no accelerated version.

Exercise 10.2 (The eigen-mode picture and what extrapolation captures). For $f(x) = \frac{1}{2}x^\top Qx$ with $Q \succ 0$ symmetric, write the gradient-descent iterate in the eigenvector basis and show that each mode contracts by $1 - \alpha\lambda_i$. Explain why the step is limited by L while the *slowest* mode is governed by μ , and hence why $\kappa = L/\mu$ controls the speed. Repeat the computation for the accelerated update with constant β to obtain (10.6) and verify the double root 0.8 for $\kappa = 25$, $\alpha = 1/L$, $\beta = 2/3$. In plain language, what does $y = x + \beta(x - x^-)$ try to capture, and why does it help most when κ is large?

Exercise 10.3 (Updates by hand). (a) On $f(x) = \frac{1}{2}(x_1^2 + 25x_2^2)$ with $\alpha = 1/25$ and $x^0 = x^{-1} = (1, 1)$, compute x^1 , x^2 and x^3 for gradient descent, for the heavy ball with $\beta = 4/9$ and for Nesterov’s update (10.3) with $\beta = 2/3$, and describe what the extrapolated points y^1 , y^2 do in each coordinate. (b) For the one-dimensional lasso $\min_x \frac{1}{2}(x - 3)^2 + |x|$ ($x^* = 2$, $F^* = 2.5$) with $\alpha = \frac{1}{2}$, show that the proximal-gradient map is $x^+ = \text{soft}_{0.5}(0.5y + 1.5)$, then compute five ISTA

iterates from $x^0 = 0$ and five FISTA iterates ([Algorithm 10.3](#) without restart, tracking θ_k , β_k and y^k). Which method overshoots x^* , when does its objective rise, and what would the restart test have done there?

Exercise 10.4 (One family). Starting from $x^{k+1} = x^k - \alpha \nabla f(x^k)$, from $x^{k+1} = \Pi_C(x^k - \alpha \nabla f(x^k))$ and from $x^{k+1} = \text{prox}_{\alpha h}(x^k - \alpha \nabla g(x^k))$, write the accelerated smooth, accelerated projected-gradient and FISTA updates. Explain, using [Proposition 9.8](#), why $h = I_C$ recovers accelerated projected gradient and $h = 0$ the smooth update, and give an example in which $y^k \notin C$ although $x^{k-1}, x^k \in C$.

Exercise 10.5 (Coding lab: least squares). Implement gradient descent, Barzilai–Borwein gradient descent (the Barzilai–Borwein exercise of [Week 6](#)) and [Algorithm 10.2](#) for an ill-conditioned least-squares problem such as the data of [Example 10.15](#), and plot the objective value and the gradient norm against the iteration count (or the gap, using a reference solution from the normal equations). Which method reaches a useful error level first, and how does the conditioning of A change what you observe? As an extension, compare the fixed step with the backtracking version of [Section 10.3.3](#).

Exercise 10.6 (Coding lab: lasso, restart and solver comparison). Implement ISTA and FISTA for the lasso ([10.15](#)), add the objective restart rule of [Section 10.5.1](#), and compare the three gap curves and the three residual curves $\|\tilde{G}_\alpha(x^k)\|_2$; explain why the FISTA curve can rise. Use RSOME with a conic solver on a small instance as a benchmark for F^* , and comment on when custom first-order code becomes the better option at large scale.

Week 11 Optimisation for Large Datasets: Stochastic Gradient Methods

What is worth computing when one gradient costs a full pass over the data? Every algorithm week so far has taken one thing for granted: [Week 6](#) descended along the exact gradient, [Week 7](#) replaced it by a subgradient at a kink, [Weeks 8 and 9](#) followed the gradient step with a projection or a proximal operator, and [Week 10](#) reused the previous move to cross an ill-conditioned valley faster, but in each case the difficulty lay in the *structure* of the problem and the gradient of the smooth part was assumed affordable. This week the model is the same convex data-fitting problem, an average of one loss per record, but there are so many records that a single gradient, which touches every one of them, is itself the bottleneck. The answer is to stop insisting on the exact gradient: one record drawn at random gives a direction that is wrong almost every time but right *on average*, at $1/N$ of the cost, and plugging it into the update of [Algorithm 6.1](#) gives the *stochastic gradient method*. Everything else in the week, step-size schedules, mini-batching, iterate averaging, momentum, adaptive steps and variance reduction, is a response to the noise that this one choice introduces.

[Section 11.1](#) states the finite-sum model and what one gradient costs. [Section 11.2](#) reads the empirical average as an expectation, defines unbiased sampled gradients, states the method and measures what happens: sampling races ahead of full gradient descent per pass over the data, then stalls on a floor. [Section 11.3](#) explains both facts with the one-step inequality of [Week 7](#), one term changed. [Section 11.4](#) turns the analysis into practice, from step-size schedules to Adam, and [Section 11.5](#) composes sampling with projection, prox and subgradients, presents variance reduction and says when not to sample at all. Throughout, the step size is α_k as in [Weeks 7 to 10](#) ([Week 6](#) wrote t_k), N is the number of records, $x \in \mathbb{R}^d$ the decision variable and s the mini-batch size.

11.1 The finite-sum model and the cost of one gradient

11.1.1 One model behind many applications

Churn prediction from weekly customer records, click-through prediction from streaming ad impressions, pricing-response models refitted over millions of transactions and large forecasting regressions are, mathematically, one object.

Model 11.1 (Finite-sum data-fitting model). Given N records, each a feature vector $a_i \in \mathbb{R}^d$ and a response b_i ,

$$\min_{x \in \mathbb{R}^d} F(x) = \frac{1}{N} \sum_{i=1}^N f_i(x), \quad f_i(x) = \ell(a_i^\top x, b_i), \quad (11.1)$$

where ℓ is a convex loss comparing the prediction $a_i^\top x$ with the response. The two running instances are the **squared loss**, $f_i(x) = \frac{1}{2}(a_i^\top x - b_i)^2$ (least squares, $b_i \in \mathbb{R}$), and the **logistic loss**, $f_i(x) = \log(1 + \exp(-b_i a_i^\top x))$ (logistic regression, $b_i \in \{-1, 1\}$).

The decision variable x holds the model coefficients, the objective is the average loss over the records, and there are no constraints for now ([Section 11.5](#) adds them). Each f_i is a convex function of the affine expression $a_i^\top x$ ([Proposition 3.19](#); for the logistic loss [Example 3.22](#)), so F is convex, and both running losses are differentiable, so F is smooth: these are the model classes of [Section 4.2](#), now at scale. Here x^* holds the fitted coefficients of [Model 1.3](#) and of [Model 4.18](#) with $\lambda = 0$, written there as β and θ (and as w in [Example 6.17](#)); the factor $1/N$, absent from the logistic sums of [Weeks 4, 6 and 9](#), changes the optimal value but not the minimiser. With A the $N \times d$ matrix whose rows are $a_i^\top$, the least-squares objective is $F(x) = \frac{1}{N} \cdot \frac{1}{2} \|Ax - b\|_2^2$, so

its constants L and μ (Definitions 6.8 and 6.12) are the extreme eigenvalues of $A^\top A/N$ and F is strongly convex exactly when A has independent columns (Proposition 1.13); the logistic loss is convex and smooth but not strongly convex without a regulariser (Remark 11.12).

11.1.2 What one gradient costs

By linearity of differentiation applied to (11.1),

$$\nabla F(x) = \frac{1}{N} \sum_{i=1}^N \nabla f_i(x), \quad (11.2)$$

so one gradient touches every record: for the two running losses one inner product and one scaled vector per record, $O(Nd)$ arithmetic in all. Measured with $d = 20$ features on one machine, a full least-squares gradient takes about 0.15 ms at $N = 10^4$, 8 ms at $N = 10^6$ and 24 ms at $N = 3 \cdot 10^6$: ten times the data, ten times the wait. A thousand gradient steps at $N = 3 \cdot 10^6$ therefore run for about 24 seconds, for *one* fit, before tuning or cross-validation multiplies the count, and every method of Weeks 6 and 10 pays this price at every iteration: acceleration reduces the number of iterations, not the cost of each. So the question is not how to take better steps but whether we must look at every record to take a step at all.

11.2 Sampling the gradient

11.2.1 The empirical average is an expectation

The objective is an average over records; giving that average its standard name lets one randomly chosen record stand in for the whole.

Definition 11.2 (Random draw and expectation). A **random draw** ξ from $\{1, \dots, N\}$ picks each index i with probability $1/N$. For any quantity $h(\xi)$ depending on the draw, a number or a vector, its **expectation** is the probability-weighted average

$$\mathbb{E}_\xi[h(\xi)] := \sum_{i=1}^N \frac{1}{N} h(i). \quad (11.3)$$

Successive draws $\xi_0, \xi_1, \dots$ are **independent**: each is a fresh draw unaffected by the earlier ones, so a pair (ξ_k, ξ_l) with $k \neq l$ takes each of the N^2 values (i, j) with probability $1/N^2$.

Three consequences follow from (11.3): expectation is *linear*, $\mathbb{E}[c_1 h_1 + c_2 h_2] = c_1 \mathbb{E}[h_1] + c_2 \mathbb{E}[h_2]$; it acts *coordinatewise* on vectors; and for independent draws the expectation of a product factorises, $\mathbb{E}[h(\xi_k)^\top h'(\xi_l)] = \sum_{i,j} \frac{1}{N^2} h(i)^\top h'(j) = \mathbb{E}[h(\xi_k)]^\top \mathbb{E}[h'(\xi_l)]$, a fact needed once, in Proposition 11.20. Applied to the finite sum,

$$\mathbb{E}_\xi[f_\xi(x)] = \sum_{i=1}^N \frac{1}{N} f_i(x) = F(x), \quad \mathbb{E}_\xi[\nabla f_\xi(x)] = \sum_{i=1}^N \frac{1}{N} \nabla f_i(x) = \nabla F(x), \quad (11.4)$$

the second by (11.2): the same sums as before, renamed. The payoff is the second identity read from right to left. One sampled gradient $\nabla f_\xi(x)$, costing $O(d)$ instead of $O(Nd)$, is already a correct estimate of the full gradient, in the precise sense that its average over the draw is exactly $\nabla F(x)$.

Algorithm 11.1 Stochastic gradient descent with mini-batches (SGD is the case $s = 1$)

Require: routines returning $\nabla f_i(x)$ for any record i ; starting point x^0 ; batch size $s \geq 1$; step-size rule $k \mapsto \alpha_k$; budget of P passes over the data

Ensure: the final iterate x^K , or the averaged iterate $\bar{x}^T$ of [Theorem 11.18](#) with horizon $T = K$

```

1:  $K \leftarrow \lfloor PN/s \rfloor$   $\triangleright$  steps that fit in the budget, Definition 11.6
2: for  $k = 0, 1, \dots, K - 1$  do
3:   draw  $S_k = \{\xi_{k,1}, \dots, \xi_{k,s}\}$ ,  $s$  independent uniform indices  $\triangleright$  fresh draw each step
4:    $g^k \leftarrow \frac{1}{s} \sum_{i \in S_k} \nabla f_i(x^k)$   $\triangleright \mathbb{E}[g^k | x^k] = \nabla F(x^k)$ ; costs  $s$  gradients
5:    $x^{k+1} \leftarrow x^k - \alpha_k g^k$ 
6: end for
```

11.2.2 Two unbiased estimates of the gradient

Definition 11.3 (Unbiased gradient estimate). A random vector g is an **unbiased estimate** of $\nabla F(x)$ if $\mathbb{E}[g] = \nabla F(x)$. Two such estimates are used throughout: the **sampled gradient** $g = \nabla f_\xi(x)$ for one draw ξ , costing one gradient evaluation, and the **mini-batch gradient** $g = \frac{1}{s} \sum_{j=1}^s \nabla f_{\xi_j}(x)$ for s independent draws $\xi_1, \dots, \xi_s$, costing s . Both are unbiased for every x , using only differentiability of the f_i : the sampled gradient by (11.4), and the mini-batch because each of its s terms has expectation $\nabla F(x)$ by the same identity, so by linearity $\mathbb{E}[g] = \frac{1}{s} \cdot s \nabla F(x)$.

The two estimates differ only in cost and in spread, and the full gradient is their noiseless limit, the spread vanishing as $s \rightarrow \infty$ ([Proposition 11.20](#)). Unbiasedness is a statement about the *average* of the draws, not about any one of them. Picture six possible values of $-g$ at one point, fanned out at $\pm 16^\circ$, $\pm 30^\circ$ and $\pm 44^\circ$ from $-\nabla F(x)$ with lengths 2.00, 1.85 and 1.70: their sideways components cancel in pairs and their mean component along $-\nabla F(x)$ is $(2 \cdot 2.00 \cos 16^\circ + 2 \cdot 1.85 \cos 30^\circ + 2 \cdot 1.70 \cos 44^\circ)/6 = 1.58$, the length of $-\nabla F(x)$ itself, so the draws average *exactly* to the full gradient while each one points well off it, and some sampled gradients point uphill. Over many steps the wrong moves cancel in expectation and the right ones accumulate, which is what the method below relies on.

Note on the lecture slides

Frame 11.6 says that “the full gradient is the case $s = N$ ”. For s *independent* draws, which is what the frame, the $1/s$ law of [Proposition 11.20](#) and the notebook’s code all use, this is false: with $s = N$ only about 63% of the records appear (others are drawn twice) and the estimate still has variance $\text{Var}(g)/N$. The full gradient is the *noiseless limit* $s \rightarrow \infty$, and the case $s = N$ only for sampling *without* replacement.

11.2.3 The stochastic gradient method

The update of [Algorithm 6.1](#) is $x^{k+1} = x^k - \alpha_k \nabla F(x^k)$, an exact direction that costs N gradients. Replace the exact direction by an unbiased one.

[Algorithm 11.1](#) is the **stochastic gradient method**; with $s = 1$ it is **stochastic gradient descent (SGD)**, $g^k = \nabla f_{\xi_k}(x^k)$ for a single draw ξ_k , and the name SGD is used loosely for every batch size. The comment “ $\mathbb{E}[g^k | x^k] = \nabla F(x^k)$ ” reads: freeze the current iterate and average over the fresh draw ([Definition 11.8](#)). There is no stopping test on $\|\nabla F(x^k)\|_2$, because evaluating ∇F would cost the very pass we are avoiding ([Section 11.4.3](#) gives substitutes), so the loop runs on a budget of passes; and the step-size rule is left open because, as [Section 11.3](#) shows, it decides whether the method converges at all.

Key idea

The optimisation template does not change; only the information at each step does. When a full gradient is too expensive we buy cheap directions that are right on average, and then manage the noise with step sizes, batching and averaging.

11.2.4 Sampled gradients for least squares and logistic regression

Example 11.4 (Least-squares sampled gradient). With $f_i(x) = \frac{1}{2}(a_i^\top x - b_i)^2$ the chain rule (Proposition 1.17) gives

$$\nabla f_i(x) = (a_i^\top x - b_i) a_i, \quad \nabla F(x) = \frac{1}{N} \sum_{i=1}^N (a_i^\top x - b_i) a_i = \frac{1}{N} A^\top (Ax - b),$$

the second being the gradient of Model 6.16 divided by N . One SGD step picks a single row (a_i, b_i) and computes one inner product, one residual and one scaled vector: $O(d)$ work, with no sum over N anywhere in the step.

Example 11.5 (Logistic sampled gradient). With $f_i(x) = \log(1 + \exp(-b_i a_i^\top x))$ and $b_i \in \{-1, 1\}$, the chain rule gives

$$\nabla f_i(x) = \frac{-b_i a_i \exp(-b_i a_i^\top x)}{1 + \exp(-b_i a_i^\top x)} = -\frac{b_i a_i}{1 + \exp(b_i a_i^\top x)}, \quad (11.5)$$

the per-record term of Example 6.17. The weight $1/(1 + \exp(b_i a_i^\top x))$ lies in $(0, 1)$: a record with a large positive margin $b_i a_i^\top x$ is confidently and correctly classified and contributes almost nothing, while a misclassified or uncertain record contributes nearly $-b_i a_i$. Again one record gives one $O(d)$ direction.

11.2.5 What to count: iterations versus passes

Iterations are no longer comparable across methods, because an iteration of full gradient descent, of mini-batch SGD and of SGD cost wildly different amounts; what is comparable is the amount of data read.

Definition 11.6 (Pass over the data). A **pass over the data** is N gradient evaluations $\nabla f_i(\cdot)$, however they are grouped into steps; machine-learning software calls one pass an **epoch**. Full gradient descent spends N gradients per step and takes 1 step per pass; mini-batch SGD spends s per step and takes N/s steps per pass; SGD spends 1 per step and takes N steps per pass.

Throughout the week progress is plotted against passes, never against steps: methods are compared at equal data read, which is, up to the vectorisation effects of Section 11.4.2, equal wall-clock time.

11.2.6 The phenomenon, measured

Example 11.7 (The running least-squares instance). All measurements in Sections 11.2 to 11.4 use one least-squares instance with $N = 4000$ records and $d = 20$ features: $A = Z \text{diag}(1, 10^{-1/19}, \dots, 10^{-1})$ with Z standard normal, so the feature scales decay geometrically from 1 to 0.1, and $b = Ax_{\text{true}} + 0.5e$ with e standard normal and x_{true} a fixed coefficient vector; the start is $x^0 = 0$. Here $L = 1.036$, $\mu = 0.00999$, $\kappa = L/\mu \approx 104$ (Definition 10.3), $F(x^0) - F^* = 1.258$, and the constants of the noise model of Definition 11.10 are $a_{\text{max}} = 4.42$, $L_g = 27.7$ and $B = 1.51$.

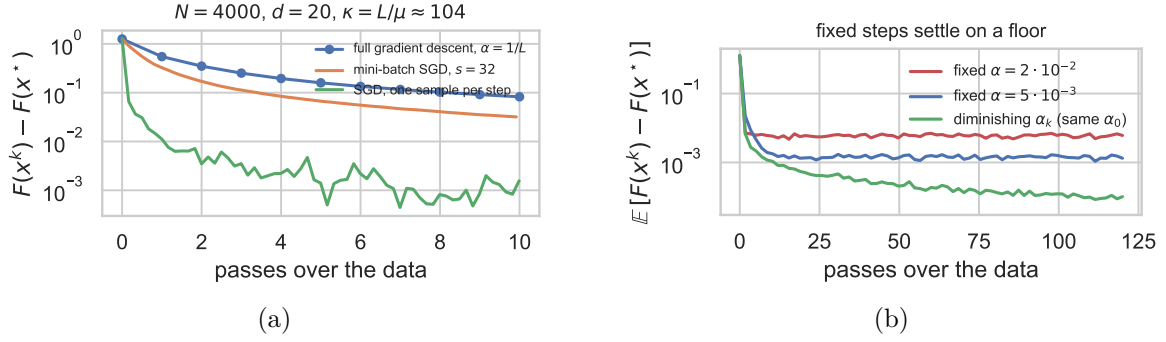


Figure 11.1: The phenomenon to be explained, on [Example 11.7](#). (a) Optimality gap $F(x^k) - F^*$ against passes over the data for full gradient descent ($\alpha = 1/L$), mini-batch SGD ($s = 32$) and SGD ($s = 1$), the sampled runs with $\alpha_k = 0.02/(1 + k/2N)$: at equal data read, sampling is far ahead. (b) SGD for 120 passes, $\mathbb{E}[F(x^k) - F^*]$ estimated from 24 independent runs: the two fixed steps descend fast and then stop, the larger step on a higher floor; only the decaying step is still falling.

[Figure 11.1\(a\)](#) runs [Algorithm 11.1](#) on this instance for ten passes, with $\alpha_k = 0.02/(1 + k/2N)$ for the two sampled methods. The optimality gaps $F(x) - F^*$ after ten passes are $8.23 \cdot 10^{-2}$ for full gradient descent at $\alpha = 1/L$ (the gap reduced 15 $\times$), $3.16 \cdot 10^{-2}$ for mini-batches of $s = 32$ (40 $\times$) and $1.25 \cdot 10^{-3}$ for SGD with $s = 1$ (1010 $\times$): full gradient descent has cut the gap by one order of magnitude and sampling one row at a time by three. Nothing about the model changed; the sampled method took 40 000 cheap steps where the full-gradient method took ten expensive ones, and each cheap step was right on average.

[Figure 11.1\(b\)](#) looks longer and more carefully. With a fixed step the cheap method stops improving: from pass 30 to pass 120 the run with $\alpha = 2 \cdot 10^{-2}$ moves from $6.20 \cdot 10^{-3}$ to $6.08 \cdot 10^{-3}$ and the run with $\alpha = 5 \cdot 10^{-3}$ from $1.41 \cdot 10^{-3}$ to $1.33 \cdot 10^{-3}$: the larger step is stuck about 4.6 $\times$ higher with a step 4 $\times$ larger, a proportionality that [Theorem 11.13](#) will pin down, while the decaying step $\alpha_k = 0.02/(1 + k/2N)$ is still descending, from $4.25 \cdot 10^{-4}$ to $1.02 \cdot 10^{-4}$. Two questions therefore organise the analysis: *why does sampling work at all*, and *what sets that floor*?

11.3 What the analysis says

11.3.1 Two more probabilistic tools

Definition 11.8 (Variance, second moment, conditional expectation). For a random vector g with expectation $\mathbb{E}[g]$, the **variance** is the scalar

$$\text{Var}(g) := \mathbb{E} \|g - \mathbb{E}[g]\|_2^2 = \mathbb{E} \|g\|_2^2 - \|\mathbb{E}[g]\|_2^2, \quad (11.6)$$

and $\mathbb{E} \|g\|_2^2$ is the **second moment**. In [Algorithm 11.1](#) the iterate x^k is a function of the earlier draws $\xi_0, \dots, \xi_{k-1}$, and ξ_k is independent of them. The **conditional expectation given x^k** , written $\mathbb{E}[\cdot | x^k]$, freezes x^k and averages over ξ_k alone; averaging over ξ_k and then over the earlier draws is averaging over all of them, $\mathbb{E}[\mathbb{E}[\cdot | x^k]] = \mathbb{E}[\cdot]$ (the **tower rule**).

The second form of (11.6) follows by expanding the square with $m = \mathbb{E}[g]$: $\mathbb{E} \|g - m\|_2^2 = \mathbb{E} \|g\|_2^2 - 2m^\top \mathbb{E}[g] + \|m\|_2^2$. For a sampled gradient it reads $\mathbb{E} \|g\|_2^2 = \|\nabla F(x)\|_2^2 + \text{Var}(g)$: the second moment, which the bounds below use, is the squared length of the true gradient plus the spread around it. Apart from the product rule for independent draws stated after [Definition 11.2](#), the probabilistic toolkit of the week is: freeze the current point, average over one fresh draw, then average again.

11.3.2 One step, in expectation

Lemma 7.13 expanded one step $x^{k+1} = x^k - \alpha_k g^k$ of the subgradient method and tracked the squared distance $\Delta_k := \|x^k - x^*\|_2^2$ that **Theorem 7.18** then analysed. We repeat the same line, with the same symbol g^k , except that g^k is now drawn at random.

Lemma 11.9 (One step in expectation). *Let x^* be a minimiser of F and let $x^{k+1} = x^k - \alpha_k g^k$ with $\mathbb{E}[g^k | x^k] = \nabla F(x^k)$. Then*

$$\mathbb{E}[\|x^{k+1} - x^*\|_2^2 | x^k] = \Delta_k - \underbrace{2\alpha_k \nabla F(x^k)^\top (x^k - x^*)}_{\text{signal}} + \underbrace{\alpha_k^2 \mathbb{E}[\|g^k\|_2^2 | x^k]}_{\text{noise}}. \quad (11.7)$$

If F is convex then $\nabla F(x^k)^\top (x^k - x^*) \geq F(x^k) - F^* \geq 0$, so the signal term is ≤ 0 .

Proof. Expanding the square as in **Lemma 7.13**, $\|x^k - \alpha_k g^k - x^*\|_2^2 = \Delta_k - 2\alpha_k (g^k)^\top (x^k - x^*) + \alpha_k^2 \|g^k\|_2^2$. Under $\mathbb{E}[\cdot | x^k]$ the quantities Δ_k , α_k and $x^k - x^*$ are frozen, so by linearity only g^k is averaged, and unbiasedness turns $(g^k)^\top (x^k - x^*)$ into $\nabla F(x^k)^\top (x^k - x^*)$. The last claim is the first-order inequality of **Theorem 3.8**, $F^* \geq F(x^k) + \nabla F(x^k)^\top (x^* - x^k)$. $\square$

Write $A_k := \mathbb{E} \Delta_k$ for the **expected squared distance** to the solution; averaging (11.7) over the earlier draws by the tower rule turns it into a statement about A_{k+1} and A_k , and every rate of this week comes from this line. The **signal** term is the same as in **Week 7**: negative for convex F , linear in α_k , pulling the iterate towards x^* . The **noise** term is nonnegative, quadratic in α_k , and the one place where randomness enters: **Week 7** had $\alpha_k^2 \|g^k\|_2^2$ with $\|g^k\|_2 \leq G$, here it is the second moment of the sampled direction. Their ratio runs the show (for a small step the signal wins, for a large step the noise wins; **Section 11.4.1** names the two regimes), and two things are new. Nothing holds step by step, only on average, so a single run can and does move away from x^* . And the noise term may shrink as $x^k \rightarrow x^*$, since for least squares the sampled gradients $(a_i^\top x - b_i)a_i$ are small where the residuals are small; it therefore needs a model.

11.3.3 A model for the noise

Definition 11.10 (Second-moment model of the sampled gradient). Write $g(x; \xi) := \nabla f_\xi(x)$ for the sampled gradient as a function of the point and the draw, $a_{\max} := \max_i \|a_i\|_2$ for the largest feature norm and, for least squares, $r_i^* := a_i^\top x^* - b_i$ for the residual of record i at the solution. Constants $L_g \geq 0$ and $B \geq 0$ form a **noise model** for the problem if for every x

$$\mathbb{E}_\xi \|g(x; \xi)\|_2^2 \leq L_g^2 \|x - x^*\|_2^2 + B^2. \quad (11.8)$$

The two constants have distinct roles: L_g bounds how fast the spread grows away from x^* , and B is what survives at x^* ; it is B that will set the floor.

Proposition 11.11 (Noise-model constants for the running losses). (a) For least squares, (11.8) holds with

$$L_g^2 = 2a_{\max}^4, \quad B^2 = \frac{2}{N} \sum_{i=1}^N \|a_i\|_2^2 r_i^{*2},$$

and $B = 0$ if and only if every residual at the solution vanishes (for records with $a_i \neq 0$). (b) For logistic regression $\|g(x; i)\|_2 \leq a_{\max}$ for every x and i , so (11.8) holds with $L_g = 0$ and $B = a_{\max}$.

Proof. (a) Since $b_i = a_i^\top x^* - r_i^*$, $g(x; i) = (a_i^\top (x - x^*) + r_i^*)a_i$, so $\|g(x; i)\|_2^2 = \|a_i\|_2^2 (a_i^\top (x - x^*) + r_i^*)^2$. The inequality $(u + v)^2 \leq 2u^2 + 2v^2$ and Cauchy–Schwarz (**Proposition 1.11**), $(a_i^\top (x - x^*))^2 \leq \|a_i\|_2^2 \|x - x^*\|_2^2$, give $\|g(x; i)\|_2^2 \leq 2\|a_i\|_2^4 \|x - x^*\|_2^2 + 2\|a_i\|_2^2 r_i^{*2}$; average over i

and bound $\|a_i\|_2^4 \leq a_{\max}^4$. Each term of B^2 is nonnegative, so $B = 0$ forces every $r_i^* = 0$. (b) From (11.5), $\|g(x; i)\|_2 = \|a_i\|_2 / (1 + \exp(b_i a_i^\top x)) \leq a_{\max}$, because $|b_i| = 1$ and the weight lies in $(0, 1)$. $\square$

The logistic case is exactly [Week 7](#)'s bounded-subgradient assumption $\|g^k\|_2 \leq G$ with an expectation in front, the case [Theorem 11.18](#) runs on. The least-squares case is different in kind: the bound shrinks as the iterate approaches x^* and disappears when the data are fitted exactly. On [Example 11.7](#), $a_{\max}^2 = 19.56$ gives $L_g^2 = 765$ and $B^2 = 2.29$, twice the true second moment at x^* , $\frac{1}{N} \sum_i \|a_i\|_2^2 r_i^{*2} = 1.15$, the factor 2 being the price of $(u + v)^2 \leq 2u^2 + 2v^2$.

Remark 11.12 (Which theorem covers which example). [Theorems 11.13](#) and [11.15](#) and [Corollary 11.14](#) assume μ -strong convexity and cover least squares with independent columns (and any loss with a ridge term $\frac{\lambda}{2} \|x\|_2^2$ added); [Theorem 11.18](#) assumes only convexity and a bounded second moment and covers logistic regression. The hypotheses also constrain each other: if F is μ -strongly convex on all of $\mathbb{R}^d$ then $\mu \|x - x^*\|_2 \leq \|\nabla F(x)\|_2 \leq (\mathbb{E} \|g(x; \xi)\|_2^2)^{1/2}$ for every x ((11.9) below with Cauchy–Schwarz, then (11.6)), so a global noise model must have $L_g \geq \mu$. The bounded case $L_g = 0$ is meaningful on a bounded region, as in [Week 7](#); the strongly convex results below keep a general L_g .

11.3.4 Strongly convex, fixed step: contraction, then a floor

Recall that F is μ -strongly convex ([Definition 6.12](#)) when $F(y) \geq F(x) + \nabla F(x)^\top (y - x) + \frac{\mu}{2} \|y - x\|_2^2$ for all x, y . Applying this at (x, x^*) and at (x^*, x) with $\nabla F(x^*) = 0$ and adding gives the form [Theorem 7.18](#) used,

$$\nabla F(x)^\top (x - x^*) \geq \mu \|x - x^*\|_2^2 \quad \text{for all } x, \quad (11.9)$$

so the signal term of (11.7) is at most $-2\alpha_k \mu \Delta_k$: strong convexity turns the signal into a contraction.

Theorem 11.13 (SGD with a fixed step under strong convexity). *Let F be differentiable and μ -strongly convex with minimiser x^* , let the directions g^k of [Algorithm 11.1](#) be unbiased with independent draws, and let (L_g, B) be a noise model (11.8) for the single-draw gradient (a batch of $s > 1$ only lowers the second moment, [Proposition 11.20](#), so the same constants serve). For a constant step $0 < \alpha < 2\mu/L_g^2$ set $\rho := 1 - 2\alpha\mu + \alpha^2 L_g^2 \in [0, 1)$. Then*

$$A_{k+1} \leq \rho A_k + \alpha^2 B^2 \quad \text{and hence} \quad A_k \leq \rho^k A_0 + \frac{\alpha B^2}{2\mu - \alpha L_g^2} \quad \text{for all } k. \quad (11.10)$$

Proof. Insert (11.9) and (11.8) into (11.7): $\mathbb{E}[\Delta_{k+1} \mid x^k] \leq \Delta_k - 2\alpha\mu\Delta_k + \alpha^2(L_g^2\Delta_k + B^2) = \rho\Delta_k + \alpha^2 B^2$; the tower rule gives the recursion in A_k . Unrolling, $A_k \leq \rho^k A_0 + \alpha^2 B^2(1 + \rho + \dots + \rho^{k-1}) \leq \rho^k A_0 + \alpha^2 B^2/(1 - \rho)$, and $1 - \rho = \alpha(2\mu - \alpha L_g^2)$, positive exactly when $\alpha < 2\mu/L_g^2$. Finally $\rho \geq 0$, because $L_g \geq \mu$ by [Remark 11.12](#) gives $\rho \geq (1 - \alpha\mu)^2$. $\square$

The bound has two terms of opposite character. The first, $\rho^k A_0$, dies geometrically: this is the sampled counterpart of the linear rate of [Theorem 6.13](#) (which bounds the objective gap of an L -smooth f at the step $1/L$), and it is why sampling works at all. The second ignores k . It is the **floor**, and since $\alpha B^2/(2\mu - \alpha L_g^2) \approx \alpha B^2/(2\mu)$ for small α , the floor is proportional to the step: halving the step halves the floor, which [Figure 11.2\(a\)](#) measures (floors 0.0529, 0.0259, 0.0129 for $\alpha = 0.02, 0.01, 0.005$, so floor/ $\alpha = 2.65, 2.59, 2.59$). Formally setting $L_g = 0$ (the bounded case, meaningful on a bounded region, [Remark 11.12](#)) gives part (a) of [Theorem 7.18](#) with B in place of G , provided also $\alpha < 1/(2\mu)$, the condition there (at $L_g = 0$ the hypothesis $\alpha < 2\mu/L_g^2$ is vacuous and $\rho \geq 0$ must come from the step): sampling noise and a kink put the same term in the same place. The practical reading is sobering: $A_k \leq \varepsilon$ needs a floor below $\varepsilon/2$,

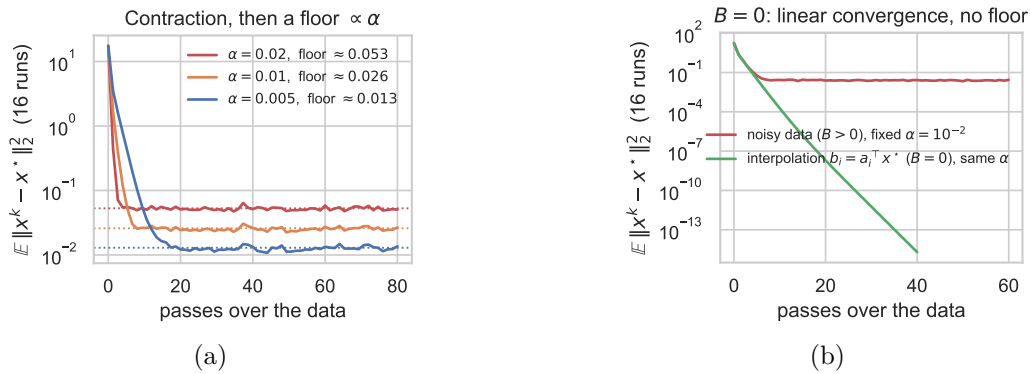


Figure 11.2: Theorem 11.13 and Corollary 11.14 on Example 11.7, $\mathbb{E} \|x^k - x^*\|_2^2$ from 16 runs. (a) Three fixed steps: each run contracts geometrically and then sits on a floor, and halving the step halves the floor. (b) The same matrix, step $\alpha = 10^{-2}$ and seeds, with the responses replaced by $b_i = a_i^\top x_{\text{true}}$ so that every residual at the solution vanishes ($B = 0$): the run converges linearly to machine precision, while the noisy-response run stalls near 0.026.

so $\alpha \approx \mu\epsilon/B^2$, and then about $\frac{1}{2\alpha\mu} \ln \frac{2A_0}{\epsilon} = \frac{B^2}{2\mu^2\epsilon} \ln \frac{2A_0}{\epsilon}$ steps: high accuracy means tiny steps and a count growing like $1/\epsilon$.

Note on the lecture slides

Frames 11.16 and 11.26 read the measured floors, and frame 11.17 the measured linear convergence, as confirming (11.10), but on Example 11.7 the hypothesis is $\alpha < 2\mu/L_g^2 = 2.6 \cdot 10^{-5}$, the runs use $\alpha = 0.02, 0.01, 0.005$, for which $\rho = 1.31, 1.08, 1.02 > 1$, and the constant $B^2/(2\mu) = 115$ is 44 times the measured floor/ α , because $L_g^2 = 2a_{\max}^4$ is very crude. The experiments confirm the *shape* the theorem predicts, a geometric phase then a floor proportional to α (and to $1/s$, Section 11.4.2), not its constants.

Corollary 11.14 (Interpolation: when a fixed step really works). *Under the hypotheses of Theorem 11.13 with $B = 0$, i.e. when every sampled gradient vanishes at the solution, $A_k \leq \rho^k A_0$: linear convergence all the way to x^* at a fixed step.*

There is nothing to prove: the additive term of (11.10) is zero. For least squares $B = 0$ means $b_i = a_i^\top x^*$ for every record, a consistent linear system; in general it means that the model *interpolates* the data. This is the one thing randomness can do that a kink cannot: in Theorem 7.18 the constant G^2 never vanishes, so a fixed step always leaves a floor, whereas here the noise itself dies at the solution (Figure 11.2(b)). Interpolating models are the boundary case, not the default; Section 11.5.3 shows how variance reduction manufactures $B = 0$ on any finite dataset.

Key idea

A fixed step buys linear convergence to a floor proportional to the step, because the noise term of (11.7) never dies; the floor vanishes only when the sampled gradients themselves vanish at the solution. To go below it the step must decay, or the iterates must be averaged.

11.3.5 Decaying steps remove the floor

The floor is proportional to α , so the step must shrink or the noise never dies; but not too fast, or the contraction dies first. The conditions are $\alpha_k \rightarrow 0$ with $\sum_k \alpha_k = \infty$, so that the signal terms keep pulling, and $\sum_k \alpha_k^2 < \infty$, so that the accumulated noise $\sum_k \alpha_k^2 B^2$ stays finite. The family $\alpha_k = \beta/(\gamma + k)$ of Theorem 7.18 satisfies both (harmonic sums diverge, their squares converge); $\beta > 0$ and $\gamma > 0$ are the schedule's constants, the first of three meanings the letter β

carries this week. To run the recursion with a varying step, fold the growth term into the noise constant: if $A_k \leq A_0$ for all k , which holds when $\alpha_0 < 2\mu/L_g^2$ and the run starts at or above the floor of [Theorem 11.13](#) at α_0 (no step can then climb above A_0), then $\alpha_k^2(L_g^2 A_k + B^2) \leq \alpha_k^2 \tilde{B}^2$ with $\tilde{B}^2 := B^2 + L_g^2 A_0$, and (11.7) with (11.9) gives

$$A_{k+1} \leq (1 - 2\alpha_k \mu) A_k + \alpha_k^2 \tilde{B}^2. \quad (11.11)$$

Theorem 11.15 (SGD with a decaying step under strong convexity). *Let F be μ -strongly convex and suppose (11.11) holds. Take $\alpha_k = \beta/(\gamma + k)$ with $2\mu\beta > 1$ and $\gamma \geq 2\mu\beta$ (so that $\alpha_0 \leq 1/(2\mu)$). Then*

$$A_k \leq \frac{\nu}{\gamma + k} = O(1/k), \quad \nu = \max\left\{\frac{\beta^2 \tilde{B}^2}{2\mu\beta - 1}, \gamma A_0\right\}.$$

Proof. By induction. For $k = 0$, $A_0 \leq \nu/\gamma$ because $\nu \geq \gamma A_0$. Suppose $A_k \leq \nu/\hat{k}$ with $\hat{k} := \gamma + k$. Since $\hat{k} \geq \gamma \geq 2\mu\beta$, the factor $1 - 2\alpha_k \mu = 1 - 2\mu\beta/\hat{k}$ is nonnegative, so (11.11) may be applied to the inductive bound:

$$A_{k+1} \leq \frac{\hat{k} - 2\mu\beta}{\hat{k}^2} \nu + \frac{\beta^2 \tilde{B}^2}{\hat{k}^2} = \frac{\hat{k} - 1}{\hat{k}^2} \nu - \frac{(2\mu\beta - 1)\nu - \beta^2 \tilde{B}^2}{\hat{k}^2} \leq \frac{\hat{k} - 1}{\hat{k}^2} \nu \leq \frac{\nu}{\hat{k} + 1},$$

using $\nu \geq \beta^2 \tilde{B}^2/(2\mu\beta - 1)$ and then $(\hat{k} - 1)(\hat{k} + 1) \leq \hat{k}^2$. Since $\hat{k} + 1 = \gamma + (k + 1)$, the induction closes. $\square$

No term independent of k survives: there is no floor left to sit on. This is the diminishing-step part of [Theorem 7.18](#) with $\tilde{B}$ for G , the same family, the same condition $2\mu\beta > 1$ and the same $O(1/k)$; $A_k \leq \varepsilon$ needs $k \geq \nu/\varepsilon$ steps. The condition $2\mu\beta > 1$ is not decorative: if $2\mu\beta < 1$ the steps decay too fast for the contraction to keep up and the rate degrades to $O(k^{-2\mu\beta})$. On [Example 11.7](#), $\alpha_k = 10^{-2}/(1 + k/8N)$ ($2\mu\beta = 6.4$) keeps falling, from $7.5 \cdot 10^{-3}$ to $2.5 \cdot 10^{-3}$ between passes 20 and 80, while $\alpha_k = 10^{-2}/(1 + 2k/N)$ ($2\mu\beta = 0.4$) crawls from 0.91 to 0.44, paying the $O(k^{-0.4})$ price.

Note on the lecture slides

Frame 11.18 states $A_k \leq \nu/(\gamma + k)$ under $2\mu\beta > 1$ alone. The induction needs $1 - 2\mu\alpha_k \geq 0$ at every step, i.e. $\gamma \geq 2\mu\beta$; without it the bound can fail (the mean problem of [Example 11.16](#) with $\beta = 1.9$, $\gamma = 1$, $\sigma^2 = 0.05A_0$ has $\nu = 1.354A_0$ but $A_1 = 0.9905A_0 > \nu/2$), although the $O(1/k)$ order survives by starting the induction at the first k with $\gamma + k \geq 2\mu\beta$. Frame 11.19's miniature ($\beta = \gamma = \mu = 1$) violates $\gamma \geq 2\mu\beta = 2$, so the notes treat it by its exact formula; the measured schedules above satisfy it, as does [Theorem 7.18](#).

The $1/k$ rule is not magic; the simplest strongly convex problem there is shows why it is the natural choice.

Example 11.16 (SGD on a mean is a running average). Let the data be numbers $\omega_1, \dots, \omega_N$ and consider $\min_{x \in \mathbb{R}} \frac{1}{2N} \sum_{i=1}^N (x - \omega_i)^2$, with $\nabla f_i(x) = x - \omega_i$, solution $x^* = \bar{\omega}$ and $\mu = L = 1$. Write $\omega_{(k+1)} := \omega_{\xi_k}$ for the value drawn at step k (the parentheses distinguish drawn values from the data). One SGD step with $\alpha_k = 1/(k + 1)$ is

$$x^{k+1} = x^k - \frac{1}{k+1} (x^k - \omega_{(k+1)}) = \frac{k}{k+1} x^k + \frac{1}{k+1} \omega_{(k+1)}.$$

At $k = 0$ the step is $\alpha_0 = 1$, so $x^1 = \omega_{(1)}$ whatever the start x^0 ; and if $x^k = \frac{1}{k} \sum_{j=1}^k \omega_{(j)}$ then $x^{k+1} = \frac{k}{k+1} \cdot \frac{1}{k} \sum_{j \leq k} \omega_{(j)} + \frac{1}{k+1} \omega_{(k+1)}$, so by induction

$$x^{k+1} = \frac{1}{k+1} \sum_{j=1}^{k+1} \omega_{(j)} \quad \text{for every } k \geq 0,$$

the running average of the values drawn so far: for the draws 4.5, 2.0, 3.8, 1.7, 3.0, 2.7, 2.5, 2.9 the iterates are 4.5, 3.25, 3.43, 3.00, 3.00, 2.95, 2.89, 2.89. The $1/k$ -type rule is the natural way to keep learning from each new draw while gradually averaging the noise out: since x^k is the mean of k independent draws, with $\sigma^2 := \frac{1}{N} \sum_i (\omega_i - \bar{\omega})^2$ one has $\mathbb{E}[x^k] = \bar{\omega}$ and $A_k = \sigma^2/k$ for $k \geq 1$, so the statistics fact “the variance of a sample mean is σ^2/k ” is the $O(1/k)$ rate, with no floor. (The schedule is $\beta/(\gamma+k)$ with $\beta = \gamma = 1$, so $2\mu\beta = 2 > 1$ but $\gamma < 2\mu\beta$; the exact computation replaces [Theorem 11.15](#).) A constant step α instead gives $x^{k+1} = (1 - \alpha)x^k + \alpha\omega_{(k+1)}$, an exponentially weighted average that forgets old draws and keeps fluctuating: a floor, as [Theorem 11.13](#) predicts ([Exercise 11.2](#) asks for the details).

11.3.6 General convex objectives: the averaged iterate and $O(1/\sqrt{T})$

Drop strong convexity: F is convex, there is no contraction to lean on, and this is the setting of logistic regression. Take the bounded case of the noise model, $L_g = 0$ and $B = G$, that is $\mathbb{E}[\|g^k\|_2^2 | x^k] \leq G^2$, as in [Theorem 7.14](#). The guarantee is about a weighted average of the iterates, which needs the many-point form of [Definition 3.2](#).

Lemma 11.17 (Jensen’s inequality for a finite average). *If F is convex, $w_0, \dots, w_{T-1} \geq 0$ and $\sum_k w_k = 1$, then $F(\sum_k w_k x^k) \leq \sum_k w_k F(x^k)$ (by induction on T : write $\sum_{k < T} w_k x^k = (1 - w_{T-1})y + w_{T-1}x^{T-1}$ with y a convex combination of the first $T - 1$ points and apply the two-point definition).*

Theorem 11.18 (SGD for convex objectives, averaged iterate). *Let F be convex and differentiable with a minimiser x^* , let x^0 be given (so $A_0 = \|x^0 - x^*\|_2^2$), and let the directions g^k be unbiased with $\mathbb{E}[\|g^k\|_2^2 | x^k] \leq G^2$. After a fixed number T of steps (the **horizon**, written K in [Corollary 7.15](#)) define the **averaged iterate***

$$\bar{x}^T := \sum_{k=0}^{T-1} w_k x^k, \quad w_k = \frac{\alpha_k}{\sum_{j < T} \alpha_j},$$

the uniform average of $x^0, \dots, x^{T-1}$ when the step is constant. Then

$$\mathbb{E}[F(\bar{x}^T) - F^*] \leq \frac{\|x^0 - x^*\|_2^2 + G^2 \sum_{k < T} \alpha_k^2}{2 \sum_{k < T} \alpha_k}, \quad (11.12)$$

and with the constant step $\alpha = \|x^0 - x^*\|_2 / (G\sqrt{T})$,

$$\mathbb{E}[F(\bar{x}^T) - F^*] \leq \frac{G \|x^0 - x^*\|_2}{\sqrt{T}}. \quad (11.13)$$

Proof. Four steps. (1) *Convexity replaces strong convexity.* By [Lemma 11.9](#) the signal is bounded by the gap itself, so $\mathbb{E}[\Delta_{k+1} | x^k] \leq \Delta_k - 2\alpha_k(F(x^k) - F^*) + \alpha_k^2 G^2$. (2) *Telescope.* Take full expectations and sum over $k = 0, \dots, T - 1$; the A_k cancel in pairs and $A_T \geq 0$ may be dropped: $2 \sum_{k < T} \alpha_k \mathbb{E}[F(x^k) - F^*] \leq A_0 + G^2 \sum_{k < T} \alpha_k^2$. (3) *Average.* Divide by $2 \sum_j \alpha_j$; the left side becomes $\sum_k w_k \mathbb{E}[F(x^k) - F^*] \geq \mathbb{E}[F(\bar{x}^T) - F^*]$ by [Lemma 11.17](#), which is (11.12): the right-hand side of the master bound of [Theorem 7.14](#), now bounding the averaged iterate in expectation (the best iterate of [Week 7](#) is unavailable here, since $F(x^k)$ cannot be evaluated cheaply). (4) *Choose the step.* For a constant α the right side is $A_0/(2\alpha T) + \alpha G^2/2$, minimised where $-A_0/(2\alpha^2 T) + G^2/2 = 0$, i.e. at $\alpha = \sqrt{A_0}/(G\sqrt{T})$, where both halves equal $G\sqrt{A_0}/(2\sqrt{T})$. $\square$

Three comments. The guarantee is about $\bar{x}^T$, not the last iterate, which is why averaging is part of the recipe in [Section 11.4.1](#). Accuracy ε costs $T \geq G^2 \|x^0 - x^*\|_2^2 / \varepsilon^2$ steps, the fixed-horizon tuning of [Corollary 7.15](#) under sampling. And the order cannot be improved: with

Table 11.1: The results of Weeks 6 and 7 (deterministic direction) and their sampled counterparts. The template survives intact; only the one-step noise term changes.

Setting, and what is measured	Weeks 6 and 7, deterministic direction	This week, sampled gradient
One step, L -smooth, $\alpha \leq 1/L$	F falls at every step (Proposition 6.9)	F falls in expectation only up to an α^2 noise term: descent while gradient-dominated (Definition 11.19)
Convex F : rate for $\mathbb{E}[F - F^*]$	$O(1/k)$ for the last iterate, L -smooth, $\alpha \leq 1/L$ (Theorem 6.11); $O(1/\sqrt{k})$ for the best-so-far value under $\ g\ _2 \leq G$ (Corollary 7.15)	$O(1/\sqrt{T})$ for the <i>averaged</i> iterate $\bar{x}^T$, optimal (Theorem 11.18)
μ -strongly convex, fixed α : rate for $A_k = \mathbb{E} \ x^k - x^*\ _2^2$	$F(x^k) - F^* \leq (1 - \mu/L)^k (F(x^0) - F^*)$ for L -smooth F at $\alpha = 1/L$ (Theorem 6.13), hence $\ x^k - x^*\ _2^2 \leq \frac{2}{\mu}$ times this; floor $\alpha G^2/(2\mu)$ under $\ g\ _2 \leq G$, $\alpha < 1/(2\mu)$ (Theorem 7.18)	linear, to a <i>ball</i> : floor $\alpha B^2/(2\mu - \alpha L_g^2)$, the floor of Theorem 7.18 with B for G (Theorem 11.13)
... the same, when the sampled gradients vanish at x^* ($B = 0$)	Week 7: no analogue, G is fixed once and for all; the full gradient of Week 6 is this case ($\nabla F(x^*) = 0$), linear to x^* (row above)	linear, all the way to x^* (Corollary 11.14)
μ -strongly convex, $\alpha_k = \beta/(\gamma + k)$, $2\mu\beta > 1$, $\gamma \geq 2\mu\beta$	$\ x^k - x^*\ _2^2 = O(1/k)$ (Theorem 7.18)	$A_k = O(1/k)$: same family, same condition (Theorem 11.15)
Cost of one step	N gradients	s gradients: the whole point

only noisy gradients of bounded variance, $O(1/\sqrt{T})$ is optimal (stated, not proved), the optimal order of Remark 7.17 now forced by noise rather than by kinks; momentum will not change it (Section 11.4.4), only extra structure will, such as strong convexity, interpolation ($B = 0$) or the variance reduction of Section 11.5.3.

11.3.7 Every deterministic result, and what randomness does to it

Table 11.1 places each result of Weeks 6 and 7 beside its sampled counterpart, and Week 13 extends the table to the whole unit.

11.4 Step sizes, batching, momentum and adaptive steps

11.4.1 The two phases of an SGD run, and practical schedules

The one-step identity (11.7) has a signal term linear in α and a noise term quadratic in it, and which is larger in magnitude changes along a run: near the start the iterate is far from x^* and the signal is large; near the floor it is small, and the noise term, which never dies at a fixed step, has caught up.

Definition 11.19 (The two phases of a run). A fixed-step SGD run is in its **gradient-dominated phase** while the magnitude of the signal term of (11.7), $2\alpha_k \nabla F(x^k)^\top (x^k - x^*)$, exceeds the noise term; it then behaves like full gradient descent at the same step. It is in its **noise-dominated phase** once the noise term has caught up; it then sits on the floor of Theorem 11.13. A smaller step buys a longer gradient-dominated phase and a lower floor.

Figure 11.3 measures the crossover on the running instance. Every step-size schedule of this week is one recipe read off the two phases: *a constant step while gradient-dominated; decay (Theorem 11.15) or average (Theorem 11.18) once noise-dominated*. The difficulty is that nothing observable announces the switch (the crossover in Figure 11.3 needed x^* to compute), so in practice the change-point is tuned, and piecewise-constant schedules are easier to tune than

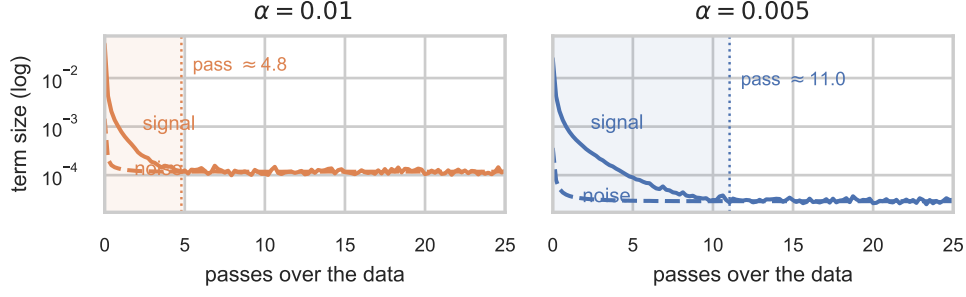


Figure 11.3: The signal and noise terms of (11.7), computed exactly along SGD runs on Example 11.7 (24 chains) at two fixed steps: the noise term first exceeds the signal at pass ≈ 4.8 for $\alpha = 0.01$ and ≈ 11.0 for $\alpha = 0.005$, the crossovers from the gradient-dominated to the noise-dominated phase.

$\alpha_k = c/k$ written literally: **epoch decay** cuts the step by a fixed factor every fixed number of passes (say $\times 0.7$ every six passes), and **epoch doubling** halves the step and holds each value twice as long as the last; both are recommended by Wright and Recht (*Optimization for Data Analysis*) as practical stand-ins for diminishing rules, and the design unit is again the pass. Machine-learning software calls α the learning rate (Section 6.5.4) and these curves **learning-rate schedules**; they are the phase recipe made explicit.

11.4.2 Mini-batching

Averaging s sampled gradients does not change the mean (Definition 11.3); it changes the spread.

Proposition 11.20 (Mini-batch variance). *Let $g_s = \frac{1}{s} \sum_{j=1}^s \nabla f_{\xi_j}(x)$ with independent draws. Then $\text{Var}(g_s) = \text{Var}(g_1)/s$, hence $\mathbb{E} \|g_s\|_2^2 = \|\nabla F(x)\|_2^2 + \text{Var}(g_1)/s$. If F is L -smooth and (L_g, B) is a noise model for g_1 , then $((L^2 + L_g^2/s)^{1/2}, B/\sqrt{s})$ is a noise model for g_s .*

Proof. Let $e_j := \nabla f_{\xi_j}(x) - \nabla F(x)$, so $\mathbb{E}[e_j] = 0$ and $g_s - \nabla F(x) = \frac{1}{s} \sum_j e_j$. Expanding the square, $\mathbb{E} \left\| \frac{1}{s} \sum_j e_j \right\|_2^2 = \frac{1}{s^2} \sum_j \mathbb{E} \|e_j\|_2^2 + \frac{1}{s^2} \sum_{j \neq l} \mathbb{E}[e_j^\top e_l]$; each cross term vanishes because independent draws factorise (Definition 11.2), $\mathbb{E}[e_j^\top e_l] = \mathbb{E}[e_j]^\top \mathbb{E}[e_l] = 0$, and each of the s diagonal terms equals $\text{Var}(g_1)$. The second-moment identity is (11.6), and the noise model follows from $\|\nabla F(x)\|_2^2 \leq L^2 \|x - x^*\|_2^2$ (L -smoothness, $\nabla F(x^*) = 0$) and $\text{Var}(g_1) \leq \mathbb{E} \|g_1\|_2^2$. $\square$

Measured at x^* on Example 11.7, the relative variance for $s = 1, 2, 4, \dots, 128$ sits on the $1/s$ line to within about 2%. But a batch of s costs s gradients, so one pass buys N/s steps instead of N : variance and step count fall together, two halves of one trade. On the other hand the s gradients of a batch are one matrix–vector product with the sampled rows, $\frac{1}{s} A_{S_k}^\top (A_{S_k} x - b_{S_k})$ for least squares, which vectorised hardware computes in little more time than a single row, so in wall-clock terms a batch step is nearly as cheap as an SGD step: batching is a hardware decision as well as a statistical one. Per pass, small batches make faster early progress and large batches are smoother later, as Figure 11.1(a) already showed for $s = 1$ against $s = 32$; measuring per *step* would flatter large batches. Carried through Theorem 11.13, the batch divides the noise constant by s , so for small α

$$\text{floor} \approx \frac{\alpha B^2}{2\mu s}.$$

This is the basis of the **linear scaling rule** of machine-learning practice: scale $\alpha \propto s$, and the floor and the number of passes to a target are unchanged while the number of *sequential* steps falls like $1/s$. On Example 11.7 the floors at $(\alpha, s) = (0.005, 1), (0.01, 2), (0.02, 4)$ are 0.0127, 0.0126, 0.0130, and with $\alpha = 0.02s$ the passes to a target stay at about 1.5 for $s = 1, \dots, 64$ while the sequential steps fall from 5850 to 98. The trade ends at the stability ceiling $2/L$ of full

gradient descent (Propositions 6.9 and 10.4): past it the step saturates, the passes bill climbs and extra batch is worthless, so the right s is an economic choice, large enough to fill the hardware and no larger than the ceiling allows.

11.4.3 Diagnostics and stopping for noisy methods

Evaluating $F(x^k)$ costs a full pass, so it cannot be the per-step stopping test, and the gradient-norm residual of Week 6 is unavailable for the same reason, while the raw per-step mini-batch loss is one noisy draw. Watch instead three cheap quantities once per pass: the *epoch average* of the mini-batch losses, which costs nothing extra and estimates the training loss along the pass; a *held-out loss* on data not used for fitting, which is what a business application cares about; and the *relative change* $\|x^{k+N/s} - x^k\|_2 / \|x^k\|_2$ across a pass. Stop on a budget of passes or when these flatten, and return the averaged iterate when the objective is merely convex. No certificate of the kind Week 5 provided declares a training run optimal; theory explains the run, and held-out data decide when it is good enough.

11.4.4 Momentum under noise

The heavy-ball update of Algorithm 10.1 added a multiple $\beta(x^k - x^{k-1})$ of the previous move to the gradient step; $\beta \in [0, 1)$ is the momentum coefficient of Week 10, the second use of the letter this week. Under sampling the same design question returns:

$$x^{k+1} = x^k - \alpha_k g^k + \beta(x^k - x^{k-1}), \quad x^{-1} := x^0. \quad (11.14)$$

Proposition 11.21 (Two forms of momentum, and the effective step). *With a constant step α , (11.14) is the same method as accumulating a running direction and stepping along it,*

$$d^{k+1} = \beta d^k + g^k, \quad x^{k+1} = x^k - \alpha d^{k+1}, \quad d^0 = 0, \quad (11.15)$$

and unrolling gives $x^{k+1} - x^k = -\alpha \sum_{j=0}^k \beta^{k-j} g^j$, a geometrically weighted sum of all past sampled gradients whose weights add up to $(1 - \beta^{k+1})/(1 - \beta) \rightarrow 1/(1 - \beta)$.

Proof. From (11.15) and $x^k - x^{k-1} = -\alpha d^k$ (both sides vanish at $k = 0$), $x^{k+1} - x^k = -\alpha \beta d^k - \alpha g^k = \beta(x^k - x^{k-1}) - \alpha g^k$, which is (11.14); unrolling $d^{k+1} = \sum_{j \leq k} \beta^{k-j} g^j$ gives the rest. $\square$

Both forms are met in software; in the second the step is along $-d^{k+1}$, so d^k is the negative of a search direction in the sense of Week 6 (as is Adam's d^k below). The proposition also warns about scale. At $\beta = 0.9$ the weights sum to 10, so in the long run momentum takes steps about ten times longer than plain SGD with the same α , and a fair comparison uses momentum at step $\alpha(1 - \beta)$. (In Example 10.1 the same step was fair: Nesterov's effective step $(\sqrt{\kappa} + 1)/(2L) = 3/L$ lay beyond the ceiling $2/L$ of plain gradient descent, Proposition 10.4; here plain SGD is held back by its noise floor instead.) Figure 11.4 measures what momentum does on a two-dimensional least-squares problem with $\kappa = 48$ and batches of $s = 8$. Plain SGD's consecutive steps are as often reversed as aligned (mean cosine between consecutive steps -0.08 , with a spike at -1): the back-and-forth that momentum was introduced to damp in Section 10.1, but caused here by sampling noise, not ill-conditioning (full gradient descent at this step, $\alpha L \approx 0.29$, moves steadily, consecutive cosines above 0.98). At $\beta = 0.9$ the mean cosine is $+0.78$: momentum really is time-averaging of directions. What that buys is the right panel: at matched effective step (momentum at $\alpha(1 - \beta) = 0.03$, effective step $0.03/(1 - 0.9) = 0.3$, against plain SGD at $\alpha = 0.3$) the two gap curves lie almost on top of each other, and that is the finding; leaving $\alpha = 0.3$ unchanged makes the effective step $\alpha/(1 - \beta) = 3$ and costs a factor of about 7 in the final gap.

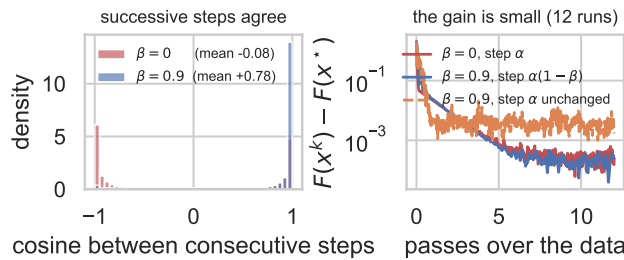


Figure 11.4: Momentum under noise on a two-dimensional least-squares instance ($N = 600$, $s = 8$, $\kappa = 48$, 12 runs). Left: cosine between consecutive steps, plain SGD and momentum with $\beta = 0.9$. Right: optimality gap against passes for plain SGD at $\alpha = 0.3$, momentum at the matched step $\alpha(1-\beta) = 0.03$ (effective step 0.3) and at the unmatched $\alpha = 0.3$ (effective step 3).

Note on the lecture slides

Frame 11.28 calls the unrolled step “a geometrically weighted *average*”, but its weights sum to $1/(1-\beta)$ (the frame’s own warning box): it is a weighted sum. Frame 11.29 calls the reversals of plain SGD “the Week 10 zig-zag”, but at this step full gradient descent does not zig-zag (Week 10 shows it only under exact line search): they are sampling noise. It also blames the unmatched run’s factor of 7 on an effective step “past the stability limit”. With $L = 0.976$ a *plain* step $3 > 2/L$ would diverge, but the momentum iteration run is stable (the roots of $r^2 - (1 - \alpha\lambda_i + \beta)r + \beta = 0$, Section 10.2, lie inside the unit circle iff $0 < \alpha\lambda_i < 2(1 + \beta)$, here $\alpha < 3.89$; largest modulus 0.949) and pays instead a higher noise floor, roughly in proportion to the effective step.

What does the theory promise? In the general noisy setting the acceleration of Week 10 does not carry over: momentum improves constants and transients, not the optimal $O(1/\sqrt{T})$ order of Theorem 11.18; stronger speedups return only in special cases, notably $B = 0$. Momentum’s help lives in the gradient-dominated phase, where consecutive directions agree; in the noise-dominated phase nothing systematic is left to accumulate. It is standard practice anyway, because transients are most of a fixed budget of passes. The practical rule: try $\beta \in [0.8, 0.95]$ on ill-conditioned smooth problems, rescale the step to $\alpha(1-\beta)$ and re-tune, and if the loss curve becomes erratic reduce α before reducing β , because an erratic curve means the effective step is too long.

11.4.5 Per-coordinate steps and Adam

One α must suit every coordinate, but business features live on wildly different scales: the conditioning problem of Example 6.14, where the large-scale coordinates force a timid step on all the others. Standardising the features fixes it in the data. The other option is to fix it inside the method, by giving coordinate j its own step, computed from its own gradient history:

$$v_j^{k+1} = \beta_2 v_j^k + (1 - \beta_2)(g_j^k)^2, \quad x_j^{k+1} = x_j^k - \frac{\alpha}{\sqrt{v_j^{k+1}} + \varepsilon} g_j^k, \quad (11.16)$$

where v_j^k is a running mean of the squared gradient of coordinate j , β_2 near 1 sets its memory, and ε is a small constant that only guards the division (not the accuracy tolerance of earlier weeks). This is a data-driven diagonal rescaling: the method standardises its own geometry as it runs, and a coordinate whose gradients are rare but large, a seldom-active customer-segment indicator or a rarely bought product, keeps a usable step instead of inheriting the timid α the large coordinates force. The price is that the update no longer moves along $-g^k$, so the guarantees of Section 11.3 do not transfer as stated: what survives is the sampling machinery, what changes is the geometry.

Adam, the default optimiser in machine-learning software, combines a running direction

with the per-coordinate step, all operations coordinatewise:

$$\begin{aligned} d^{k+1} &= \beta_1 d^k + (1 - \beta_1) g^k, & v^{k+1} &= \beta_2 v^k + (1 - \beta_2) (g^k)^2, \\ \hat{d}^{k+1} &= \frac{d^{k+1}}{1 - \beta_1^{k+1}}, & \hat{v}^{k+1} &= \frac{v^{k+1}}{1 - \beta_2^{k+1}}, & x^{k+1} &= x^k - \alpha \frac{\hat{d}^{k+1}}{\sqrt{\hat{v}^{k+1}} + \varepsilon}, \end{aligned} \quad (11.17)$$

with $d^0 = v^0 = 0$ and defaults $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\varepsilon = 10^{-8}$; β_1, β_2 are its two averaging constants, the third use of β this week. Two details differ from [Proposition 11.21](#). Adam's direction is normalised, its weights summing to 1 rather than $1/(1 - \beta_1)$, so the $\alpha(1 - \beta)$ rescaling is built in; and the divisions by $1 - \beta_1^{k+1}$ and $1 - \beta_2^{k+1}$ (**bias correction**) undo the pull of the zero initial values in the first steps. On a logistic instance with badly scaled features and one rare binary indicator, after 40 passes tuned SGD reaches loss 0.637 and Adam 0.621, against that design's optimum 0.619 (SGD has moved the rare coefficient to 0.14, Adam to 2.44, of the optimum's 3.04); after standardising the features both reach 0.611, against the standardised optimum 0.610. Much of Adam is standardisation done inside the optimiser: fix the scaling in the data first when you can, and tuned mini-batch SGD is then competitive.

Remark 11.22 (AdamW, and Adam's status). Software's "weight decay" (**AdamW**) applies $x \leftarrow (1 - \alpha\lambda)x$ *after* the step: to first order this is the proximal operator of $h = \frac{\lambda}{2} \|x\|_2^2$ from [Table 9.1](#), $\text{prox}_{\alpha h}(v) = v/(1 + \alpha\lambda) = (1 - \alpha\lambda)v + O(\alpha^2)$, so the ridge penalty is applied as in [Week 9](#), outside the adaptive scaling. Beyond the convex setting of this unit, Adam is the empirical default and is not fully understood: the original convergence proof was wrong, convex counterexamples exist, and no benchmark crowns a consistent winner over it.

11.5 Scope: constraints, nonsmooth losses, variance reduction, and when not to sample

11.5.1 Projected and proximal SGD

[Model 11.1](#) had no constraints. With a feasible set $\mathcal{X}$ that has a cheap projection ([Week 8](#)) or a convex penalty h with a cheap proximal operator ([Week 9](#)), the constrained and composite versions of the method are the updates of [Algorithms 8.2](#) and [9.1](#) with the sampled direction in place of the gradient:

$$\min_{x \in \mathcal{X}} F(x) : \quad x^{k+1} = \Pi_{\mathcal{X}}(x^k - \alpha_k g^k), \quad \min_x F(x) + h(x) : \quad x^{k+1} = \text{prox}_{\alpha_k h}(x^k - \alpha_k g^k),$$

where g^k is an unbiased sampled gradient of the smooth part F and $\Pi_{\mathcal{X}}$ is the projection of [Definition 8.10](#) (unlike [Weeks 9](#) and [10](#), where $F = g + h$ had smooth part g , here F is the smooth finite sum, the composite is $F + h$, and g is a sampled (sub)gradient). Nothing else changes: **projected SGD** and **proximal SGD** cost one sampled gradient plus one projection or prox per step, and large-scale lasso ([Model 9.3](#), prox = soft-thresholding) and simplex-constrained portfolio fitting ([Proposition 8.13](#)) are the standard examples. They converge for the same reason as their full-gradient parents: both operators are non-expansive ([Propositions 8.12](#) and [9.10](#)), so they cannot amplify the sampling noise. For projected SGD, $x^* = \Pi_{\mathcal{X}}(x^*)$ gives $\|x^{k+1} - x^*\|_2 \leq \|x^k - \alpha_k g^k - x^*\|_2$, so (11.7) holds with " $\leq$ " and the noise term is carried through untouched. The floor therefore behaves exactly as in the unconstrained case, proportional to α and removable by decay: the operators of [Weeks 8](#) and [9](#) compose with sampling.

11.5.2 The stochastic subgradient method

When the per-record loss has kinks, the sampled gradient is replaced by a sampled subgradient ([Definition 7.1](#)),

$$x^{k+1} = x^k - \alpha_k g^k, \quad g^k \in \partial f_{\xi_k}(x^k),$$

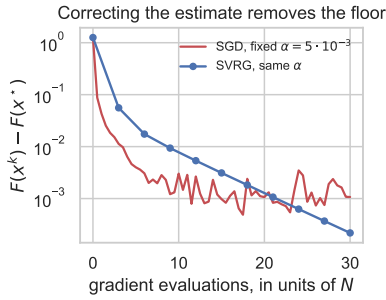


Figure 11.5: SGD and SVRG at the same fixed step $\alpha = 5 \cdot 10^{-3}$ on [Example 11.7](#): optimality gap against gradient evaluations in units of N (SVRG's markers are one epoch, $3N$ evaluations, apart). SGD reaches its floor near 10^{-3} by pass 20; SVRG's gap falls from 1.26 to $2.2 \cdot 10^{-4}$ after 30 units, by a factor of about 1.7 per epoch.

and the only new ingredient relative to [Algorithm 7.1](#) is that we sample which f_i to differentiate. The running examples are one term of the LAD objective of [Model 7.6](#), $f_i(x) = |a_i^\top x - b_i|$, with sampled subgradient $\text{sign}(a_i^\top x - b_i) a_i$, and one hinge term of the SVM of [Model 4.17](#), $f_i(x) = \max\{0, 1 - b_i a_i^\top x\}$, with sampled subgradient $-b_i a_i$ if $b_i a_i^\top x < 1$ and 0 otherwise; both are valid at the kinks too ([Proposition 7.4](#)). Here x is the β of [Model 7.6](#) and the w of [Model 4.17](#), whose labels y_i (so named as b was the intercept) become b_i ; the intercept is absorbed into a constant feature and $\frac{1}{2} \|w\|_2^2$ is left out. Unbiasedness holds in the appropriate sense: if $g_i \in \partial f_i(x)$ for every i then $\frac{1}{N} \sum_i g_i \in \partial F(x)$ by the sum rule, so $\mathbb{E}[g^k | x^k] \in \partial F(x^k)$, and the subgradient inequality gives $\mathbb{E}[g^k | x^k]^\top (x^k - x^*) \geq F(x^k) - F^*$, exactly what step (1) of the proof of [Theorem 11.18](#) used. Since $\|g^k\|_2 \leq \|a_{\xi_k}\|_2$, the bound $\mathbb{E}[\|g^k\|_2^2 | x^k] \leq G^2$ holds with $G = a_{\max}$ for both examples, and the theorem holds verbatim: $\mathbb{E}[F(\bar{x}^T)] - F^* = O(1/\sqrt{T})$ for the averaged iterate. Nonsmoothness costs nothing extra: the sampled smooth case and the deterministic subgradient method ([Corollary 7.15](#)) already sat at this order, optimal by [Remark 7.17](#). As in [Week 7](#), monotone descent is not expected and averaging matters even more, because now both the nonsmoothness and the sampling push against the last iterate.

11.5.3 Variance reduction: SVRG

[Corollary 11.14](#) showed that a fixed step converges linearly when the sampled gradients vanish at x^* , which real data do not provide. Variance reduction manufactures the property. **SVRG** keeps a **snapshot** $\tilde{x}$, refreshed once per pass, together with its full gradient $\nabla F(\tilde{x})$, and steps along the **corrected direction**

$$g^k = \nabla f_{i_k}(x^k) - \nabla f_{i_k}(\tilde{x}) + \nabla F(\tilde{x}), \quad x^{k+1} = x^k - \alpha g^k,$$

with i_k a fresh uniform draw (the ξ_k of [Section 11.2](#)) and a fixed step α . Since $\mathbb{E}[\nabla f_{i_k}(\tilde{x})] = \nabla F(\tilde{x})$, the two extra terms cancel in expectation and $\mathbb{E}[g^k | x^k] = \nabla F(x^k)$: the direction is still unbiased. But its spread now vanishes as x^k and $\tilde{x}$ approach x^* : for least squares $g^k - \nabla F(x^k) = (a_{i_k} a_{i_k}^\top - \frac{1}{N} A^\top A)(x^k - \tilde{x})$, which is zero whenever $x^k = \tilde{x}$, whatever the residuals. SVRG thus *buys* $B = 0$ on any finite dataset, and linear convergence at a fixed step returns (stated here, not proved), as [Figure 11.5](#) shows on the running instance. The price is in the cost accounting: each inner step evaluates two sampled gradients and each snapshot a full one, $3N$ gradient evaluations per epoch of N inner steps (SAGA instead keeps a table of stored per-sample gradients, trading memory for the re-pass). Variance reduction is worth its machinery when a fixed dataset is revisited many times and high accuracy is wanted; it is not available for streaming data, where there is no dataset to take a snapshot of.

11.5.4 When not to use SGD, and choosing a method

Sampling buys speed by giving up exactness, and there are three situations in which that is a bad trade. When N is moderate and one pass is cheap, the methods of [Weeks 6](#) and [10](#) win outright and carry guarantees SGD does not: convergence with no floor (linear under strong convexity,

Theorem 10.6, $O(1/k^2)$ in the objective otherwise, **Theorem 10.11**) and a checkable residual. When accuracy is the goal, the floor is proportional to α and the iteration count of [Section 11.3.4](#) grows like $1/\varepsilon$. And when the model is solver-ready, a moderate LP, QP or SOCP belongs in the modelling route of [Section 4.5](#), not in a hand-written loop (RSOME's role this week is only as a small-instance check). Otherwise the choice among the first-order families is a short decision procedure. Is one pass over the data cheap? If yes, use full-gradient or accelerated methods, where per-step quality is high. If not, sample: SGD or mini-batch SGD for a smooth objective, with momentum if it is ill-conditioned; the stochastic subgradient method, with averaging, for a nonsmooth one, or proximal SGD when the nonsmooth part is a simple h ; $\Pi_{\mathcal{X}}$ around either sampled branch for constraints; SVRG or SAGA for a fixed dataset revisited many times at high accuracy; per-coordinate steps for badly scaled features that cannot be standardised. The right question is no longer which theorem is strongest, but which information is cheap and reliable for the problem at hand.

11.6 Summary and looking ahead

The pass over the data, not the step, is the unit of cost. A sampled gradient is unbiased but noisy, and the one-step identity (11.7) of [Algorithm 11.1](#) is [Week 7](#)'s inequality with $\alpha_k^2 \mathbb{E} \|g^k\|_2^2$ in place of $\alpha_k^2 G^2$: a fixed step converges linearly to a floor proportional to α , absent under interpolation and removed by a decaying step at the price of $O(1/k)$, and without strong convexity the averaged iterate attains the optimal $O(1/\sqrt{T})$. Schedules follow the two phases of a run; batching, momentum and averaging manage the variance without changing these orders; projection, prox and subgradients compose with sampling; and variance reduction buys $B = 0$. SGD is the right default for huge or streaming data at rough-to-medium accuracy and the wrong default almost everywhere else; in machine-learning software, step size is learning rate, pass is epoch, schedules are the phase recipe, and Adam is momentum plus per-coordinate steps.

This week cut the cost of each step by looking at less *data*, the answer to the first of three sources of scale. [Week 12](#) takes up the other two, too many variables and too much coupling, by solving the cheapest meaningful subproblem at each step: (block) coordinate descent updates one coordinate or block at a time, and dual decomposition and ADMM split a coupled problem into local pieces coordinated by a price. Its guarantees for randomized coordinate selection hold in expectation ([Definition 11.2](#)), and its cost is again counted in passes over the data. [Week 13](#) then places SGD's row, $O(1/\sqrt{T})$ on $\bar{x}^T$ and optimal, in the ladder of the whole unit.

11.7 Exercises

Exercise 11.1 (Cost model and sampled gradients). For [Model 11.1](#) explain what one *iteration* and one *pass over the data* mean for full gradient descent, for SGD and for mini-batch SGD with batch size s . Derive the per-sample gradients for least squares and for logistic regression, verify that averaging the sampled gradients over the draw gives the full gradient in each case, and say which property of expectation you used. If $\mathbb{E}[g \mid x^k] = \nabla F(x^k)$ and α_k is a number, what is $\mathbb{E}[x^k - \alpha_k g \mid x^k]$? Finally, a colleague reports that mini-batch SGD with $s = 256$ “converges in far fewer steps” than $s = 1$: what is wrong with that comparison?

Exercise 11.2 (Step-size intuition from the mean example). For $\min_x \frac{1}{2N} \sum_{i=1}^N (x - \omega_i)^2$ write the SGD update for one drawn value $\omega_{(k+1)}$ and show that with $\alpha_k = 1/(k+1)$ the iterate x^{k+1} is the running average of the first $k+1$ drawn values for *any starting point* x^0 . Explain in words why this supports diminishing step sizes for noisy methods. What happens instead if $\alpha_k = \alpha$ is held constant? Describe the resulting estimator in words.

Exercise 11.3 (Signal versus noise). Starting from $x^{k+1} = x^k - \alpha_k g^k$, expand $\mathbb{E}[\|x^{k+1} - x^\star\|_2^2 \mid x^k]$ and identify the helpful and the harmful term, together with the power of α_k multiplying

each. Using $A_{k+1} \leq \rho A_k + \alpha^2 B^2$ with $1 - \rho \approx 2\alpha\mu$, predict how the noise floor changes when α is halved, then check the prediction numerically on a least-squares instance.

Exercise 11.4 (Momentum and per-coordinate steps, fairly compared). Implement momentum SGD (11.14) for least squares or logistic regression. Compare it with plain SGD at the same α , and then at $\alpha(1 - \beta)$. Which comparison is fair, and why? Then implement the per-coordinate update (11.16). On a least-squares instance whose features you have *not* standardised, compare it with plain SGD, each at its best step from a small grid; then standardise and compare again. What does the result say about what the per-coordinate step is doing?

Exercise 11.5 (Nonsmooth and constrained). For LAD regression or hinge-loss minimisation write down a stochastic subgradient update, implement it with iterate averaging, and compare the averaged iterate with the raw final iterate. Then add the constraint $\mathbf{1}^\top x = 1$, $x \geq 0$ and run projected SGD, reusing the simplex projection of Proposition 8.13. Does the noise floor behave as it did without the constraint?

Exercise 11.6 (A design memo). In a short paragraph explain when plain SGD is preferable, when an SVRG-style method is worth the extra machinery, and when you would not use a stochastic method at all. As a coding extension, compare fixed-step SGD, epoch-decay SGD, mini-batch SGD, momentum SGD and Adam (11.17) on one dataset, raw and standardised, plotting the objective against both passes over the data and wall-clock time.

Week 12 Large-Scale Optimisation and Decomposition

What is the cheapest meaningful subproblem we can solve at each step? Every method from [Week 6](#) to [Week 10](#) moved the *whole* decision vector at once. [Week 11](#) met the first reason this can be too expensive, *too many samples*, and answered it by sampling. This week meets the other two: *too many variables*, so that a full-vector update is wasteful, and *too much coupling*, so that easy local pieces are tied by a shared constraint. In both cases the remedy is a smaller *subproblem* (one coordinate, one block or one local model at a time) when that is far cheaper than a full step.

Nothing in the week is built from scratch: the quadratic upper model and rates of [Week 6](#) reappear one coordinate at a time; the proximal operator of [Week 9](#) reappears as a scalar soft-threshold and, later, as one half-step of ADMM; the Lagrangian and dual function of [Week 5](#) become an algorithm in which the multiplier is a price that coordinates independent units; and the expectation and “cost per pass” accounting of [Week 11](#) are the language of the guarantees.

[Section 12.1](#) separates the three sources of scale and fixes an honest unit of work. [Section 12.2](#) builds coordinate descent as [Week 6](#) built gradient descent, and [Section 12.3](#) extends it to blocks and to smooth-plus-separable objectives such as the lasso. [Section 12.4](#) shows how a price decomposes a problem coupled by a shared resource; [Section 12.5](#) stabilises the price with an augmented Lagrangian, recovers separability by splitting, and arrives at ADMM.

Four conventions apply throughout. The step size is α , as in every week since [Week 7](#); λ is both the lasso penalty and the price of a coupling constraint, and the text says which is meant; m counts business units, regions or ADMM blocks (block coordinate descent has J blocks), not inequality constraints, and R is a shared resource; and a_i is *column* i of the data matrix A , whereas in [Week 11](#) $a_i^\top$ was row i , because a coordinate update touches one feature, not one observation. Accordingly i now indexes coordinates (later units or blocks), and $\nabla_i f(x) = \partial f / \partial x_i$ is a partial derivative, a number, not the per-record gradient ∇f_i of [Week 11](#). Least squares is $\frac{1}{2} \|Ax - b\|_2^2$, as in [Week 6](#), without the $1/N$ of [Model 11.1](#): every smoothness constant is N times larger, but the minimiser is unchanged.

12.1 Structure as a source of scale

12.1.1 Three bottlenecks: samples, variables, coupling

Three models recur through the week. *Sparse prediction* is the lasso of [Model 9.3](#), which minimises $\frac{1}{2} \|Ax - b\|_2^2 + \lambda \|x\|_1$ over x and in which the feature dimension d is large and most coefficients should end up exactly zero. *Shared budget allocation* has m business units that each optimise a local objective but compete for one resource pool, $\sum_{i=1}^m x_i = R$. *Consensus* has m local models, each fitted to its own data, that must agree on a common vector: $x_i = z$ for every i , or $x - z = 0$ for a single split variable. The first is large in d ; the other two are large because of coupling. Too many samples were answered in [Week 11](#) by sampling ([Algorithm 11.1](#)); too many variables are answered by updating one coordinate or block at a time, too much coupling by prices and splitting.

12.1.2 The coordinate idea and an honest unit of work

Let $e_i \in \mathbb{R}^d$ be the i -th standard unit vector, with a one in position i and zeros elsewhere. Gradient descent ([Algorithm 6.1](#)) takes the **full step** $x^{k+1} = x^k - \alpha \nabla f(x^k)$, moving every coordinate. A **coordinate step** changes one coordinate only,

$$x^{k+1} = x^k + u_k e_{i_k},$$

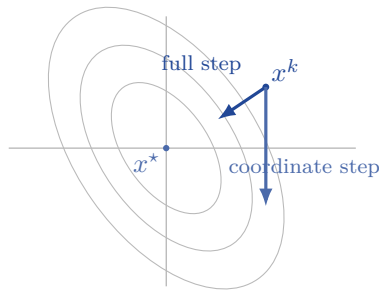


Figure 12.1: Level sets of the quadratic of Example 12.1. From x^k the full step with $\alpha = 0.5/L$ moves diagonally towards x^* ; the coordinate step moves only along x_2 , to the minimiser on that vertical line.

where $i_k \in \{1, \dots, d\}$ is the coordinate chosen at iteration k and $u_k \in \mathbb{R}$ is a scalar increment; the move is axis-aligned. Figure 12.1 compares the two.

Example 12.1 (Full step versus coordinate step). Take $f(x) = \frac{1}{2}x^\top Qx$ with $Q = \begin{pmatrix} 0.776 & 0.320 \\ 0.320 & 0.544 \end{pmatrix}$, whose level sets are ellipses rotated by 35° ; the largest curvature is $L = 1$ and the smallest $\mu = 0.320$. From $x^k = (1.550, 0.950)$, where $f(x^k) = 1.649$, the full step with $\alpha = 0.5/L$ gives $x^k - 0.5 Qx^k = (0.797, 0.444)$, and f falls to 25% of its starting value. The exact coordinate step along x_2 keeps $x_1 = 1.550$ and sets $x_2 = -Q_{21}x_1/Q_{22} = -0.912$, the minimiser of the one-variable quadratic $f(1.550, x_2)$; f falls to 43%. A single coordinate step typically makes less progress than a full step, because it uses one partial derivative where the full step uses all d (this is a rule of thumb, not a theorem: along x_1 instead of x_2 the same coordinate step would have reached 11%). What makes the trade worthwhile is that the coordinate step can be far cheaper.

Cheaper by how much needs a fair unit of cost. For least squares $f(x) = \frac{1}{2}\|Ax - b\|_2^2$ (Model 6.16) with $A \in \mathbb{R}^{N \times d}$, a full gradient $A^\top(Ax - b)$ costs $O(Nd)$ operations, whereas a coordinate update needs only the partial derivative $\nabla_i f(x) = a_i^\top r$, with $r = Ax - b$ the residual and a_i column i of A , which costs $O(N)$ if r is stored and refreshed after each update. So d coordinate updates cost about one full gradient, and we call them one **full-gradient equivalent**. Like a pass over the data in Definition 11.6 (N per-record gradients, also $O(Nd)$), it reads the whole data matrix once, so this week also calls it an epoch or a pass. In this unit the coordinate method's weaker steps are no handicap: on a least-squares instance with $N = 2000$ and $d = 200$ (Figure 12.2a), after 30 full-gradient equivalents gradient descent ($\alpha = 1/L$) has an optimality gap of 1.6×10^2 while randomised coordinate descent has reached 5.7×10^{-9} . Coordinate descent wins when the cost per update falls much faster than the number of updates rises.

12.1.3 Three structural patterns to recognise

Three patterns account for most of what this week exploits. A **separable regulariser** is a sum of functions of single coordinates, $\sum_{i=1}^d h_i(x_i)$, such as $\|x\|_1 = \sum_i |x_i|$; separability is what makes scalar proximal steps cheap. **Natural block variables** arise when the coordinates group themselves, $x = (x_{(1)}, \dots, x_{(J)})$ with J blocks, because a product family, a region or a marketing channel across all periods shares data and constraints; then a block is the right unit of work. **Coupling constraints** tie otherwise separable pieces together, $\sum_i A_i x_i = b$ or $x = z$, and Lagrange multipliers will coordinate the pieces. Model recognition is the main skill of the week: is the model separable, block-structured, or coupled by a shared constraint?

12.2 Coordinate descent

12.2.1 The lasso as laboratory, and exact coordinate minimisation

The lasso of [Model 9.3](#), written this week in the variable x ,

$$\min_{x \in \mathbb{R}^d} \frac{1}{2} \|Ax - b\|_2^2 + \lambda \|x\|_1, \quad \lambda > 0,$$

is the laboratory for the whole week. Its dimension d can be large, but a step on coordinate i needs only one partial derivative of the smooth part and one scalar shrinkage step, because the penalty is separable. We first develop the smooth case $f(x) = \frac{1}{2} \|Ax - b\|_2^2$ and return to the penalty in [Section 12.3.2](#). The most basic coordinate move holds all coordinates but one fixed and solves the resulting one-variable problem exactly.

Definition 12.2 (Coordinate slice and exact coordinate minimisation). For $f : \mathbb{R}^d \rightarrow \mathbb{R}$, a point x and a coordinate i , the **slice** of f along coordinate i is the one-variable function $\phi_i(u) = f(x + ue_i)$, $u \in \mathbb{R}$; $u = 0$ is the current point. **Exact coordinate minimisation** chooses $u^* \in \arg\min_{u \in \mathbb{R}} \phi_i(u)$ and sets $x_i \leftarrow x_i + u^*$, all other coordinates unchanged.

For a quadratic such as least squares, ϕ_i is a one-variable quadratic, $\phi_i(u) = f(x) + \nabla_i f(x) u + \frac{1}{2} (\partial^2 f / \partial x_i^2) u^2$, minimised in closed form. For a general smooth f it is not, and exact minimisation gives way to the coordinate analogue of the gradient step.

12.2.2 Coordinate-wise smoothness and the coordinate gradient step

[Week 6](#) built the gradient step from one global quadratic upper model, valid because the gradient is L -Lipschitz ([Definition 6.8](#)). Coordinate descent uses the same idea along one coordinate at a time, and the relevant constant bounds how fast the *partial* derivative $\nabla_i f(x) = \partial f / \partial x_i$ changes along e_i .

Definition 12.3 (Coordinate-wise Lipschitz constants). A differentiable $f : \mathbb{R}^d \rightarrow \mathbb{R}$ has **coordinate-wise Lipschitz constants** $L_1, \dots, L_d > 0$ if for every coordinate i ,

$$|\nabla_i f(x + ue_i) - \nabla_i f(x)| \leq L_i |u| \quad \text{for all } x \in \mathbb{R}^d, u \in \mathbb{R}. \quad (12.1)$$

For a twice differentiable convex f this holds whenever $0 \leq \partial^2 f / \partial x_i^2 \leq L_i$ everywhere. For least squares, $f(x) = \frac{1}{2} \|Ax - b\|_2^2$, one can take $L_i = \|a_i\|_2^2$, the squared norm of column i of A . The constants imply the coordinate upper model of [Proposition 12.4](#), and for a convex, twice differentiable f the smallest such constants satisfy $\max_i L_i \leq L \leq \sum_i L_i$, with L the global constant of [Definition 6.8](#) ([Equation \(12.4\)](#) below).

Proposition 12.4 (Coordinate upper model and coordinate step). *Let f satisfy (12.1). Then for every x , i and $u \in \mathbb{R}$,*

$$f(x + ue_i) \leq q_i(u) := f(x) + \nabla_i f(x) u + \frac{L_i}{2} u^2. \quad (12.2)$$

The upper model q_i is minimised where $0 = \nabla_i f(x) + L_i u^*$, that is at $u^* = -\nabla_i f(x) / L_i$, and the resulting point satisfies

$$f\left(x - \frac{1}{L_i} \nabla_i f(x) e_i\right) \leq f(x) - \frac{1}{2L_i} (\nabla_i f(x))^2. \quad (12.3)$$

Proof. The slice $\phi_i(u) = f(x + ue_i)$ has derivative $\phi'_i(u) = \nabla_i f(x + ue_i)$, which by (12.1) is L_i -Lipschitz on $\mathbb{R}$, so the one-dimensional descent lemma ([Proposition 6.9](#) with $d = 1$) gives (12.2). Substituting u^* into q_i gives $q_i(u^*) = f(x) - (\nabla_i f(x))^2 / (2L_i) \geq f(x + u^* e_i)$. $\square$

Here $\nabla_i f(x)$ is the local slope in direction e_i , L_i the curvature along that coordinate, and q_i a one-dimensional quadratic model. Minimising it gives the **smooth coordinate-descent update**

$$x_i^{k+1} = x_i^k - \frac{1}{L_i} \nabla_i f(x^k), \quad x_j^{k+1} = x_j^k \text{ for } j \neq i,$$

the gradient step of [Week 6](#) with a coordinate-specific step $1/L_i$ on a single coordinate. For least squares with $L_i = \|a_i\|_2^2$ the slice is q_i , so the $1/L_i$ step is exact coordinate minimisation.

Why not use the global constant L of [Definition 6.8](#) for every coordinate? For a convex, twice differentiable f with Hessian $\nabla^2 f(x)$, and the smallest valid constants,

$$\max_i L_i \leq L \leq \sum_{i=1}^d L_i, \quad (12.4)$$

because $(\nabla^2 f(x))_{ii} = e_i^\top \nabla^2 f(x) e_i$ is at most the largest eigenvalue of $\nabla^2 f(x)$, which for a positive semidefinite matrix is at most the trace $\sum_i (\nabla^2 f(x))_{ii}$; taking the supremum over x gives (12.4). The global constant is set by the worst *direction*, which can be steeper than every coordinate axis, so one step $1/L$ is timid on every coordinate flatter than the worst one. Two axis-aligned quadratics show the compromise. With the global step $1/L$, $L = 1$, the round quadratic $\frac{1}{2}(x_1^2 + x_2^2)$ is solved exactly along each coordinate; on the stretched quadratic $\frac{1}{2}(0.2x_1^2 + x_2^2)$, where $L_1 = 0.2$, the same step from $x_1 = 1.55$ reaches only $x_1 = 1.24$, 20% of the way along the flat coordinate, whereas the coordinate step $1/L_1$ reaches the minimiser at once.

Note on the lecture slides

Frame 12.11 writes the global constant as $L = \max_i L_i$. That holds for the two axis-aligned quadratics drawn there (diagonal Hessians) but not in general; the correct relation is the sandwich (12.4). The deck's own tilted quadratic of [Example 12.1](#) has $L = 1$ but $\max_i L_i = Q_{11} = 0.776$, and the least-squares instance of [Figure 12.2a](#) has $L = 2248$ against $\max_i L_i = 2036$. The frame's takeaway says the global step is set by the “worst coordinate”; in general it is set by the worst *direction*, which can be steeper than every coordinate axis, and this only strengthens the case for coordinate-wise steps.

12.2.3 When an update is cheap, and which coordinate to pick

Coordinate descent is mainly a computational design choice, and three features make an update cheap. With *sparse features*, column a_i has few nonzeros and $a_i^\top r$ costs $O(\text{nnz}(a_i))$, the number of nonzero entries of a_i , rather than $O(N)$. With a *cached residual*, $r = Ax - b$ is stored and refreshed by the rank-one update $r \leftarrow r + a_i(x_i^+ - x_i)$ after each change of x_i , so no full product Ax is ever recomputed. With a *simple scalar prox*, a separable penalty such as the lasso's turns the one-dimensional step into a closed formula ([Section 12.3.2](#)).

Which coordinate to update is the **selection rule**. The **cyclic** rule visits $1 \rightarrow 2 \rightarrow \dots \rightarrow d$ and repeats; it is predictable and cache-friendly. The **randomised** rule draws i_k uniformly from $\{1, \dots, d\}$, independently at each step; it has the cleanest theory, in expectation over the draws exactly as in [Week 11](#). The **greedy** rule picks the coordinate with the largest $|\nabla_i f(x^k)|$; it makes good progress per update but needs the whole gradient to choose, hence more bookkeeping. A practical compromise, **random permutation**, reshuffles the order at the start of every epoch and then sweeps it: it keeps the simplicity of the cyclic rule while avoiding orderings that happen to be bad for the instance. [Algorithm 12.1](#) collects the method.

One update costs one partial derivative and one scalar operation, $O(N)$ for least squares with the cached residual; one epoch of d updates costs $O(Nd)$, a full-gradient equivalent. A stopping rule is the gradient-norm test of [Week 6](#), checked once per epoch at the cost of one more full-gradient equivalent; to save it, check every few epochs or monitor the change in x over an epoch.

Algorithm 12.1 Coordinate descent for a smooth f (least-squares specialisation in comments)

Require: routine for $\nabla_i f$; constants L_i (Definition 12.3); starting point x^0 ; selection rule; tolerance $\varepsilon > 0$; epoch cap K

Ensure: an approximate minimiser of f

```

1:  $x \leftarrow x^0$  ▷ least squares: also  $r \leftarrow Ax - b$ ,  $L_i = \|a_i\|_2^2$ 
2: for epoch = 1, ...,  $K$  do
3:   for  $d$  updates do
4:     choose  $i$  by the selection rule ▷ cyclic, uniform random, reshuffled or greedy
5:      $x_i^+ \leftarrow x_i - \nabla_i f(x)/L_i$  ▷ or exact minimisation of  $\phi_i$ ; LS:  $\nabla_i f(x) = a_i^\top r$ 
6:     ▷ least squares:  $r \leftarrow r + a_i(x_i^+ - x_i)$ , an  $O(N)$  refresh
7:      $x_i \leftarrow x_i^+$ 
8:   end for
9:   if  $\|\nabla f(x)\|_2 \leq \varepsilon$  then return  $x$  ▷ checked once per epoch; LS:  $\|A^\top r\|_2$ 
10:  end if
11: end for
12: return  $x$ 

```

12.2.4 Convergence guarantees

The guarantees are stated in a norm that weights each coordinate by its own constant.

Definition 12.5 (Weighted norm, radius and weighted strong convexity). For constants $L_1, \dots, L_d > 0$ the **weighted norm** is $\|v\|_L^2 = \sum_{i=1}^d L_i v_i^2$. For a starting point x^0 and a minimiser x^* of f , the **radius** $R_L^2 = \max\{\|x - x^*\|_L^2 : f(x) \leq f(x^0)\}$ measures the start-ing sublevel set in that norm. The function f is μ_L -**strongly convex in** $\|\cdot\|_L$ if $f(y) \geq f(x) + \nabla f(x)^\top (y - x) + \frac{\mu_L}{2} \|y - x\|_L^2$ for all x, y , the inequality of Definition 6.12 with $\|\cdot\|_2$ replaced by $\|\cdot\|_L$. One can show that $\mu/\max_i L_i \leq \mu_L \leq 1$ when f is μ -strongly convex in $\|\cdot\|_2$.

Theorem 12.6 (Randomised coordinate descent, convex case). *Let f be convex and differentiable with coordinate-wise constants L_i (Definition 12.3), and suppose a minimiser x^* exists and $R_L < \infty$. Choose i_k uniformly from $\{1, \dots, d\}$, independently at each step, and update $x^{k+1} = x^k - \frac{1}{L_{i_k}} \nabla_{i_k} f(x^k) e_{i_k}$. Then, with $\mathbb{E}$ the expectation over the coordinate draws $i_0, \dots, i_{k-1}$ (Definition 11.2, with the d coordinates in place of the N records, each drawn with probability $1/d$),*

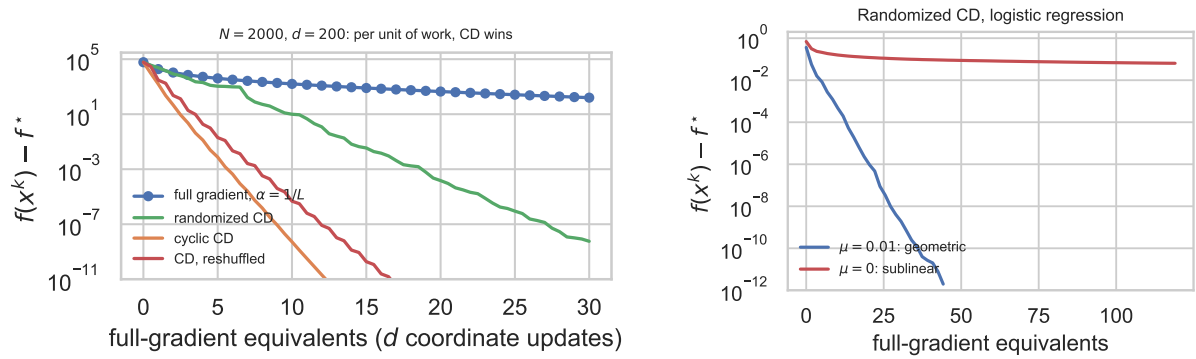
$$\mathbb{E}[f(x^k) - f^*] \leq \frac{2dR_L^2}{k} = O\left(\frac{1}{k}\right). \quad (12.5)$$

Theorem 12.7 (Randomised coordinate descent, strongly convex case). *Under the hypotheses of Theorem 12.6, if f is in addition μ_L -strongly convex in $\|\cdot\|_L$, then*

$$\mathbb{E}[f(x^k) - f^*] \leq \left(1 - \frac{\mu_L}{d}\right)^k (f(x^0) - f^*). \quad (12.6)$$

Proof sketch of Theorem 12.7. Averaging (12.3) over the uniform draw of i_k with x^k fixed, the conditional expectation of Section 11.3, gives $\mathbb{E}[f(x^{k+1}) | x^k] \leq f(x^k) - \frac{1}{2d} \sum_i (\nabla_i f(x^k))^2 / L_i$. Minimising both sides of the μ_L -strong-convexity inequality over y gives the weighted gradient-dominance inequality $f(x) - f^* \leq \frac{1}{2\mu_L} \sum_i (\nabla_i f(x))^2 / L_i$, the analogue of the one used in Theorem 6.13. Combining the two contracts the expected gap by the factor $1 - \mu_L/d$ per update; taking full expectations by the tower rule (Section 11.3) and iterating gives (12.6). $\square$

Accuracy ε in expectation is guaranteed in the convex case after $k \geq 2dR_L^2/\varepsilon$ updates, that is $2R_L^2/\varepsilon$ epochs: the factor d in (12.5) is absorbed by counting in full-gradient equivalents, which is why epochs are the fair axis. In the strongly convex case one epoch contracts the expected gap



(a) Least squares, $N = 2000$, $d = 200$: gap $f(x^k) - f^*$ against full-gradient equivalents for gradient descent ($\alpha = 1/L$) and randomised, cyclic and reshuffled coordinate descent (steps $1/L_i$).

(b) Randomised coordinate descent on logistic regression ($N = 2000$, $d = 50$): with a ridge term $\mu = 0.01$ the gap falls along a straight line; without it the data are separable, $\inf f = 0$ is not attained, and the plotted $f(x^k) - \inf f$ bends away.

Figure 12.2: Measured behaviour of coordinate descent (log axes). Left: per unit of work every coordinate rule is far ahead of the full gradient. Right: linear versus sublinear decay.

by $(1 - \mu_L/d)^d \approx e^{-\mu_L}$. Neither this nor the factor $1 - \mu/L$ per gradient step (Theorem 6.13) is better in general, but μ_L can be far larger than μ/L , because weighting each coordinate by its own L_i can improve the conditioning: on the instance of Figure 12.2a, $\mu/L = 0.027$ while $\mu_L = 0.483$. As in Week 6, strong convexity turns $1/k$ into geometric decay, a straight line on a log axis (Figure 12.2b); on a linear axis the two would look alike, so rates are read on a log axis.

Note on the lecture slides

Frame 12.14 states the randomised guarantee “under the previous assumptions”, listed informally afterwards; frames 12.15–12.16 add strong convexity and the cyclic rule. No frame says that the constants $2d$ and $1 - \mu_L/d$ need *uniform*, independent draws and a minimiser with $R_L < \infty$; the theorems above state these hypotheses, and the constants are correct. The “ $\mu = 0$: sublinear” curve of frame 12.15 (Figure 12.2b) is *not* an instance of the $O(1/k)$ theorem: without the ridge term no minimiser exists, and the decay is slower than $1/k$ (the number of passes times $f(x^k)$ grows from 2.2 at pass 17 to 7.5 at pass 117). It illustrates what strong convexity buys, not the convex-case bound.

Proposition 12.8 (Cyclic coordinate descent). *Let f be convex and differentiable with coordinate-wise constants L_i , and suppose the sublevel set $\{x : f(x) \leq f(x^0)\}$ is bounded, so that a minimiser exists. Cyclic coordinate descent with the steps $1/L_i$, or with exact coordinate minimisation, converges: $f(x^k) \rightarrow f^*$. If f is strongly convex the convergence is linear. The constants depend on the coordinate ordering and on how strongly the coordinates are coupled, and are less clean than in the randomised case.*

Stated without proof, the proposition says that cyclic and reshuffled sweeps are safe defaults and often strong empirically, while the randomised rule is the one whose guarantee has explicit constants. In Figure 12.2a all three rules reach gaps below 10^{-8} within 30 passes, the cyclic and reshuffled sweeps even faster than uniform sampling, whereas gradient descent, which has the strongest deterministic theory, is still above 10^2 . Reshuffling each epoch costs nothing and protects against an adversarial ordering, which makes random permutation a strong practical default.

12.2.5 Worked example: cyclic coordinate descent for least squares

For $f(x) = \frac{1}{2} \|Ax - b\|_2^2$ with residual $r = Ax - b$ the chain rule (Proposition 1.17) gives $\nabla f(x) = A^\top r$, whose i -th entry is $\nabla_i f(x) = a_i^\top r$. The slice along coordinate i has second derivative $\|a_i\|_2^2 = L_i$, so the $1/L_i$ step is the exact coordinate minimiser and, with the residual refreshed after each update, one pass of cyclic coordinate descent reads

$$x_i^+ = x_i - \frac{a_i^\top r}{\|a_i\|_2^2}, \quad r^+ = r + a_i(x_i^+ - x_i), \quad i = 1, \dots, d. \quad (12.7)$$

The refresh is correct because $Ax^+ - b = Ax - b + a_i(x_i^+ - x_i)$ when only coordinate i changes. Each update is one inner product and one vector addition, $O(N)$, where a full gradient costs $O(Nd)$: residual caching is why coordinate descent can be competitive.

Example 12.9 (Two passes by hand). Let $A = \begin{pmatrix} 1 & 0 \\ 1 & 1 \\ 0 & 1 \end{pmatrix}$, $b = (1, 2, 3)$ and $x^0 = 0$, so $r = -b = (-1, -2, -3)$, $f(x^0) = 7$ and $L_1 = L_2 = 2$; the normal equations give $x^* = (1/3, 7/3)$ with $f^* = 2/3$. *Pass 1.* Coordinate 1: the slope is $a_1^\top r = -1 - 2 = -3$, so $x_1 = 0 - (-3)/2 = 1.5$ and $r \leftarrow r + 1.5a_1 = (0.5, -0.5, -3)$, with $f = 4.75$. Coordinate 2: $a_2^\top r = -0.5 - 3 = -3.5$, so $x_2 = 1.75$ and $r \leftarrow (0.5, 1.25, -1.25)$, with $f = 1.6875$. *Pass 2.* Coordinate 1: $a_1^\top r = 1.75$, $x_1 = 1.5 - 0.875 = 0.625$, $r \leftarrow (-0.375, 0.375, -1.25)$, $f = 0.9219$. Coordinate 2: $a_2^\top r = -0.875$, $x_2 = 2.1875$, $r \leftarrow (-0.375, 0.8125, -0.8125)$, $f = 0.7305$. After two passes, two full-gradient equivalents, the gap $f - f^*$ is 0.064; on this instance it shrinks by exactly $\frac{1}{4}$ at every update after the first. Two gradient-descent steps with $\alpha = 1/L$ ($L = 3$, the largest eigenvalue of $A^\top A$) cost the same and reach $f = 70/81 \approx 0.864$, a gap three times larger.

12.3 From coordinates to blocks, and proximal-coordinate steps

12.3.1 Block coordinate descent

Sometimes one coordinate is too small a unit of work. Marketing spend for one channel across many time periods forms a natural group, as does the portfolio vector of one business unit: the variables share covariates, constraints or interpretation, and are decided together. Partition the coordinates into J **blocks**, $x = (x_{(1)}, \dots, x_{(J)})$ with $x_{(j)} \in \mathbb{R}^{d_j}$ and $\sum_j d_j = d$, and let $U_j \in \mathbb{R}^{d \times d_j}$ be the matrix that places the entries of block j into their positions in $\mathbb{R}^d$ (its columns are the unit vectors e_i of the coordinates in the block), so that $x = \sum_j U_j x_{(j)}$. The **block gradient** is $\nabla_{(j)} f(x) = U_j^\top \nabla f(x) \in \mathbb{R}^{d_j}$, the vector of partial derivatives of block j , and the **block Lipschitz constant** L_j is defined by $\|\nabla_{(j)} f(x + U_j v) - \nabla_{(j)} f(x)\|_2 \leq L_j \|v\|_2$ for all x and $v \in \mathbb{R}^{d_j}$, which is Definition 12.3 with the axis e_i replaced by the subspace of block j . For a block of size one these reduce to $\nabla_i f$ and L_i .

Two moves are common. **Exact block minimisation** holds the other blocks fixed and solves the d_j -dimensional problem in the increment v ,

$$v^* \in \operatorname{argmin}_{v \in \mathbb{R}^{d_j}} f(x^k + U_j v), \quad x_{(j)}^{k+1} = x_{(j)}^k + v^*, \quad (12.8)$$

the block version of Definition 12.2. The **block gradient step** minimises the block upper model $f(x^k) + \nabla_{(j)} f(x^k)^\top v + \frac{L_j}{2} \|v\|_2^2$ instead,

$$x_{(j)}^{k+1} = x_{(j)}^k - \frac{1}{L_j} \nabla_{(j)} f(x^k),$$

all other blocks unchanged; each block has its own L_j and its own step $1/L_j$.

Note on the lecture slides

Frame 12.20 writes exact block minimisation as $x_{(j)}^{k+1} = \operatorname{argmin}_v f(x^k + U_j v)$. Since $x^k + U_j v$ adds v to block j , the minimiser v^* is an increment, exactly as u is in [Definition 12.2](#), and the formula as written sets the new block equal to the increment. For $f(x) = \frac{1}{2} \|x - \mathbf{1}\|_2^2$ with a single block and $x^k = \mathbf{1}$ already optimal it would return $x^{k+1} = 0$. The notes use (12.8).

The convergence logic survives blockwise: randomised block descent with uniformly chosen blocks inherits [Theorems 12.6](#) and [12.7](#) with J in place of d and the norm weighted by the L_j , and cyclic block sweeps inherit [Proposition 12.8](#). The rule of thumb is to update the smallest *meaningful* block, not the smallest possible one: a block that respects how decisions are made keeps the subproblem interpretable and lets it reuse shared data, at the price of a costlier update.

12.3.2 Proximal-coordinate descent

We now return the penalty to the lasso. Consider the composite objective of [Model 9.2](#) with a *separable* nonsmooth part,

$$F(x) = f(x) + \lambda \sum_{i=1}^d h_i(x_i),$$

where f is smooth with coordinate-wise constants L_i and each $h_i : \mathbb{R} \rightarrow \mathbb{R} \cup \{+\infty\}$ is convex: $h_i(t) = |t|$ gives the lasso, h_i the indicator of an interval gives a box constraint, and $\sum_i \lambda_i h_i(x_i)$ a weighted variant. Here λ is the penalty, as in [Week 9](#); [Section 12.4](#) will use the same letter for a price. The smooth part, g in [Model 9.2](#), is written f this week because g is reserved for the second ADMM term of [Section 12.5](#). [Week 9](#) handled F with a full gradient step followed by the prox of the whole penalty. The coordinate version uses the coordinate upper model (12.2) for the smooth part and keeps the scalar nonsmooth term exact: the new value t of coordinate i minimises

$$\nabla_i f(x^k)(t - x_i^k) + \frac{L_i}{2}(t - x_i^k)^2 + \lambda h_i(t) = \frac{L_i}{2} \left(t - \left(x_i^k - \frac{1}{L_i} \nabla_i f(x^k) \right) \right)^2 + \lambda h_i(t) + \text{const},$$

where the square was completed as in [Equation \(6.3\)](#). Dividing by L_i shows that this is the defining problem of the proximal operator of [Definition 9.6](#) with parameter λ/L_i , so the **proximal-coordinate step** is

$$x_i^{k+1} = \operatorname{prox}_{(\lambda/L_i)h_i} \left(x_i^k - \frac{1}{L_i} \nabla_i f(x^k) \right), \quad x_j^{k+1} = x_j^k \text{ for } j \neq i. \quad (12.9)$$

For $h_i(t) = |t|$ the prox is soft-thresholding ([Proposition 9.11](#)), $\operatorname{soft}_\tau(v) = \operatorname{sign}(v) \max\{|v| - \tau, 0\}$ with $\tau = \lambda/L_i$: the smooth coordinate step is taken first, then values within τ of zero are set *exactly* to zero and larger values are shrunk towards zero by τ . That is why the lasso produces sparse solutions, and why one of its coordinate updates costs one partial derivative and one scalar comparison. Proximal-coordinate descent is to [Week 9](#) what coordinate descent is to [Week 6](#): the same operator, one coordinate at a time. Because the prox handles the nonsmooth part exactly, the rates of [Section 12.2.4](#) carry over to F , $O(1/k)$ in general and linear under strong convexity, with modified constants in the convex case (stated without proof). For the lasso with $L_i = \|a_i\|_2^2$ the upper model of the smooth part is exact, so (12.9) is also exact coordinate minimisation of F , with the residual cached and refreshed as in (12.7).

Example 12.10 (Lasso by cyclic proximal-coordinate descent). Take the data of [Example 12.9](#) and $\lambda = 2$, so that both thresholds are $\lambda/L_i = 1$; start at $x^0 = 0$ with $r = -b$ and $F(x^0) = 7$. *Pass 1.* Coordinate 1: the smooth step gives $v_1 = 0 - (-3)/2 = 1.5$ and $x_1 = \operatorname{soft}_1(1.5) = 0.5$; the refresh gives $r \leftarrow (-0.5, -1.5, -3)$ and $F = 6.75$. Coordinate 2: $a_2^\top r = -4.5$, $v_2 = 2.25$, $x_2 = \operatorname{soft}_1(2.25) = 1.25$, $r \leftarrow (-0.5, -0.25, -1.75)$, $F = 5.1875$. *Pass 2.* Coordinate 1:

$a_1^\top r = -0.75$, $v_1 = 0.5 + 0.375 = 0.875$, and $\text{soft}_1(0.875) = 0$: the coefficient is set exactly to zero, $r \leftarrow (-1, -0.75, -1.75)$, $F = 4.8125$. Coordinate 2: $a_2^\top r = -2.5$, $v_2 = 2.5$, $x_2 = 1.5$, $r \leftarrow (-1, -0.5, -1.5)$, $F = 4.75$. A third pass changes nothing ($v_1 = 0.75 \mapsto 0$, $v_2 = 2.5 \mapsto 1.5$), so $x^* = (0, 1.5)$ with $F^* = 4.75$, reached exactly after two passes. The optimality condition of [Theorem 9.5](#) confirms it: $A^\top(Ax^* - b) = (-1.5, -2)$, with $|-1.5| \leq \lambda$ on the zero coordinate and $-2 = -\lambda \text{sign}(x_2^*)$ on the active one. Compared with the unpenalised solution $(1/3, 7/3)$, the penalty has removed the first feature entirely.

12.3.3 When coordinate and block methods are a good idea

The good signs are the ones the lasso displays: partial gradients are cheap, residuals can be cached, the variables are sparse or naturally grouped, and the nonsmooth term is separable. The warning signs are their opposites. When the coordinates are strongly coupled, a step along one axis is largely undone by the next (the constant μ_L of [Theorem 12.7](#) is small). When one block update is almost as expensive as a full step, nothing is gained per full-gradient equivalent. And when a hard shared constraint such as $\sum_i x_i = R$ ties the pieces together, no coordinate can move without the others compensating. This last sign is the subject of the rest of the week.

12.4 Coupling constraints and prices: dual decomposition

12.4.1 The budget-sharing model

Even when the objective is separable, a shared constraint can destroy independent local optimisation: in the running example m business units with local cost curves compete for one resource pool.

Model 12.11 (Budget sharing).

$$\begin{aligned} \min_{x \in \mathbb{R}^m} \quad & \sum_{i=1}^m f_i(x_i) \\ \text{s.t.} \quad & \sum_{i=1}^m x_i = R, \\ & x_i \geq 0, \quad i = 1, \dots, m. \end{aligned} \tag{12.10}$$

The decision variable $x_i \in \mathbb{R}$ is the amount of the resource allocated to unit i , and $f_i : \mathbb{R} \rightarrow \mathbb{R}$ is that unit's convex local cost (or negative benefit) as a function of its allocation; $R > 0$ is the total resource available (B is reserved for a constraint matrix in [Section 12.5](#)). Without the equality constraint the problem would fall apart into m scalar problems $\min_{x_i \geq 0} f_i(x_i)$ that each unit could solve alone, and would prefer to. The single row $\sum_i x_i = R$ couples them all: it is a coupling constraint ([Section 12.1](#)). A solver of [Week 4](#) handles a small instance directly; what follows is a method whose steps are local to the units and whose only communication is a single number.

12.4.2 A fixed price separates the problem

Attach a multiplier $\lambda \in \mathbb{R}$ to the coupling constraint, keeping $x \geq 0$ as a domain restriction rather than dualising it. The Lagrangian of [Definition 5.7](#) is

$$L(x, \lambda) = \sum_{i=1}^m f_i(x_i) + \lambda \left(\sum_{i=1}^m x_i - R \right) = \sum_{i=1}^m (f_i(x_i) + \lambda x_i) - \lambda R.$$

The letter L is the Lagrangian here, not a Lipschitz constant, and λ is the multiplier of an *equality* constraint, free in sign; [Week 5](#) wrote ν for such a multiplier, this week writes λ . The rearrangement is the whole idea: for a *fixed* λ the last term is a constant and the sum separates, so minimising $L(x, \lambda)$ over $x \geq 0$ is m independent scalar problems,

$$\min_{x_i \geq 0} f_i(x_i) + \lambda x_i, \quad i = 1, \dots, m. \tag{12.11}$$

Read economically, λ is a **price** per unit of the resource: unit i pays λx_i for what it uses and chooses its allocation to minimise local cost plus bill, knowing nothing about the other units. A fixed price turns one coupled problem into many independent local problems; what remains is to find the price at which the local choices add up to R .

12.4.3 The dual function and the price update

Definition 12.12 (Dual function of the budget model). The *dual function* of [Model 12.11](#) is

$$q(\lambda) = \inf_{x \geq 0} L(x, \lambda) = \sum_{i=1}^m \left(\inf_{x_i \geq 0} f_i(x_i) + \lambda x_i \right) - \lambda R, \quad \lambda \in \mathbb{R}, \quad (12.12)$$

and $x_i(\lambda)$ denotes a minimiser of the i -th local problem (12.11) at price λ . When the local problems have minimisers, as in all our examples, the infima are minima; in general $q(\lambda)$ may be $-\infty$.

This is the dual function $g(\lambda, \nu)$ of [Definition 5.8](#), renamed q because g is needed for the second objective term in [Section 12.5](#) (it is not the upper model q_i of [Section 12.2](#)). Its properties are those of [Week 5](#), with one addition that turns duality into an algorithm.

Proposition 12.13 (Properties of the dual function). Let p^* be the optimal value of [Model 12.11](#).

1. q is concave, whatever the functions f_i are.
2. Weak duality: $q(\lambda) \leq p^*$ for every $\lambda \in \mathbb{R}$.
3. If the f_i are convex and $R > 0$, then $\max_{\lambda} q(\lambda) = p^*$: maximising q gives the optimal value, and the maximiser λ^* is the shadow price of the resource ([Definition 5.21](#)). If in addition each local problem at λ^* has a unique solution (for example, each f_i is strictly convex), the $x_i(\lambda^*)$ form the optimal allocation.
4. If the f_i are convex and each local problem has a unique solution $x_i(\lambda)$, then q is differentiable with

$$q'(\lambda) = \sum_{i=1}^m x_i(\lambda) - R, \quad (12.13)$$

the aggregate over-use of the resource at price λ . If a local problem has several minimisers, the same bracket, for any choice among them, is a supergradient of the concave q : its negative is a subgradient of the convex $-q$ in the sense of [Definition 7.1](#).

Proof sketch. (i) q is a pointwise minimum of the affine functions $\lambda \mapsto L(x, \lambda)$, hence concave ([Proposition 3.19](#) applied to $-q$). (ii) is [Theorem 5.9](#). (iii) Since $R > 0$, the point $x_i = R/m > 0$ is strictly feasible, so Slater's condition ([Theorem 5.13](#)) gives $\max_{\lambda} q = p^*$; every optimal allocation x^* then satisfies $L(x^*, \lambda^*) = p^* = q(\lambda^*)$, so it minimises $L(\cdot, \lambda^*)$, and a unique local minimiser must be it. (iv) The local solutions $x(\lambda)$ are feasible for the minimisation defining $q(\lambda')$, so $q(\lambda') \leq L(x(\lambda), \lambda') = q(\lambda) + (\lambda' - \lambda)(\sum_i x_i(\lambda) - R)$, the supergradient inequality; for convex f_i every supergradient arises in this way, so a unique minimiser gives a unique slope, the derivative. $\square$

Since q is concave and its slope is computable from the local solutions, the price can be found by gradient ascent on q , that is [Algorithm 6.1](#) applied to $-q$:

$$\lambda^{k+1} = \lambda^k + \alpha \left(\sum_{i=1}^m x_i^{k+1} - R \right), \quad x_i^{k+1} = x_i(\lambda^k), \quad (12.14)$$

with a step size $\alpha > 0$. The economics and the calculus agree: when the units together want more than R the bracket is positive and the price rises, which lowers every local demand; when

Algorithm 12.2 Dual decomposition for the budget-sharing model

Require: strictly convex local costs $f_1, \dots, f_m$; resource R ; step size $\alpha > 0$; initial price λ^0 ; tolerance $\varepsilon > 0$

Ensure: a price λ and allocations $x_1, \dots, x_m$ with $|\sum_i x_i - R| \leq \varepsilon$

```

1: for  $k = 0, 1, 2, \dots$  do
2:   fix the price  $\lambda^k$ 
3:   for  $i = 1, \dots, m$  (independently, in parallel) do
4:      $x_i^{k+1} \leftarrow \operatorname{argmin}_{x_i \geq 0} f_i(x_i) + \lambda^k x_i$  ▷ local subproblem
5:   end for
6:    $\delta^k \leftarrow \sum_{i=1}^m x_i^{k+1} - R$  ▷ aggregate over-use =  $q'(\lambda^k)$ 
7:   if  $|\delta^k| \leq \varepsilon$  then return  $\lambda^k, x^{k+1}$ 
8:   end if
9:    $\lambda^{k+1} \leftarrow \lambda^k + \alpha \delta^k$  ▷ price update
10: end for

```

they want less, the price falls. Because of (12.13) the coordinator needs to know only the total $\sum_i x_i^{k+1}$, not the cost curves. Algorithm 12.2 is the resulting cycle: the multipliers of Week 5, proof devices there, have become *coordination variables*.

What the method does is solve m local problems per iteration and exchange one number; why it works is that the price update is gradient ascent on a concave function whose maximum equals p^* ; it applies when the objective is separable, the coupling is a shared linear constraint, and the local costs are strictly (ideally strongly) convex, so that each local problem has a unique solution. If each f_i is strongly convex with modulus μ_i , each $x_i(\lambda)$ is $(1/\mu_i)$ -Lipschitz in λ and q' is Lipschitz with constant $S = \sum_i 1/\mu_i$; by the descent lemma of Week 6 applied to $-q$ every price step with $0 < \alpha < 2/S$ increases q (and Week 6's rates hold for $\alpha \leq 1/S$); on a quadratic dual the price converges exactly when $\alpha < 2/S$, the stability condition of Example 6.14, and Example 12.14 shows that this limit is sharp. With merely convex costs (linear ones, say) a local problem can have several minimisers, q has kinks, the bracket is only a supergradient (iv), and the update behaves like the subgradient method of Algorithm 7.1, which needs diminishing steps; recovering a feasible allocation then needs care (iii). In every case the iterates are infeasible until convergence (hence the stopping test).

12.4.4 Worked example: five units, and the step-size limit

Example 12.14 (Five units sharing a budget). Let $m = 5$ units have quadratic costs $f_i(x_i) = \frac{c_i}{2}(x_i - t_i)^2$, where t_i is the allocation unit i would choose alone and c_i how steeply it suffers away from it, with $c = (1, 0.5, 2, 0.8, 1.5)$, $t = (3, 5, 2, 4, 2.5)$ and $R = 10$; the units want $\sum_i t_i = 16.5$ in total, so the resource is scarce. The local problem (12.11) has the stationarity condition $c_i(x_i - t_i) + \lambda = 0$, clipped at zero,

$$x_i(\lambda) = \max\{0, t_i - \lambda/c_i\},$$

a demand curve that falls with the price and switches off at $\lambda = c_i t_i$. While all units are active, $q'(\lambda) = \sum_i t_i - \lambda \sum_i 1/c_i - R$, so with $S = \sum_i 1/c_i = 5.4167$ the market clears at $\lambda^* = (16.5 - 10)/S = 1.2$, giving $x^* = (1.8, 2.6, 1.4, 2.5, 1.7)$, which sums to 10, and $p^* = q(\lambda^*) = 3.9$: zero duality gap, as (iii) promises, and a solver returns the same allocation with multiplier 1.2. The price is the marginal value of the budget: in the minimisation convention of Definition 5.21, $dp^*/dR = -\lambda^*$, so one more unit of R would lower the total cost by about 1.2, the most the units together should pay for it. Along the price update $q''(\lambda) = -S$ on the active set, so

$$\lambda^{k+1} - \lambda^* = (1 - \alpha S)(\lambda^k - \lambda^*),$$

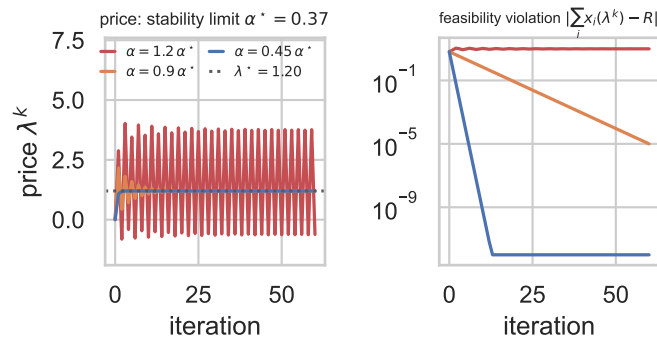


Figure 12.3: Dual decomposition on the five-unit budget model of Example 12.14: the price λ^k (left) and the constraint violation $|\sum_i x_i(\lambda^k) - R|$ (right, log axis) for $\alpha = 1.2, 0.9$ and 0.45 times $\alpha^* = 0.37$.

and the price converges if and only if $|1 - \alpha S| < 1$, that is $0 < \alpha < \alpha^* = 2/S = 0.369$: the condition $\alpha < 2/L$ of Week 6 seen through the dual, with $L = S$ the curvature of $-q$. From $\lambda^0 = 0$: with $\alpha = 0.45\alpha^*$ the factor is 0.1 and the price runs 0, 1.08, 1.188, 1.199; with $\alpha = 0.9\alpha^*$ the factor is -0.8 and the price overshoots on alternate steps (0, 2.16, 0.432, 1.814, ...) before converging slowly; with $\alpha = 1.2\alpha^*$ the linearised factor is -1.4 and the price (0, 2.88, -0.815 , 4.021, ...) never settles (Figure 12.3); negative prices are allowed for an equality multiplier. Nor does it blow up: above $c_i t_i \in \{2.5, 3, 3.2, 3.75, 4\}$ units switch off and the curvature shrinks, and the run ends in a bounded oscillation between prices near -0.62 and 3.75 , with the constraint violated by about 9.9 units.

Just above the limit the price oscillates forever; just below it, it converges. The lesson is a limitation: the limit $2/S$ depends on the curvature of the dual function, which the coordinator does not know and which changes as units enter or leave the active set. The augmented Lagrangian of the next subsection is a correction that does not need α tuned so finely.

Key idea

A fixed price separates a problem coupled by a shared constraint into independent local problems, and the market-clearing price is found by gradient ascent on the concave dual function, whose slope is the aggregate over-use. The multiplier of Week 5 has become a coordination variable, and the price update inherits a gradient method's step-size limit.

12.5 The augmented Lagrangian and ADMM

12.5.1 The augmented Lagrangian: stabilise, but re-couple

Write the coupling constraint in the general form $Ax = b$; for the budget model it is the single row $\mathbf{1}^\top x = R$. In this subsection A and b are constraint data, not the data matrix and response of the lasso; the split lasso below has both meanings in one formula and says which is which. The remedy for the fragile price update is to charge for infeasibility directly.

Definition 12.15 (Augmented Lagrangian). For $\min f(x)$ subject to $Ax = b$, with x ranging over a convex domain such as $x \geq 0$, and a **penalty parameter** $\rho > 0$ (not the contraction factor ρ of Theorem 11.13), the **augmented Lagrangian** is

$$\mathcal{L}_\rho(x, \lambda) = f(x) + \lambda^\top (Ax - b) + \frac{\rho}{2} \|Ax - b\|_2^2. \quad (12.15)$$

For $\rho = 0$ this is the ordinary Lagrangian; λ still plays the role of a price, and the quadratic term charges directly for the violation $Ax - b$, whatever its sign. The square vanishes on the

Algorithm 12.3 ADMM in scaled form for $\min f(x) + g(z)$ subject to $Ax + Bz = c$

Require: f, g convex; A, B, c ; penalty $\rho > 0$; starting z^0, u^0 ; tolerances $\varepsilon_{\text{pri}}, \varepsilon_{\text{dual}} > 0$
Ensure: an approximate minimiser (x, z) with $Ax + Bz \approx c$

```

1: for  $k = 0, 1, 2, \dots$  do
2:    $x^{k+1} \leftarrow \operatorname{argmin}_x \{f(x) + \frac{\rho}{2} \|Ax + Bz^k - c + u^k\|_2^2\}$  ▷  $x$ -update
3:    $z^{k+1} \leftarrow \operatorname{argmin}_z \{g(z) + \frac{\rho}{2} \|Ax^{k+1} + Bz - c + u^k\|_2^2\}$  ▷  $z$ -update, with the new  $x$ 
4:    $u^{k+1} \leftarrow u^k + Ax^{k+1} + Bz^{k+1} - c$  ▷ price update, step  $\rho$  built in
5:    $r^{k+1} \leftarrow Ax^{k+1} + Bz^{k+1} - c, \quad s^{k+1} \leftarrow \rho A^\top B(z^{k+1} - z^k)$  ▷ residuals
6:   if  $\|r^{k+1}\|_2 \leq \varepsilon_{\text{pri}}$  and  $\|s^{k+1}\|_2 \leq \varepsilon_{\text{dual}}$  then return  $(x^{k+1}, z^{k+1})$ 
7:   end if
8: end for

```

feasible set, so the minimisers and optimal multipliers are unchanged. What changes is the dual function: for convex f , the dual function $\lambda \mapsto \min_x \mathcal{L}_\rho(x, \lambda)$ has the same maximisers as before but a $(1/\rho)$ -Lipschitz gradient (stated without proof), so ρ is exactly the safe step $1/L$ of [Week 6](#) for it, and the multiplier can be updated by $\lambda^{k+1} = \lambda^k + \rho(Ax^{k+1} - b)$ without tuning a step to an unknown curvature. The stabilisation has a cost, and it is exactly the property that made decomposition work: $\|Ax - b\|_2^2$ *does not separate*. For the budget model $(\sum_i x_i - R)^2$ contains every cross term $x_i x_j$, so the x -minimisation no longer splits into local problems. The augmented term stabilises but re-couples; recovering separability is the job of splitting, and that is why ADMM exists.

12.5.2 Splitting and the scaled-form updates

Model 12.16 (ADMM standard split form).

$$\begin{aligned} \min_{x, z} \quad & f(x) + g(z) \\ \text{s.t.} \quad & Ax + Bz = c, \end{aligned} \tag{12.16}$$

with f and g convex and A, B, c a conformable pair of matrices and a vector (c is unrelated to the cost slopes of [Example 12.14](#)).

The objective is a sum of two terms, each depending on its own block of variables, and the blocks interact only through a linear constraint. Splitting is a modelling move: one introduces a second variable z so that one part is smooth and the other nonsmooth, or so that one block has an easy linear solve while the other has an easy proximal step; a difficult model can become easy after splitting. The lasso is the standard case, written $\min \frac{1}{2} \|Ax - b\|_2^2 + \lambda \|z\|_1$ subject to $x - z = 0$: the two copies must agree, and each term now sees only its own copy.

ADMM, the alternating direction method of multipliers, applies the augmented-Lagrangian idea to (12.16) but replaces the joint minimisation over (x, z) by one sweep of block coordinate minimisation, x then z . A change of variable makes the updates tidy. With $r = Ax + Bz - c$ and the **scaled dual variable** $u = \lambda/\rho$, completing the square gives

$$\lambda^\top r + \frac{\rho}{2} \|r\|_2^2 = \frac{\rho}{2} \|r + u\|_2^2 - \frac{\rho}{2} \|u\|_2^2, \tag{12.17}$$

as expanding the right-hand side confirms. The last term of (12.17) involves neither x nor z , so minimising $\mathcal{L}_\rho$ over either block means minimising f or g plus $\frac{\rho}{2} \|Ax + Bz - c + u\|_2^2$. Here r is the constraint residual, not the least-squares residual of [Section 12.2](#), and u is the scaled price, not a coordinate increment. [Algorithm 12.3](#) collects the cycle.

Three remarks read the updates. First, u is still the price, in units of ρ : multiplying the third line by ρ gives $\lambda^{k+1} = \lambda^k + \rho(Ax^{k+1} + Bz^{k+1} - c)$, the price update (12.14) with the step

ρ built in rather than tuned. Second, when $B = -I$ and $c = 0$ the constraint reads $z = Ax$ and the z -update is

$$z^{k+1} = \underset{z}{\operatorname{argmin}} \left\{ g(z) + \frac{\rho}{2} \|z - (Ax^{k+1} + u^k)\|_2^2 \right\} = \operatorname{prox}_{g/\rho}(Ax^{k+1} + u^k), \quad (12.18)$$

the proximal operator of Definition 9.6 with parameter $1/\rho$, unchanged. If g is an indicator the prox is a projection (Proposition 9.8), which is how Week 13 will use ADMM for constraints without a cheap projection. Third, the first two steps are one sweep of block coordinate minimisation (Section 12.3) and the third corrects the violation: ADMM alternates two easy block subproblems, then updates the price.

12.5.3 What to expect: convergence and residuals

Termination uses two residuals rather than objective values. The **primal residual** $r^k = Ax^k + Bz^k - c$ measures the violation of the coupling constraint. The **dual residual** $s^k = \rho A^\top B(z^k - z^{k-1})$ (unrelated to the mini-batch size s of Week 11) measures how far the x -block is from stationarity: the optimality condition of the x -update, combined with $\lambda^k = \lambda^{k-1} + \rho r^k$, gives $s^k \in \partial f(x^k) + A^\top \lambda^k$, the stationarity condition of Definition 5.14 up to the error s^k . Both residuals vanish exactly at a KKT point, which is why they, and not the objective, are the stopping test.

Theorem 12.17 (Convergence of two-block ADMM). *Let f and g be closed, proper and convex (they may take the value $+\infty$, as an indicator does, but are finite somewhere and have closed epigraphs), and suppose the ordinary Lagrangian of (12.16) has a saddle point, that is, a primal solution and a multiplier λ^* with zero duality gap. Then for every $\rho > 0$ the iterates of Algorithm 12.3 satisfy $f(x^k) + g(z^k) \rightarrow p^*$, and the primal residual $r^k = Ax^k + Bz^k - c$ and dual residual $s^k = \rho A^\top B(z^k - z^{k-1})$ satisfy $r^k \rightarrow 0$ and $s^k \rightarrow 0$. For the averaged iterates $\bar{x}^k = \frac{1}{k} \sum_{j=1}^k x^j$ and $\bar{z}^k$ the objective gap and the constraint violation are $O(1/k)$.*

Stated without proof, the theorem is the central result of the subsection: *global convergence* in the two-block convex setting, with no step-size tuning beyond ρ . The averaged iterate is the device of Theorem 7.18 and Theorem 11.18; in practice the residuals matter more than the constant. Unlike projected gradient (Algorithm 8.2), ADMM satisfies the coupling constraint only in the limit: a run stopped early violates $Ax + Bz = c$ by r^k .

12.5.4 Two applications: the split lasso and consensus

Model 12.18 (Split lasso).

$$\begin{aligned} \min_{x, z \in \mathbb{R}^d} \quad & \frac{1}{2} \|Ax - b\|_2^2 + \lambda \|z\|_1 \\ \text{s.t.} \quad & x - z = 0. \end{aligned} \quad (12.19)$$

Here $A \in \mathbb{R}^{N \times d}$ and b are the lasso's data, $\lambda > 0$ is again the lasso penalty, and the template (12.16) has constraint matrices I and $-I$ with right-hand side 0; the price is carried by u , the multiplier is ρu , and neither appears by name. The x -update minimises $\frac{1}{2} \|Ax - b\|_2^2 + \frac{\rho}{2} \|x - z^k + u^k\|_2^2$, a strongly convex quadratic; setting its gradient $A^\top(Ax - b) + \rho(x - z^k + u^k)$ to zero (Proposition 1.17) gives a linear system. The z -update is (12.18) with $g = \lambda \|\cdot\|_1$, whose prox is soft-thresholding (Proposition 9.11). Altogether,

$$x^{k+1} = (A^\top A + \rho I)^{-1} (A^\top b + \rho(z^k - u^k)), \quad z^{k+1} = \operatorname{soft}_{\lambda/\rho}(x^{k+1} + u^k), \quad u^{k+1} = u^k + x^{k+1} - z^{k+1}, \quad (12.20)$$

with residuals $r^k = x^k - z^k$ and $s^k = -\rho(z^k - z^{k-1})$. The matrix $A^\top A + \rho I$ is fixed, so it is factorised once and every later x -update is a cheap back-substitution. Splitting has turned one composite problem into a linear solve plus a soft-thresholding step, and half of ADMM on the lasso is the operator of Week 9.

Example 12.19 (A scalar split lasso, by hand). Take $d = 1$, $A = 1$, $b = 3$ and $\lambda = \rho = 1$, so that the lasso solution is $\text{soft}_1(3) = 2$, and start from $z^0 = u^0 = 0$. Iteration 1: $x^1 = (3 + 0)/2 = 1.5$, $z^1 = \text{soft}_1(1.5) = 0.5$, $u^1 = 0 + 1.5 - 0.5 = 1$, with $r^1 = 1$, $s^1 = -0.5$. Iteration 2: $x^2 = (3 + 0.5 - 1)/2 = 1.25$, $z^2 = \text{soft}_1(2.25) = 1.25$, $u^2 = 1$, $r^2 = 0$. From then on $u^k = 1$ and $x^k = z^k$ runs 1.625, 1.8125, 1.906, $\dots$, converging geometrically to 2. In the limit $\rho u = 1 = \lambda \text{sign}(z^*)$, as the optimality condition of [Theorem 9.5](#) requires. The scaled price equals the threshold λ/ρ only because $z^* \neq 0$: with $b = 0.2$, $z^* = 0$ and $u^* = 0.2$ while $\lambda/\rho = 1$, so the two are different objects.

The second application is **consensus**. Suppose m regions each hold their own data and their own loss f_i , and must agree on one model:

$$\begin{aligned} \min \quad & \sum_{i=1}^m f_i(x_i) \\ \text{s.t.} \quad & x_i - z = 0, \quad i = 1, \dots, m, \end{aligned}$$

where now $x_i \in \mathbb{R}^d$ is region i 's own copy of the model and $z \in \mathbb{R}^d$ the global consensus vector. This is (12.16) with $x = (x_1, \dots, x_m)$, $g = 0$, and a constraint that stacks the m copies against z , so it is two-block. Because f is separable across regions and the constraint couples each x_i only to z , the x -update splits into m independent problems, one per region, and the z -update, which minimises $\sum_i \|x_i^{k+1} - z + u_i^k\|_2^2$, is an average:

$$\begin{aligned} x_i^{k+1} &= \underset{x_i}{\operatorname{argmin}} \left\{ f_i(x_i) + \frac{\rho}{2} \|x_i - z^k + u_i^k\|_2^2 \right\}, \quad i = 1, \dots, m, \\ z^{k+1} &= \frac{1}{m} \sum_{i=1}^m (x_i^{k+1} + u_i^k), \quad u_i^{k+1} = u_i^k + x_i^{k+1} - z^{k+1}. \end{aligned}$$

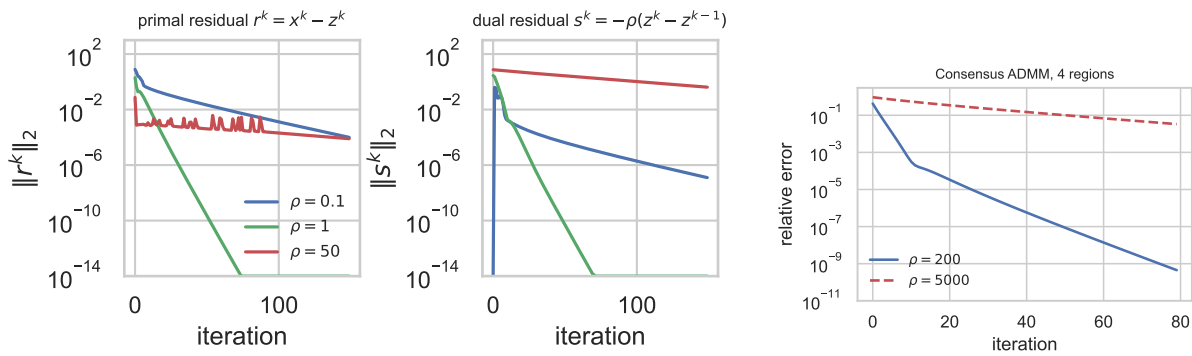
For regional least squares, $f_i(x_i) = \frac{1}{2} \|A_i x_i - b_i\|_2^2$, the local update is the split-lasso linear solve with region i 's own rows (A_i, b_i) . Each region shares only its copy x_i and its price u_i , never its rows, and z converges to the solution of the *pooled* problem $\min_z \sum_i f_i(z)$. [Figure 12.4b](#) shows a run with $m = 4$ regions of 1000 rows each and $d = 20$, in which every region's own least-squares fit is 1–1.6% away from the pooled solution: with $\rho = 200$ the relative error of z^k is 4.5×10^{-10} after 80 iterations, nine digits; with $\rho = 5000$ it is still 3.3×10^{-2} .

Note on the lecture slides

Frame 12.35 says the consensus run reaches the pooled solution to nine digits “provided ρ is in the right range”, which reads as a condition for convergence. Consensus ADMM is two-block, so [Theorem 12.17](#) applies for every $\rho > 0$; ρ governs only the speed within the 80 plotted iterations, and the $\rho = 5000$ run reaches 1.4×10^{-8} by iteration 500 and 10^{-15} by iteration 2000.

12.5.5 Many blocks: why the naive extension is unsafe

For the coupled template $\min \sum_{i=1}^m f_i(x_i)$ subject to $\sum_{i=1}^m A_i x_i = b$ with $m \geq 3$ blocks it is tempting to extend the two-block scheme by updating $x_1, \dots, x_m$ one after another, each with the latest values, and then u . That loop is *not* automatically convergent: there are convex three-block instances, even linear systems, on which it diverges from generic starting points for every $\rho > 0$, and the proof behind [Theorem 12.17](#) does not extend. Safer designs regroup the blocks into a genuine two-block splitting, use a consensus form (every x_i coupled only to z , two-block by construction) or add proximal terms to each block subproblem; randomising the block order also helps.



(a) Split lasso ($N = 400$, $d = 100$): $\|r^k\|_2$ and $\|s^k\|_2$ for $\rho \in \{0.1, 1, 50\}$. (b) Consensus least squares, 4 regions: relative error of z^k to the pooled solution.

Figure 12.4: Measured ADMM runs (log axes). Left: too small a ρ leaves the primal residual behind, too large a ρ the dual residual. Right: a poorly chosen ρ is slow, not wrong.

12.5.6 Reading the residuals: ρ as a tuning knob

Figure 12.4a applies (12.20) to a lasso instance with $N = 400$, $d = 100$, unit-norm columns, 10 true nonzeros and $\lambda = 0.1 \|A^\top b\|_\infty = 0.537$, for $\rho \in \{0.1, 1, 50\}$. All three runs head for the same solution, with the same 9-element support, but the residuals fall differently. After 150 iterations, $\rho = 0.1$ has $\|r^k\|_2 = 1.0 \times 10^{-4}$ but $\|s^k\|_2 = 1.2 \times 10^{-7}$: a small ρ under-enforces the coupling and leaves the primal residual behind. With $\rho = 50$, $\|r^k\|_2 = 8 \times 10^{-5}$ but $\|s^k\|_2 = 0.41$, and the objective is 11.748 against the optimal value 11.657: a large ρ over-enforces the coupling, so the copies agree while z is still moving, and the dual residual lags. With $\rho = 1$ both residuals are near 10^{-16} , and the objective agrees with an RSOME solve of Model 12.18 (11.657398 against 11.657401) within the solver's tolerance. The residuals are thus a tuning instrument, not only a stopping test: a healthy run drives both down together, and if they stall or separate, adjust ρ (up when r^k lags, down when s^k lags) or rethink the splitting rather than iterate longer.

Key idea

Two-block ADMM alternates two easy block minimisations of the augmented Lagrangian with a price update whose step ρ is built in; it converges for every $\rho > 0$, and its primal and dual residuals are both the stopping test and the guide to choosing ρ .

12.6 Summary and looking ahead

The week's methods answer one question, what is the cheapest meaningful subproblem at each step, for three kinds of structure; Table 12.1 lists the question to ask before choosing. Coordinate and block methods exploit variable structure: each step uses the coordinate upper model in place of the global one of Week 6, and pays off when a partial gradient is much cheaper than a full one, but not when the coordinates are strongly coupled or a shared constraint ties them together. Proximal-coordinate descent is the coordinate-level extension of Week 9: the same prox, one scalar at a time. Dual decomposition uses multipliers as coordination prices, and inherits a gradient method's step-size limit. The augmented Lagrangian removes the tuning but re-couples the variables; two-block ADMM restores separability by splitting and is the standard safe convex case, while naive many-block ADMM needs caution. The durable lesson is not one formula but a modelling habit: exploit separability whenever possible, and always respect the assumptions behind the convergence story.

Week 13 drops the premise, held since Week 8, that projection is cheap. Three escapes

remain: a linear oracle (the Frank–Wolfe method); moving the constraints into the objective, this week’s ADMM with a z -step that is a projection; and the log barrier of interior-point solvers. Their certificates, the residuals of [Theorem 12.17](#) among them, then close the unit in one map of methods.

12.7 Exercises

Exercise 12.1 (Structure recognition). For each model choose the most natural first method to try, and justify the choice: (a) the lasso with millions of features; (b) budget sharing with one common resource cap; (c) least squares with a dense matrix of moderate size; (d) the split lasso with the agreement constraint $x - z = 0$.

Exercise 12.2 (Coordinate update for least squares). (a) Starting from $f(x) = \frac{1}{2} \|Ax - b\|_2^2$, derive the coordinate update for coordinate i and the residual update formula. (b) *Extension*: state the cost of one update when the residual is cached.

Exercise 12.3 (Assumptions and rates). State and interpret the assumptions behind the randomised and cyclic coordinate-descent guarantees. What is coordinate-wise Lipschitz smoothness? Why is the existence of a minimiser needed? What changes when strong convexity is added? Why is performance compared by epochs rather than by iterations? Then explain the practical meaning of $\mathbb{E}[f(x^k) - f^*] = O(1/k)$ and of $\mathbb{E}[f(x^k) - f^*] \leq q^k C$ ($q \in (0, 1)$ a contraction factor, not the dual function).

Exercise 12.4 (Proximal-coordinate lasso update). For $F(x) = \frac{1}{2} \|Ax - b\|_2^2 + \lambda \|x\|_1$: (a) write the coordinate upper model for coordinate i ; (b) derive the proximal-coordinate step $x_i^{k+1} = \text{soft}_{\lambda/L_i}(x_i^k - \frac{1}{L_i} \nabla_i f(x^k))$; (c) compute one numerical update when $L_i = 5$, $\lambda = 0.8$ and $x_i^k - \nabla_i f(x^k)/L_i = 0.10$.

Exercise 12.5 (Dual decomposition of the budget-sharing model). For [Model 12.11](#): (a) form the Lagrangian; (b) show explicitly how the minimisation separates for a fixed multiplier λ ; (c) interpret λ as a common price; (d) write one dual-decomposition iteration in words. (e) *Extension*: for the five-unit data of [Example 12.14](#), compute the stability limit α^* when every c_i is multiplied by 10.

Exercise 12.6 (One ADMM iteration for the split lasso). Starting from [Model 12.18](#): (a) write the x -update; (b) write the z -update and identify the soft-thresholding step; (c) write the dual update for u ; (d) define the primal and dual residuals for this split.

Method	Structure to exploit	Good first question to ask
Full gradient descent (Week 6)	one full gradient is affordable	Can I update the whole vector reliably using global smoothness?
Stochastic gradient (Week 11)	too many samples	Can I replace the full gradient by a cheap unbiased estimate?
Coordinate descent	cheap scalar updates, sparse structure	Is one coordinate update dramatically cheaper than a full gradient?
Block coordinate descent	natural grouped variables	Is there a meaningful block partition that respects the application?
Proximal-coordinate descent	smooth + separable nonsmooth	Does the penalty split into cheap scalar proximal steps?
Dual decomposition	separable objective + shared resource	Can a price variable separate the local decisions?
ADMM	split model with two easy blocks	Can I reformulate so that each block step is simple and the coupling is only linear?

Table 12.1: Method choice by structure: the question to ask before choosing a method.

Exercise 12.7 (Many-block ADMM). Consider $\min \sum_{i=1}^4 f_i(x_i)$ subject to $\sum_{i=1}^4 A_i x_i = b$. (a) Why is the naive update order x_1, x_2, x_3, x_4 , then u , not automatically safe? (b) Propose one safer redesign: regroup into two blocks, add proximal regularisation, randomise the block order, or use another corrected scheme. (c) Explain the practical trade-off of your redesign.

Exercise 12.8 (Coding: coordinate descent and ADMM against RSOME). Implement cyclic or randomised coordinate descent for a small lasso instance (objective against epoch, sparsity pattern of the final estimate), and ADMM for the split lasso or a simple sharing or consensus model (objective, r^k, s^k , sensitivity to ρ). Compare both with an RSOME solve of the same small instance, explain any difference in runtime or diagnostics, and note when coordinate descent appears attractive in practice.

Week 13 Optimisation with Complex Constraints, and the Course in One Map

What if the feasible set is too hard to project onto? [Week 8](#) made gradient descent a constrained method by projecting onto C , and [Weeks 9 to 11](#) built on it, always under one premise: the set is *simple*, its projection closed-form or very cheap. This week removes the premise. The feasible sets that real mandates produce are intersections of simple sets, which in general have no closed-form projector; on a general polyhedron one projection is itself a quadratic programme.

What does not change is the target, the optimality condition $0 \in \nabla f(x^*) + N_C(x^*)$ of [Theorem 8.7](#); what changes is the primitive that chases it. [Section 13.1](#) states the bottleneck and measures it. [Sections 13.2 to 13.4](#) develop three escapes: ask C a linear question instead of a quadratic one (the Frank–Wolfe method), move the constraints into the objective (the ADMM of [Week 12](#)), and never reach the boundary (a log barrier, the Newton’s method of [Week 6](#) and the weak duality of [Week 5](#), which together are the idea behind interior-point solvers). [Section 13.5](#) compares them on one instance and one clock, and then answers the unit’s recurring question, *what is cheap to compute here, and what does it certify?*, for the whole course.

Some notation is reused with a new meaning. $A \in \mathbb{R}^{m \times d}$ and $b \in \mathbb{R}^m$ are *constraint* data, $C = \{x : Ax \leq b\}$ with rows $a_i^\top$ (in [Weeks 6](#) and [9 to 11](#) they were least-squares data; in [Week 12](#) a_i was a *column* of the data matrix and m counted units or blocks, whereas here $a_i^\top$ is again a row and m again counts inequality constraints). The letter t is the barrier parameter, not [Week 6](#)’s step size t_k ; the Frank–Wolfe step is γ_k , g_k is a scalar gap, not a subgradient, and s^k is the oracle’s vertex, not [Week 12](#)’s dual residual. The vector c holds sector caps, the linear coefficients of the running QP and the cost of a linear programme; μ is the mean-return vector and, in rates, the strong-convexity constant; p^* ([Week 5](#)’s notation) is f^* .

13.1 The bottleneck: when projection stops being cheap

13.1.1 One assumption behind projected gradient

Recall the projected-gradient method ([Algorithm 8.2](#)): for a convex differentiable f and a nonempty closed convex C ,

$$x^{k+1} = \Pi_C(x^k - \alpha \nabla f(x^k)), \quad \Pi_C(y) = \operatorname{argmin}_{z \in C} \frac{1}{2} \|z - y\|_2^2, \quad (13.1)$$

a gradient step followed by a return to the nearest feasible point. The method inherits the guarantees of gradient descent (the factor $1 - \mu/L$ of [Theorem 8.18](#), now for the distance to x^* , and the $O(1/k)$ of [Theorem 9.18](#) with $h = I_C$) only because the projection is counted as free, as it is for a box or the orthant (a clip), a ball (a rescaling) and the simplex (a sort; [Table 8.1](#) and [Proposition 8.13](#)). The prox of [Week 9](#), the acceleration of [Week 10](#) and the sampled gradients of [Week 11](#) rest on the same cheap primitive.

Key idea

A method is only as cheap as its most expensive primitive. If Π_C stops being cheap, the whole lane of [Weeks 8 to 11](#) stops with it: the optimality condition survives, the cost accounting does not.

13.1.2 Feasible sets that real mandates produce

The simple sets of [Week 8](#) are where business models start, not where they end. Take the long-only portfolio of [Model 1.2](#) once an investment committee has written its mandate.

Model 13.1 (Portfolio with a mandate). Let f be the mean–variance objective of [Model 1.2](#), written as a minimisation. Given a sector-exposure matrix S (row j marks the assets in sector j), sector caps c , the current portfolio w^{old} and a turnover limit $\tau > 0$,

$$\begin{aligned} \min_{w \in \mathbb{R}^d} \quad & f(w) \\ \text{s.t.} \quad & w \geq 0, \quad \mathbf{1}^\top w = 1, \quad Sw \leq c, \quad \|w - w^{\text{old}}\|_1 \leq \tau. \end{aligned}$$

The first two constraints are the simplex of [Definition 2.12](#); the sector caps are halfspaces; the turnover limit is an ℓ_1 ball centred at the current portfolio, the hard version of the penalty of [Model 9.4](#). The intersection is convex ([Proposition 2.17](#)), and every piece *on its own* has a cheap projection: a sort for the simplex, a shift along the normal for each sector cap (sectors being disjoint), a sort and a soft-threshold for the turnover ball. The intersection has none, so no method of [Weeks 8 to 11](#) applies as it stands. The three escapes below are three ways to solve the model, and since its solution is an allocation a committee will trade, whether the iterates stay feasible during the run is part of the choice. The same happens to the media-mix model of [Model 1.1](#) once channel caps, minimum spends and ratio rules are added, and to a transportation model ([Section 4.2](#)) with lane limits and blending requirements. Caps, floors, turnover limits and blending ratios are the rules a business actually imposes: complex constraints are the normal case, and the simple sets of [Week 8](#) were the special case.

13.1.3 What one projection costs

To measure the bottleneck we use, throughout the week, the general polyhedron ([Definition 2.11](#))

$$C = \{x \in \mathbb{R}^d : Ax \leq b\}, \quad A \in \mathbb{R}^{m \times d}, \quad b \in \mathbb{R}^m, \quad (13.2)$$

with $m = 2d$ rows drawn at random and b chosen so that 0 is strictly feasible. Projecting a point y onto C means solving

$$\min_{x \in \mathbb{R}^d} \|x - y\|_2^2 \quad \text{s.t.} \quad Ax \leq b, \quad (13.3)$$

a convex quadratic programme with no closed form: a solver call inside every iteration. [Figure 13.1](#) reports the median time of one such solve, made with RSOME and MOSEK in the companion notebook `week13_constraints.ipynb`. The simplex projection of [Proposition 8.13](#) stays below 0.03 ms up to $d = 800$; the polyhedral projection costs about 13 ms at $d = 100$ and half a second at $d = 800$, a factor of roughly 15 000, paid every iteration (times depend on the machine; the ratio is the point). The subproblem is now as hard as the problem. The third curve, minimising a *linear* function over the same polyhedron (intersected with the box $\|x\|_\infty \leq 50$ so that the linear programme is bounded), is about as expensive as the projection; we return to it in [Section 13.2.7](#).

13.1.4 Three ways out, one certificate

Whatever we do about the cost, the target does not move. For convex differentiable f and nonempty closed convex C , x^* minimises f over C if and only if

$$0 \in \nabla f(x^*) + N_C(x^*) \quad \Longleftrightarrow \quad \nabla f(x^*)^\top (y - x^*) \geq 0 \quad \text{for all } y \in C, \quad (13.4)$$

the normal-cone condition and its variational-inequality form ([Theorem 8.7](#) and [Equation \(8.6\)](#)): no feasible direction points downhill. Complexity of C changes neither statement, only which computation we can afford in pursuit of it: a linear question whose answer is a corner (escape 1), a price on the constraints, which splitting turns into a clip (escape 2), or a barrier finite inside C and infinite at its edge (escape 3). All three certify [\(13.4\)](#).

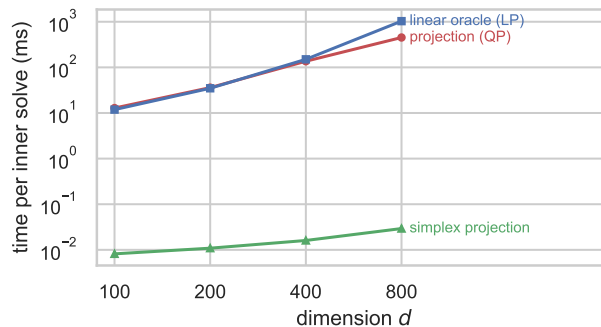


Figure 13.1: Median wall-clock time of one inner solve as d grows (log-log scales, $m = 2d$ constraints): projection onto the general polyhedron (13.2) (a QP), linear minimisation over the same polyhedron intersected with a large box (an LP), and projection onto the simplex (a sort). At $d = 800$ the polyhedral projection costs about 15 000 simplex projections, and the LP is about as expensive as the QP.

13.2 Escape 1: a linear oracle and the Frank–Wolfe method

13.2.1 Asking C a linear question

Projection, in (13.1), asks C a *quadratic* question: which feasible point is nearest to $x^k - \alpha \nabla f(x^k)$? The first escape asks a *linear* one instead: which feasible point makes $\nabla f(x^k)^\top s$ smallest? This minimises the first-order model $f(x^k) + \nabla f(x^k)^\top (s - x^k)$ of Equation (1.4), the model gradient descent stepped along in Week 6, over the whole of C . For the answer to exist we assume from now on that C is *compact* as well as convex; a linear function on an unbounded set may have no minimiser.

Definition 13.2 (Linear minimisation oracle; vertex). Let $C \subset \mathbb{R}^d$ be nonempty, compact and convex. A **linear minimisation oracle** (LMO) for C is a routine that, given a direction $v \in \mathbb{R}^d$, returns a point

$$s \in \operatorname{argmin}_{z \in C} v^\top z. \quad (13.5)$$

The Frank–Wolfe method calls it with $v = \nabla f(x^k)$ and writes the answer s^k . A point $\bar{s} \in C$ is a **vertex** (or corner, or extreme point) of C if it is not a convex combination $\theta y + (1 - \theta)z$, $0 < \theta < 1$, of two points $y \neq z$ of C . A compact polyhedron has finitely many vertices $\bar{s}_1, \dots, \bar{s}_n$ and is their convex hull (Definition 2.4), so a linear objective over it is always minimised at one of them: if $s = \sum_i \theta_i \bar{s}_i$ with $\theta_i \geq 0$, $\sum_i \theta_i = 1$, then $v^\top s = \sum_i \theta_i v^\top \bar{s}_i \geq \min_i v^\top \bar{s}_i$.

So over a polyhedron the oracle (13.5) is a linear programme whose answer can be taken to be a corner, as in the picture of Example 1.6, where the last contour of $2x_1 + 3x_2$ to touch the pentagon touched it at the corner $(2, 3)$, and over the sets that matter most in this course it is a single **argmin** or **argmax**. Write e_j for the j -th standard basis vector (on the simplex, all weight on asset j). On the simplex $\{s \geq 0, \mathbf{1}^\top s = 1\}$, $v^\top s = \sum_j s_j v_j$ is a weighted average of the entries of v , so its minimum is the smallest entry: $s = e_j$ with $j \in \operatorname{argmin}_j v_j$, at a cost of $O(d)$. On the ℓ_1 ball $\{\|s\|_1 \leq \tau\}$, whose vertices are the $2d$ points $\pm \tau e_j$, the minimum $-\tau \|v\|_\infty$ is attained at $s = -\tau \operatorname{sign}(v_j) e_j$ with $j \in \operatorname{argmax}_j |v_j|$: one **argmax** with a sign. On the Euclidean ball $\{\|s\|_2 \leq r\}$, whose vertices in the sense just defined are all the points of its boundary sphere (it has no corners in the polyhedral sense), the Cauchy–Schwarz inequality (Proposition 1.11) gives $s = -r v / \|v\|_2$ for $v \neq 0$, again a vertex. In every case the oracle asks C only *which way is downhill*; no nearest-point problem is ever solved.

Algorithm 13.1 Frank–Wolfe (conditional gradient)

Require: convex differentiable f ; a linear minimisation oracle for the compact convex set C ; a feasible start $x^0 \in C$; tolerance $\varepsilon > 0$

Ensure: a feasible x^k with $f(x^k) - f^* \leq g_k \leq \varepsilon$

```

1: for  $k = 0, 1, 2, \dots$  do
2:    $s^k \leftarrow$  a point of  $\operatorname{argmin}_{s \in C} \nabla f(x^k)^\top s$   $\triangleright$  the linear oracle; a vertex
3:    $g_k \leftarrow \nabla f(x^k)^\top (x^k - s^k)$   $\triangleright$  the gap: free, see Proposition 13.5
4:   if  $g_k \leq \varepsilon$  then return  $x^k$ 
5:    $\gamma_k \leftarrow 2/(k+2)$ ;  $x^{k+1} \leftarrow (1 - \gamma_k)x^k + \gamma_k s^k$   $\triangleright$  stays in  $C$ 
6: end for
```

13.2.2 The Frank–Wolfe step, and feasibility for free

Having found the corner s^k that the linear model prefers, the method moves a fraction of the way towards it, since jumping there would ignore the curvature of f :

$$x^{k+1} = (1 - \gamma_k)x^k + \gamma_k s^k, \quad \gamma_k = \frac{2}{k+2}, \quad k = 0, 1, 2, \dots \quad (13.6)$$

The step size is an open-loop schedule, fixed in advance: $\gamma_0 = 1$, $\gamma_1 = \frac{2}{3}$, $\gamma_2 = \frac{1}{2}$, decreasing to zero like $2/k$. Nothing is tuned, no line search is run and no smoothness constant L is needed, in contrast to the step $1/L$ of [Week 8](#). The method is also called the *conditional-gradient* method.

Proposition 13.3 (Feasibility for free). *If C is convex, $x^k \in C$, $s^k \in C$ and $\gamma_k \in [0, 1]$, then $x^{k+1} \in C$.*

Proof. x^{k+1} is a convex combination of two points of C , and a convex set contains the segment between any two of its points ([Definition 2.1](#)). $\square$

Started at any feasible point, every iterate is feasible without a repair step: the definition of a convex set used directly as an algorithm, the opposite of [Week 8](#), where feasibility was *restored* at the cost of a projection ([Figure 13.2a](#)).

[Algorithm 13.1](#) collects the method; each iteration costs one gradient and one oracle call. One property of [Week 8](#)'s method is given up: with the fixed schedule, Frank–Wolfe is *not* a descent method. The first step, with $\gamma_0 = 1$, jumps to the corner s^0 whatever x^0 was, and f can rise (in the measured portfolio run below it rises in 152 of 399 steps). What converges is the sequence, not each step, exactly as for the subgradient method of [Algorithm 7.1](#).

13.2.3 Worked example: the oracle answers with a corner

Example 13.4 (Projection onto a pentagon by linear questions). Let C be the pentagon with vertices $v_1 = (0, 0)$, $v_2 = (2.6, 0.35)$, $v_3 = (3.0, 2.1)$, $v_4 = (1.3, 3.0)$, $v_5 = (-0.2, 1.7)$, let $y = (2.85, 3.88)$, a point outside C , and minimise $f(x) = \frac{1}{2} \|x - y\|_2^2$ over C , so that $\nabla f(x) = x - y$. The minimiser is the projection of y onto the edge $[v_3, v_4]$: $x^* = 0.498v_3 + 0.502v_4 = (2.147, 2.552)$, with $f^* = 1.129$; no single corner is the answer. Start at $x^0 = v_1$. Then $\nabla f(x^0) = (-2.85, -3.88)$ and the five values $\nabla f(x^0)^\top v_i$ are 0, -8.77 , -16.70 , -15.34 , -6.03 , so $s^0 = v_3$ and $g_0 = \nabla f(x^0)^\top (x^0 - s^0) = 16.70$. With $\gamma_0 = 1$, $x^1 = v_3 = (3, 2.1)$. Now $\nabla f(x^1) = (0.15, -1.78)$ and the vertex values are 0, -0.23 , -3.29 , -5.14 , -3.06 , so $s^1 = v_4$, $g_1 = 0.15 \cdot 1.7 + 1.78 \cdot 0.9 = 1.857$, and with $\gamma_1 = \frac{2}{3}$, $x^2 = \frac{1}{3}v_3 + \frac{2}{3}v_4 = (1.867, 2.7)$. At x^2 the oracle returns v_3 again ($g_2 = 0.406$), $\gamma_2 = \frac{1}{2}$ gives $x^3 = (2.433, 2.4)$, and from then on the oracle alternates $v_4, v_3, v_4, \dots$ while the iterate settles on the edge between them: the suboptimality $f(x^k) - f^*$ is 10.46, 0.466, 0.050, 0.053, 0.018, 0.019 for $k = 0, \dots, 5$ ([Figure 13.2b](#)); neither g_k nor $f(x^k)$ decreases at every step. The method reaches a point C has no name for, a 0.498/0.502 mix of two corners, using only questions whose answers are corners.

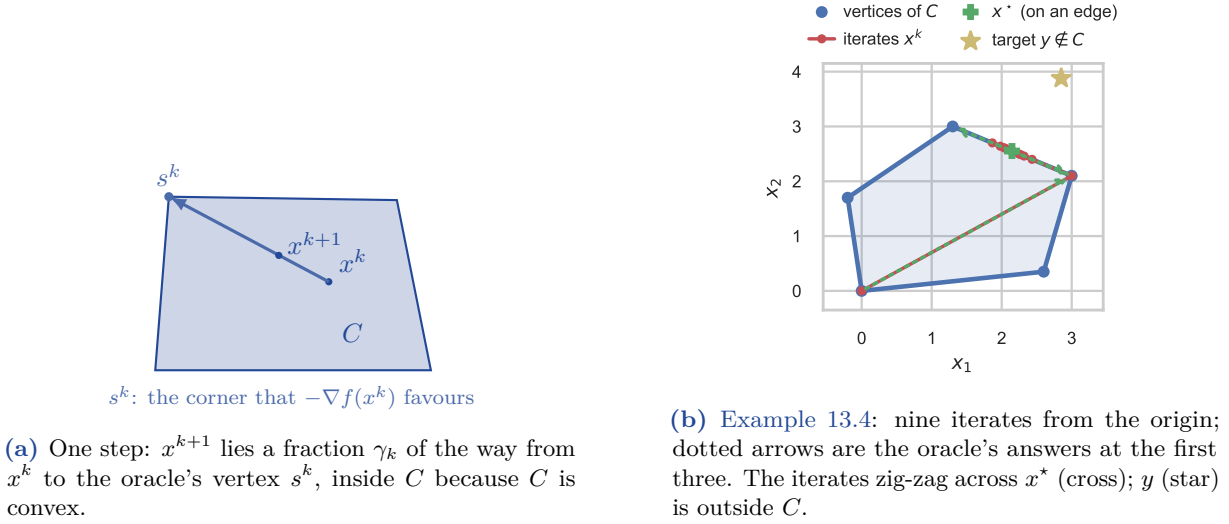


Figure 13.2: The geometry of Frank–Wolfe: no point outside C is ever produced.

13.2.4 The gap you already computed is a certificate

The oracle hands back s^k , so the number

$$g_k := \nabla f(x^k)^\top (x^k - s^k) = \max_{s \in C} \nabla f(x^k)^\top (x^k - s) \quad (13.7)$$

costs nothing extra (the maximiser of $\nabla f(x^k)^\top (x^k - s)$ over s is the minimiser of $\nabla f(x^k)^\top s$, which is s^k). It is called the **Frank–Wolfe gap**, and it is far more than a diagnostic.

Proposition 13.5 (The Frank–Wolfe gap bounds the suboptimality). *Let f be convex and differentiable, C nonempty compact convex, $x^k \in C$, and let x^* minimise f over C with $f^* = f(x^*)$. Then*

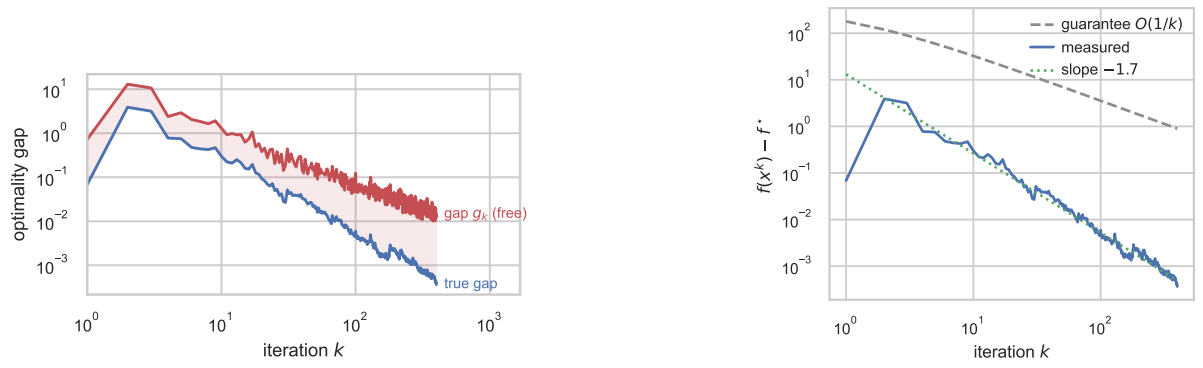
$$0 \leq f(x^k) - f^* \leq \nabla f(x^k)^\top (x^k - x^*) \leq \max_{s \in C} \nabla f(x^k)^\top (x^k - s) = g_k, \quad (13.8)$$

and $g_k = 0$ if and only if the variational inequality $\nabla f(x^k)^\top (y - x^k) \geq 0$ holds for all $y \in C$, that is, if and only if $0 \in \nabla f(x^k) + N_C(x^k)$ and x^k is optimal.

Proof. The first-order inequality of [Theorem 3.8](#) at x^k , evaluated at $y = x^*$, reads $f(x^*) \geq f(x^k) + \nabla f(x^k)^\top (x^* - x^k)$, which rearranges to the second inequality. Since $x^* \in C$, the value at $s = x^*$ is at most the maximum over C , which is the third; the equality is (13.7), and $s = x^k$ shows $g_k \geq 0$. Finally $g_k = 0$ means $\nabla f(x^k)^\top (x^k - s) \leq 0$ for every $s \in C$, the variational inequality at x^k , equivalent to $0 \in \nabla f(x^k) + N_C(x^k)$ by [Theorem 8.7](#). $\square$

Two things follow from (13.8). First, g_k is a computable *upper bound on how suboptimal the current iterate is*, valid at every iteration without knowing f^* , so stopping when $g_k \leq \varepsilon$ guarantees $f(x^k) - f^* \leq \varepsilon$. Projected gradient offered nothing as direct: its projected-gradient mapping $\|G_\alpha(x^k)\|_2$ ([Definition 8.16](#)) vanishes at the optimum, but turning it into a bound on the suboptimality needs a further constant, such as the diameter of C , whereas g_k is already a bound in the units of f . Second, $g_k = 0$ is not a proxy for optimality; it *is* the condition (13.4), evaluated. The stopping rule is the certificate.

[Figure 13.3a](#) measures the certificate on a long-only portfolio with $d = 60$ assets, the objective $-\mu^\top w + 4w^\top \Sigma w$ over the simplex ([Model 1.2](#) with risk-aversion weight 4; L , eight times the largest eigenvalue of Σ , is 89.65), started from the uniform portfolio and run for 400 iterations against a reference optimum computed to machine precision. The gap sits above the true suboptimality at every iteration, but it is a *bound*, not an estimate: the ratio $g_k/(f(x^k) - f^*)$ is



(a) The gap g_k dominates the true suboptimality $f(x^k) - f^*$ at every iteration; the shaded band is the looseness of the certificate.

(b) The suboptimality against the guarantee $2LD^2/(k+2)$ of Theorem 13.6; the dotted line has the fitted slope -1.7 .

Figure 13.3: Frank–Wolfe on the long-only portfolio ($d = 60$, 400 iterations, log–log scales).

about 4 at $k = 10$, about 11 at $k = 100$ and about 38 at $k = 399$, fluctuating on the way (its minimum is 3.1, its maximum 50). Always valid, sometimes loose: the certificate can be trusted in one direction.

13.2.5 One new corner per step: sparse iterates

The update (13.6) adds one vertex to the mix at each step and never removes the old ones. Since $\gamma_0 = 1$ discards x^0 , $x^1 = s^0$ is a single vertex, x^2 a combination of at most two, and x^k a convex combination of at most k vertices for $k \geq 1$ (at most $k+1$ counting a vertex start). On the simplex a vertex is a single asset, so the number of holdings rises by at most one per step: 1, 5, 17, 35 assets at $k = 1, 5, 20, 50$, against 44 in the optimal portfolio. Projected gradient from the same uniform start holds 60, 60, 56, 53 assets at $k = 1, 5, 20, 50$ and still 45 at $k = 399$; started instead from a single asset, one projected-gradient step with $\alpha = 1/L$ already holds between eight and sixteen of the 60, depending on the asset. Where the vertices mean something, one asset, one route, one product, the early Frank–Wolfe iterates are interpretable by construction, and feasible.

Note on the lecture slides

Frame 13.12 says that “one projection, by contrast, spreads weight across almost every asset immediately”. In the run both methods start from the uniform portfolio, so projected gradient *starts* with all 60 assets and sheds them slowly; no projection spreads anything. The notes report the run as it is and add a single-asset start for contrast.

13.2.6 The guarantee, and what the runs do

Theorem 13.6 (Frank–Wolfe rate). *Let f be convex and L -smooth (Definition 6.8), and let C be compact and convex with **diameter** $D = \max_{x,y \in C} \|x - y\|_2$. With $\gamma_k = 2/(k+2)$, the iterates of Algorithm 13.1 satisfy, for every $k \geq 1$ and any start $x^0 \in C$,*

$$f(x^k) - f^* \leq \frac{2LD^2}{k+2} = O(1/k). \quad (13.9)$$

Proof sketch. Write $h_k = f(x^k) - f^*$. The quadratic upper bound of Proposition 6.9, applied along $x^{k+1} - x^k = \gamma_k(s^k - x^k)$ with $\|s^k - x^k\|_2 \leq D$, gives $f(x^{k+1}) \leq f(x^k) - \gamma_k g_k + \frac{L}{2} \gamma_k^2 D^2$, and $g_k \geq h_k$ by Proposition 13.5, so $h_{k+1} \leq (1 - \gamma_k)h_k + \frac{L}{2} \gamma_k^2 D^2$. With $\gamma_0 = 1$ this yields

$h_1 \leq LD^2/2 \leq 2LD^2/3$, and if $h_k \leq 2LD^2/(k+2)$ then $h_{k+1} \leq 2LD^2[\frac{k}{(k+2)^2} + \frac{1}{(k+2)^2}] = \frac{2LD^2(k+1)}{(k+2)^2} \leq \frac{2LD^2}{k+3}$, because $(k+1)(k+3) \leq (k+2)^2$. $\square$

The bound (13.9) is of the same order as projected gradient's (Theorem 9.18 with $h = I_C$ and $\alpha = 1/L$, which gives $f(x^k) - f^* \leq L \|x^0 - x^*\|_2^2 / (2k)$), with LD^2 in place of $L \|x^0 - x^*\|_2^2$ up to a constant factor, and the method needs no knowledge of L to run: ε -optimality is guaranteed after $k \geq 2LD^2/\varepsilon - 2$ iterations. It holds from $k = 1$ on; at $k = 0$ it can fail, since x^0 is arbitrary. It is a worst case: on the portfolio (Figure 13.3b; $D^2 = 2$ for the simplex) it holds at every iteration and is about 760 times looser than the run at $k = 100$, whose decay is like $k^{-1.7}$, not k^{-1} . And it cannot be accelerated: unlike the $O(1/k)$ of projected gradient, which Week 10 lifted to $O(1/k^2)$ with the same primitive, a method that sees C only through a linear oracle cannot in general guarantee a better order than $1/k$ (in high dimension, for k up to about d). A worst-case bound is a promise, not a forecast; quote it for what it guarantees, and measure the run for what it does.

13.2.7 When Frank–Wolfe earns its place

Frank–Wolfe is the method of choice when the linear oracle is far cheaper than the projection, or is the only primitive available at all; when vertices are meaningful and sparse, feasible iterates are wanted; and when a computable optimality certificate matters. The first condition is about structure, not about polyhedra in general. On a generic polyhedron $\{Ax \leq b\}$ the oracle is a linear programme, which Figure 13.1 shows to be about as expensive as the projection QP; in Example 13.13, 59 LPs leave Frank–Wolfe far further from p^* than 39 projections leave projected gradient. Escape 1 pays where the oracle is *structurally* cheap: an **argmin** on the simplex, a signed **argmax** on the ℓ_1 ball, a shortest path on a network. It is the wrong choice when the projection is already cheap, because then Week 8's method is faster, simpler and can be accelerated; when high accuracy is needed, because the $O(1/k)$ order cannot be improved; and when the optimum lies on a face between several vertices, where the oracle keeps swapping among them and the iterates zig-zag across the face (the away steps of Section 13.6 repair this). Frank–Wolfe buys freedom from projection and pays for it in accuracy, as the subgradient method of Week 7 bought generality and paid in the same coin.

Note on the lecture slides

Frame 13.6's takeaway says that “linear questions about a feasible set are the ones optimisation software answers best”, and the act treats the oracle as the easier primitive. On the deck's own running polyhedron the LP oracle is about as expensive as the projection QP (frame 13.4's figure; dearer at $d = 800$), and in frame 13.23 Frank–Wolfe trails projected gradient. The notes therefore confine the advantage to sets whose oracle is structurally cheap, as frame 13.14's first “use it when” does.

Note on the lecture slides

Frame 13.14 lists, among the cases to avoid, “the optimum sits at a vertex the oracle keeps swapping away from”. As worded this can be the best case: with $y = (4.5, 2.5)$ in Example 13.4, $x^* = v_3$, and from $x^0 = v_1$ the oracle returns v_3 at every iteration, so $x^1 = x^*$. The swapping occurs when x^* lies on a face *between* vertices, Example 13.4's own situation; the notes state it that way.

13.3 Escape 2: moving the constraints into the objective

13.3.1 Pricing a constraint instead of enforcing it

The second escape needs nothing new: it is the machinery of [Week 12](#) applied to $Ax \leq b$. A constraint can be *priced* instead of enforced: the Lagrangian of [Definition 5.7](#) adds $\lambda_i \geq 0$ times the violation of constraint i to the objective, and the dual function $q(\lambda)$ ([Definition 5.8](#), written g there and q from [Definition 12.12](#) on) minimises the priced objective. In [Week 12](#) a shared budget $\sum_i x_i = R$ ([Model 12.11](#)) was priced by dual decomposition; because its price step must stay below a curvature-dependent limit that the coordinator cannot compute, above which the price oscillates ([Example 12.14](#)), the augmented term ([Definition 12.15](#)) and a split of the variables gave ADMM ([Algorithm 12.3](#)). For a general polyhedron the hard set likewise enters the objective, where the only question asked of it is a proximal one.

13.3.2 The split that makes $Ax \leq b$ easy

The running model is a convex quadratic programme over the polyhedron (13.2), the instance on which all three escapes are timed in [Section 13.5.1](#).

Model 13.7 (Convex QP over a general polyhedron, and its split form). Given $Q \in \mathbb{R}^{d \times d}$ symmetric positive semidefinite, $c \in \mathbb{R}^d$, and $A \in \mathbb{R}^{m \times d}$, $b \in \mathbb{R}^m$ with no structure that would make projection onto $\{Ax \leq b\}$ cheap,

$$\begin{aligned} \min_{x \in \mathbb{R}^d} \quad & f(x) = \frac{1}{2}x^\top Qx + c^\top x \\ \text{s.t.} \quad & Ax \leq b. \end{aligned} \tag{13.10}$$

Introducing $z = Ax \in \mathbb{R}^m$ and writing $I_{\{z \leq b\}}$ for the indicator of the box $\{z : z \leq b\}$ ([Definition 8.4](#)), the equivalent **split form** is

$$\begin{aligned} \min_{x \in \mathbb{R}^d, z \in \mathbb{R}^m} \quad & f(x) + I_{\{z \leq b\}}(z) \\ \text{s.t.} \quad & Ax - z = 0. \end{aligned} \tag{13.11}$$

The split form (13.11) is [Algorithm 12.3](#)'s template with second block $-I$ and right-hand side zero: a function of x plus a function of z , coupled only by a linear equation. The inequalities now constrain z alone, and a constraint on z alone is simple. Writing $\rho > 0$ for the penalty parameter and $u^k = \lambda^k / \rho \in \mathbb{R}^m$ for the scaled dual variable of [Week 12](#), the three ADMM updates of [Algorithm 12.3](#) become

$$x^{k+1} = \operatorname{argmin}_x \left\{ f(x) + \frac{\rho}{2} \|Ax - z^k + u^k\|_2^2 \right\} \iff (Q + \rho A^\top A)x^{k+1} = -c + \rho A^\top (z^k - u^k), \tag{13.12}$$

$$z^{k+1} = \operatorname{argmin}_{z \leq b} \frac{\rho}{2} \|Ax^{k+1} - z + u^k\|_2^2 = \Pi_{\{z \leq b\}}(Ax^{k+1} + u^k) = \min(Ax^{k+1} + u^k, b), \tag{13.13}$$

$$u^{k+1} = u^k + Ax^{k+1} - z^{k+1}. \tag{13.14}$$

The x -step (13.12) sets the gradient $Qx + c + \rho A^\top (Ax - z^k + u^k)$ to zero; the matrix $Q + \rho A^\top A$ is positive definite whenever Q is, or whenever A has independent columns, and it does not change with k , so one Cholesky factorisation at the start serves every iteration, each then a pair of triangular solves. The z -step (13.13) is the prox of the indicator, which is the projection ([Proposition 9.8](#)), here a componentwise clip, the *simple* set that [Table 8.1](#) began with. The u -step (13.14) accumulates the constraint residual. So each ADMM iteration is one cached linear solve plus one clip: no inner optimisation at all. The complexity was never in the inequalities themselves; it was in the matrix A coupling them to x , and splitting separates the two.

13.3.3 What splitting gives, and what it takes

Three things are gained. The $O(d^3)$ factorisation is paid once; every later step is $O(d^2 + md)$. The primal and dual residuals of [Theorem 12.17](#), here $\|Ax^k - z^k\|_2$ and $\rho \|A^\top(z^k - z^{k-1})\|_2$, provide a stopping rule, and, since its hypotheses hold whenever the QP has an optimal solution (f and $I_{\{z \leq b\}}$ are closed convex, and because 0 is strictly feasible [Theorem 5.13](#) supplies the multiplier), the same theorem guarantees convergence for any $\rho > 0$, at an $O(1/k)$ rate in general for the averaged iterate. And it scales: on the instance of [Section 13.5.1](#) twenty thousand ADMM steps took about four seconds, where one projection QP took a twentieth of a second.

Two things are given up. The iterates are *not feasible* until convergence: x^k is never returned to C , only z^k is clipped, and the two agree only in the limit; on the same instance the largest violation $\max_i(a_i^\top x^k - b_i)$ fell to 4×10^{-10} only after 20 000 steps. And ρ must be chosen, and a poor choice slows the run badly ([Section 12.5](#)). The first point is decisive for a business user: Frank–Wolfe and projected gradient can be stopped at any moment and their feasible iterate handed to a manager; ADMM's cannot.

Key idea

Ask whether a feasible answer is needed *during* the run or only at the end. If during, choose a primitive that keeps the iterates in C (a projection, or a linear oracle); if only at the end, move the constraints into the objective and let splitting make them a clip.

13.4 Escape 3: the log barrier and interior-point methods

13.4.1 The log barrier

The trouble with $Ax \leq b$ is its boundary: that is where the projection lands, where the oracle's corners sit and where the indicator jumps to $+\infty$. The third escape replaces the constraints by a penalty that is finite inside C and infinite at its edge, so that a minimiser can never sit on the boundary.

Definition 13.8 (Log barrier; barrier problem). For $C = \{x : Ax \leq b\}$ with rows $a_i^\top$, the **log barrier** is

$$\phi(x) = -\sum_{i=1}^m \log(b_i - a_i^\top x), \quad \text{defined on the strict interior } \{x : Ax < b\}, \quad (13.15)$$

with $\phi(x) \rightarrow +\infty$ as any slack $b_i - a_i^\top x$ falls to zero. For a convex differentiable f and a **barrier parameter** $t > 0$, the **barrier problem** is

$$\min_{x: Ax < b} t f(x) + \phi(x) = t f(x) - \sum_{i=1}^m \log(b_i - a_i^\top x). \quad (13.16)$$

Three facts make this work. The barrier problem is convex: $-\log$ is convex on $\mathbb{R}_{++}$ ([Example 3.6](#)), and composition with an affine map and nonnegative sums preserve convexity ([Proposition 3.19](#)). It is smooth and, on its open domain, *unconstrained*: the setting of [Week 6](#). And writing $\sigma_i = b_i - a_i^\top x > 0$ for the slacks,

$$\nabla \phi(x) = \sum_{i=1}^m \frac{a_i}{\sigma_i}, \quad \nabla^2 \phi(x) = \sum_{i=1}^m \frac{a_i a_i^\top}{\sigma_i^2} = A^\top \text{diag}(1/\sigma_1^2, \dots, 1/\sigma_m^2) A, \quad (13.17)$$

which Newton's method will need. The parameter t sets the trade: small t lets the barrier dominate and keeps the minimiser central; large t takes f seriously and pushes the minimiser towards the constrained optimum.

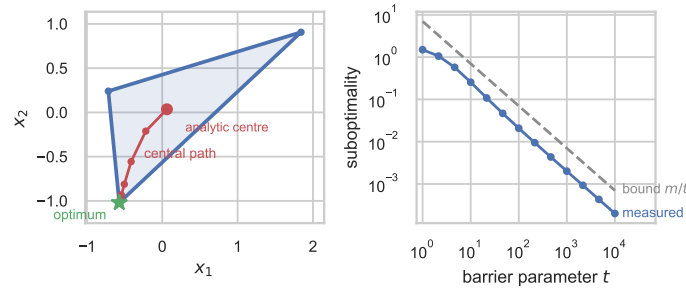


Figure 13.4: Example 13.10. Left: the feasible triangle and the central path for t from 1 to 10^4 , ending at the optimal vertex (star). The point labelled “analytic centre” is $x^*(1)$; the analytic centre proper ($t \rightarrow 0$) lies further back along the curve, away from the optimum, near $(0.52, 0.29)$. Right: $c^\top x^*(t) - p^*$ against the bound m/t of Proposition 13.11 (log-log); the true gap settles at $0.286 m/t = 2/t$.

13.4.2 The central path

Definition 13.9 (Central path; analytic centre). Assume that C is bounded and its strict interior $\{x : Ax < b\}$ is nonempty, so that for every $t > 0$ the barrier problem (13.16) has a unique minimiser $x^*(t)$ (a strongly convex f also guarantees this). The curve $\{x^*(t) : t > 0\}$ is the **central path** of the problem, and the minimiser of the barrier (13.15) alone, the limit of $x^*(t)$ as $t \rightarrow 0$, is the **analytic centre** of C . As $t \rightarrow \infty$, $x^*(t)$ approaches the set of optimal points.

Example 13.10 (The central path of a small LP). Figure 13.4 traces the path for the linear programme $\min c^\top x$ subject to $Ax \leq b$ in $\mathbb{R}^2$ with $c = (0.7, 1.0)$ and $m = 7$ random constraints, whose feasible region turns out to be a triangle (four constraints are redundant). Its optimum is the vertex $x^* = (-0.571, -1.020)$, $p^* = -1.4195$, with constraints 1 and 4 active. The analytic centre is $(0.520, 0.291)$; the path, each point found by damped Newton from the previous one, is drawn from $x^*(1) = (0.062, 0.035)$ to the optimal vertex as t grows to 10^4 .

13.4.3 How suboptimal is a point on the path?

The right panel of Figure 13.4 shows the measured suboptimality staying below the straight line m/t . That line is a theorem, and it comes from Week 5.

Proposition 13.11 (Duality gap on the central path). Consider the linear programme $\min c^\top x$ subject to $Ax \leq b$, with optimal value p^* , and let $x^*(t)$ be the point of its central path at parameter $t > 0$. Define

$$\lambda_i = \frac{1}{t(b_i - a_i^\top x^*(t))} > 0, \quad i = 1, \dots, m. \quad (13.18)$$

Then λ is dual feasible, the duality gap at $x^*(t)$ is exactly $c^\top x^*(t) - q(\lambda) = m/t$, and consequently

$$c^\top x^*(t) - p^* \leq \frac{m}{t}. \quad (13.19)$$

The same statement holds, with $f(x^*(t))$ in place of $c^\top x^*(t)$, for any convex differentiable f .

Proof. The Lagrangian of the LP is $L(x, \lambda) = c^\top x + \lambda^\top (Ax - b)$ with $\lambda \geq 0$ (Definition 5.7), and since it is affine in x the dual function is $q(\lambda) = \inf_x L(x, \lambda) = -b^\top \lambda$ if $A^\top \lambda + c = 0$ and $-\infty$ otherwise; dual feasibility means $\lambda \geq 0$ together with $A^\top \lambda + c = 0$. Now $x^*(t)$ is the unconstrained minimiser of the convex differentiable function $tc^\top x + \phi(x)$, so its gradient vanishes there: by

Algorithm 13.2 Barrier (path-following) method: an interior-point method

Require: convex twice-differentiable f ; $C = \{Ax \leq b\}$; a strictly feasible x ($Ax < b$); $t > 0$; tolerance $\varepsilon > 0$

Ensure: a strictly feasible $x \approx x^*(t)$ with $m/t \leq \varepsilon$, so $f(x) - p^* \lesssim \varepsilon$

- 1: **repeat**
 - 2: **centre:** starting from the current x , minimise $tf(x) + \phi(x)$ by damped Newton steps
 $x \leftarrow x + \alpha \Delta x$, $(t\nabla^2 f(x) + \nabla^2 \phi(x))\Delta x = -(t\nabla f(x) + \nabla \phi(x))$, with $\alpha \leq 1$ chosen by backtracking (keeping $Ax < b$, with sufficient decrease)
 - 3: **if** $m/t \leq \varepsilon$ **then return** x ▷ certificate of [Proposition 13.11](#)
 - 4: $t \leftarrow 10t$ ▷ any fixed factor > 1 ; warm start from the current x
 - 5: **until** stopped
-

(13.17), $tc + \sum_i a_i / (b_i - a_i^\top x^*(t)) = 0$, which, divided by t , reads $c + \sum_i \lambda_i a_i = c + A^\top \lambda = 0$ with the λ_i of (13.18). Since also $\lambda_i > 0$, λ is dual feasible and $q(\lambda) = -b^\top \lambda$. Then, using $c = -A^\top \lambda$,

$$c^\top x^*(t) - q(\lambda) = -\lambda^\top Ax^*(t) + \lambda^\top b = \sum_{i=1}^m \lambda_i (b_i - a_i^\top x^*(t)) = \sum_{i=1}^m \frac{1}{t} = \frac{m}{t}.$$

Weak duality ([Theorem 5.9](#)) gives $q(\lambda) \leq p^*$, and (13.19) follows. For a general convex f , the stationarity condition reads $\nabla f(x^*(t)) + A^\top \lambda = 0$, so $x^*(t)$ minimises the convex function $L(\cdot, \lambda) = f + \lambda^\top (A \cdot - b)$ and $q(\lambda) = f(x^*(t)) + \lambda^\top (Ax^*(t) - b) = f(x^*(t)) - m/t$; the rest is unchanged. $\square$

Accuracy can thus be bought outright, $t = m/\varepsilon$ for ε -optimality, because the barrier produces, as a by-product of stationarity, a dual feasible point, the object [Week 5](#) used to certify optimality, with a gap known in advance to be m/t ; this is why an interior-point solver can report a duality gap. On the LP of [Example 13.10](#) the measured gap at $t = 1, 100, 10^4$ is 1.50, 2.08×10^{-2} , 2.00×10^{-4} against $m/t = 7, 7 \times 10^{-2}, 7 \times 10^{-4}$, the ratio settling at $0.286 = 2/7$. The proof explains the 2: with $c = -A^\top \lambda$ and the optimal slacks $\sigma_i^* = b_i - a_i^\top x^*$, the true gap is $c^\top (x^*(t) - x^*) = \sum_i \lambda_i (\sigma_i(t) - \sigma_i^*)$. Each of the two active constraints ($\sigma_i^* = 0$) contributes $\lambda_i \sigma_i(t) = 1/t$, each inactive one $\frac{1}{t}(1 - \sigma_i^*/\sigma_i(t))$, a vanishing fraction of $1/t$ as its slack tends to a positive limit; so the true gap tends to $2/t$, whereas the bound charges $1/t$ for all seven.

13.4.4 Following the path: the barrier method

[Algorithm 13.2](#) follows the central path. Each outer stage solves one barrier problem by Newton's method ([Definition 6.15](#), with the Hessian of (13.17) added to $t\nabla^2 f$), warm-started at the previous $x^*(t)$; then t is multiplied by a fixed factor, until m/t is below the tolerance. The loop needs a strictly feasible start, found by a short preliminary computation called *phase I* (production solvers avoid it; see [Remark 13.12](#)). Warm starting is what makes the scheme cheap: multiplying t by ten moves the target only a little way along the path, so a handful of Newton steps suffices per stage. Starting from t_0 , the test $m/t \leq \varepsilon$ is passed after $\lceil \log_{10}(m/(t_0\varepsilon)) \rceil$ updates of t ; that each centring needs only a bounded number of Newton steps is a self-concordance result beyond this unit. On the LP of [Example 13.10](#), from $t = 1$ with $\varepsilon = 10^{-4}$, the six stages $t = 1, 10, \dots, 10^5$ took 3, 6, 7, 6, 6, 6 Newton steps, and ended with a true gap of 2.0×10^{-5} against the certified $m/t = 7 \times 10^{-5}$.

Why Newton, and not the gradient descent of [Week 6](#)? Near the boundary the Hessian (13.17) has terms of size $1/\sigma_i^2$ for the nearly active constraints beside moderate terms for the rest. The barrier is not even L -smooth on its open domain, and along the central path, whenever fewer than d constraints are active at the solution (typical for a QP), the condition number κ of [Definition 10.3](#) grows without bound as t increases, which [Week 10](#) showed is ruinous for first-order methods. Newton's step is invariant to that scaling, because it solves the local

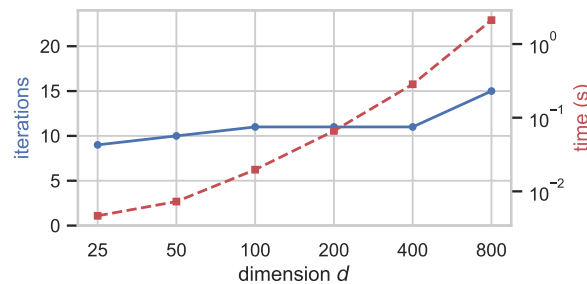


Figure 13.5: Interior-point iterations (left axis) and wall-clock time (right axis, log scale) for MOSEK on the QP (13.10), $m = 2d$, as d grows from 25 to 800 (one run): a near-constant number of steps, each an $O(d^3)$ factorisation.

quadratic model exactly, at the price of one $d \times d$ solve, $O(d^3)$, per step. Every iterate stays strictly inside C , an *interior point* in the sense of Week 8, where $N_C(x) = \{0\}$ (Definition 8.5); this names the family of **interior-point methods**, of which the barrier method is the textbook member.

13.4.5 Why a solver call feels so strong

Figure 13.5 profiles an interior-point solver (MOSEK through RSOME) on the running QP (13.10) with $m = 2d$. Over a $32\times$ range of d the iteration count moves only from 9 to 15, while the time moves from 0.005 s to 2.1 s: each iteration is a factorisation of a $d \times d$ system. The near-constant iteration count is the signature of interior-point codes, and the trade of the unit is in this one picture: interior point buys accuracy, with a certificate, by paying a factorisation per iteration; first-order methods buy scale by refusing to pay it. Methods of this kind are what conic solvers such as MOSEK run behind many of the `m.solve()` calls of Section 4.5, which is why the modelling route of Week 4 can return a certified gap.

Remark 13.12 (Textbook barrier method versus production solvers). Strict feasibility of every iterate belongs to Algorithm 13.2. Production interior-point codes, MOSEK included, run an infeasible-start primal–dual scheme in which feasibility arrives only with optimality, so a solver’s intermediate iterates cannot supply a feasible answer during the run.

13.5 Choosing an escape, and the course in one map

13.5.1 One problem, three escapes, one clock

Rates count iterations, but an LP, a clip and a factorisation are not the same iteration; the only common axis is wall-clock time on one instance.

Example 13.13 (Three escapes on one QP). Take Model 13.7 with $d = 200$, $Q = GG^\top/d + 0.1I$ for a random G , and $m = 800$ inequalities: 400 random rows of the running polyhedron (13.2) plus the box $-10 \leq x_i \leq 10$, added so that C is compact for the linear oracle (it does not bind: $\|x^*\|_\infty = 3.37$). In one run an interior-point solver (MOSEK through RSOME) finished in 14 iterations and 0.06 s, with a certified gap. By then projected gradient, each step a projection QP (13.3), had not finished its first projection, and Frank–Wolfe, each step an LP, had completed one. After 39 projections (2.0 s) projected gradient was within 0.1 of p^* ; after 59 LPs (2.5 s) Frank–Wolfe was still 9 above it. ADMM with $\rho = 1$, one Cholesky factorisation and 20 000 solve-and-clip steps reached solver accuracy in about 4 s: an objective error of 2.8×10^{-9} against the exact optimal value (from the KKT system on the active set) and a violation of 4×10^{-10} (Figure 13.6). The solver wins outright at this size; of the first-order escapes only the one with no inner problem competes, and its iterates are infeasible on the way.

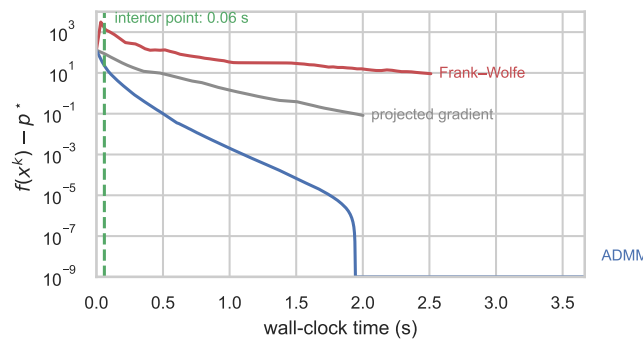


Figure 13.6: Example 13.13: $f(x^k) - p^*$ against wall-clock time (one run); the vertical line is the solver's time. The reference p^* is the solver's value, a few millionths above the exact optimal value, so the plunge of the ADMM curve near 2 s marks the moment ADMM becomes more accurate than the reference, not 10^{-9} accuracy.

Table 13.1: Choosing between the three escapes.

Escape	Cheap primitive	Certificate	Feasible during the run?	Wins when
Frank–Wolfe	a linear oracle	the gap g_k , free	yes, always	the oracle is cheap and sparse answers matter
Dualise and split (Week 12)	clip + cached solve	primal and dual residuals	no, only in the limit	the split makes both steps trivial
Interior point	a Newton solve	duality gap $\leq m/t$	yes, strictly inside (but see Remark 13.12)	accuracy matters and d is moderate

Table 13.1 summarises the comparison. All three escapes certify the same condition (13.4); they differ in what they can compute cheaply and in what they can promise while they run.

Key idea

Choose by the primitive you can afford and the certificate you must produce, not by the headline rate.

13.5.2 The unit in one question

Every week from Week 6 on answered the same question about a different structure: *what is cheap to compute here, and what does it certify?* A cheap gradient means gradient descent or acceleration (Weeks 6 and 10), and an affordable Hessian solve means Newton's method (Week 6); a cheap subgradient and nothing more, the subgradient method (Week 7); a cheap projection, projected gradient (Week 8); a cheap prox, proximal gradient or FISTA (Weeks 9 and 10); a cheap sampled gradient, SGD (Week 11); a cheap partial derivative or scalar prox, coordinate descent (Week 12); a cheap linear oracle, Frank–Wolfe. If nothing about the problem as a whole is cheap but its pieces are, dualise and split (Week 12); otherwise model the problem and hand it to a conic solver (Week 4), typically an interior-point method. The primitive names the method.

13.5.3 One optimality condition, four notations; one residual per structure

The unit stated one optimality condition in four notations: $\nabla f(x^*) = 0$ for a smooth unconstrained problem (Theorem 5.1); $0 \in \partial f(x^*)$ for a nonsmooth one (Theorem 7.7); $0 \in \nabla f(x^*) + N_C(x^*)$ under constraints (Theorem 8.7); $0 \in \nabla g(x^*) + \partial h(x^*)$ for a composite objective $g + h$, with g smooth and h prox-friendly (Theorem 9.5). Each says that no improving direction remains, and

Table 13.2: The residual to monitor depends on the structure.

Structure	Optimality condition	What is monitored	Week
smooth, unconstrained	$\nabla f(x^*) = 0$	$\ \nabla f(x^k)\ _2$	6
nonsmooth	$0 \in \partial f(x^*)$	no single reliable residual: one subgradient norm need not be small	7
simple constraints	$0 \in \nabla f + N_C$	the projected-gradient mapping $\ G_\alpha(x^k)\ _2$	8
composite $g + h$	$0 \in \nabla g + \partial h$	the same $\ G_\alpha(x^k)\ _2$, with prox for Π	9
finite sum, sampled	$\nabla F(x^*) = 0$	running averages; no monotone quantity exists	11
split or dualised	KKT for the split form	primal and dual residuals	12
complex C , linear oracle	$0 \in \nabla f + N_C$	the Frank–Wolfe gap g_k	13
barrier / solver	KKT	duality gap $\leq m/t$; solver status	13, 4

the last two are one statement twice, since $N_C = \partial I_C$ (Definition 8.5): [Week 8](#) is [Week 9](#) with $h = I_C$. What differs between the weeks is the *computable residual* that monitors the condition ([Table 13.2](#)). A stopping rule is meaningful only when the method controls the quantity watched: the projected- and proximal-gradient mappings, ADMM’s residuals, the Frank–Wolfe gap and m/t are driven to zero by their own methods, whereas a subgradient norm need not be small near a nonsmooth optimum and nothing is monotone along an SGD run. In the tables, $F = \frac{1}{N} \sum_{i=1}^N f_i$ is [Model 11.1](#)’s finite sum, $\bar{x}^T$ the averaged SGD iterate after T steps, and KKT the conditions of [Definition 5.14](#).

13.5.4 One ladder, four mistakes and two ways to trust an answer

Read down the second column of [Table 13.3](#), the unit is one search for the cheapest primitive a structure allows; read across, it records what each assumption bought. Smoothness moved $O(1/\sqrt{k})$ to $O(1/k)$ ([Table 7.1](#)), the single biggest gain in the unit. Strong convexity moved $O(1/k)$ to a linear rate ([Theorem 6.13](#)); in the subgradient and stochastic methods the fixed-step floor gave way only to a decaying step ([Theorem 7.18](#)) or to vanishing noise at the solution ([Corollary 11.14](#)). Structure decided which primitive was affordable. Two rates no algorithmic device, momentum included, could improve within their class, $O(1/\sqrt{k})$ for nonsmooth problems ([Remark 7.17](#)) and $O(1/\sqrt{T})$ for sampled gradients ([Theorem 11.18](#)), and this week added a third, $O(1/k)$ in high dimension for a method that sees C only through a linear oracle ([Section 13.2](#)). Speed came back only through structure: a composite form ([Week 9](#)), which acceleration then exploits ([Week 10](#)); a finite sum, which variance reduction exploits ([Section 11.5](#)); and a set whose projection is cheap after all ([Week 8](#)). Rates are statements about assumptions; when a rate cannot be improved, look for unused structure.

Four mistakes recur, each visible before a line of code is written: projecting onto a set with no cheap projection (use a linear oracle, or move the constraints into the objective); coding a first-order method for a medium-sized conic model (model it in RSOME and let an interior-point solver finish it); reading a worst-case rate as a forecast (measure the run); and stopping on a residual the method does not control (use the certificate matching the structure).

As for trust, convex structure bought *machine-checkable evidence*, the certificates of [Table 13.2](#): one does not trust a solver, one checks its certificate, which is what the convex modelling of [Week 4](#) purchases. In large-scale machine learning no certificate declares a training run optimal; theory is a compass there, explaining behaviour (the two phases of [Definition 11.19](#), the noise floor) while held-out data decides. Bertsekas spoke of optimisation’s two cultures, analysis and experiment (via M. Schmidt’s tutorial); the working analyst needs both, and knows which situation applies.

Table 13.3: One ladder, the whole course: each method's cheap primitive, the structure it exploits and its headline rate.

Method	Cheap primitive	Structure exploited	Headline rate
Gradient descent (Week 6)	∇f	smooth, unconstrained	$O(1/k)$; linear $(1 - \mu/L)^k$ if μ -strongly convex
Subgradient (Week 7)	$g \in \partial f$	none: general nonsmooth	$O(1/\sqrt{k})$; $O(1/k)$ if strongly convex; not accelerable
Projected gradient (Week 8)	Π_C	smooth + simple set	$O(1/k)$; linear $(1 - \mu/L)^k$ in $\ x^k - x^*\ _2$ if μ -strongly convex
Proximal gradient (Week 9)	$\text{prox}_{\alpha h}$	smooth + cheap prox	$O(1/k)$; linear $(1 - \mu/L)^k$ in $\ x^k - x^*\ _2$ if the smooth part is μ -strongly convex
Accelerated, FISTA (Week 10)	the same, + extrapolation	smooth (+ cheap prox)	$O(1/k^2)$; $(1 - 1/\sqrt{\kappa})^k$ if μ -strongly convex with μ known
SGD (Week 11)	one sampled ∇f_i	finite sum, huge N	$O(1/\sqrt{T})$ on $\bar{x}^T$, and optimal; $O(1/k)$ in $\mathbb{E} \ x^k - x^*\ _2^2$ if strongly convex, with a decaying step
Coordinate, ADMM (Week 12)	block solve, prox	separable or coupled	$O(1/k)$: in expectation for randomised coordinate descent (linear if strongly convex), for the averaged iterate in ADMM
Frank–Wolfe (this week)	a linear oracle	compact C , costly projection	$O(1/k)$
Interior point (this week)	a Newton solve	any convex conic model	$\approx$ constant iterations, each $O(d^3)$

13.6 Summary and looking ahead

When the feasible set has no cheap projection, the certificate $0 \in \nabla f(x^*) + N_C(x^*)$ survives and only the primitive chasing it changes: a linear oracle gives Frank–Wolfe, splitting gives Week 12's ADMM with a clip, a log barrier gives the interior-point method (Table 13.1). Each continues beyond the unit: Frank–Wolfe in Jaggi (2013), the *away steps* of Lacoste-Julien and Jaggi (2015), and Wirth, Kerdreux and Pokutta (2023) on why $2/(k+2)$ often beats $1/k$; interior point in Boyd and Vandenberghe (chapter 11) and Nesterov and Nemirovski's self-concordance; splitting in Boyd et al. (2011) and Bertsekas (1996); beyond, proximal point (Parikh and Boyd, §4) and monotone operators (Ryu and Boyd, 2016). None changes the question one asks.

That question is the shape of the unit. A decision becomes a model (Weeks 1 and 4); convexity, which turns a local check into a global claim, and structure are checked (Weeks 2 and 3); the certificate is named (Week 5); the cheap primitive is found and names the method (Weeks 6 to 13); and one chooses between a solver call and a custom method, two settings of one dial. Optimisation is a language for decisions, not a single algorithm, and the question to carry out of the unit is the one asked of every method here: what is cheap to compute, and what does it certify?

13.7 Exercises

Exercise 13.1 (Which sets are simple?). For each set, say whether projection onto it is cheap, and give its linear minimisation oracle: (a) $\{x : \|x\|_2 \leq r\}$; (b) $\{x \geq 0 : \mathbf{1}^\top x = 1\}$; (c) $\{x : \|x\|_1 \leq \tau\}$; (d) $\{x : Ax \leq b\}$ with A a general $m \times d$ matrix; (e) the set in (b) intersected with sector caps $Sx \leq c$. For any set where projection is expensive but the linear oracle is cheap, say what the oracle returns and why it is a vertex.

Exercise 13.2 (One Frank–Wolfe step by hand). Minimise $f(x) = \frac{1}{2} \|x - y\|_2^2$ with $y = (0.9, 0.4, 0)$ over the simplex in $\mathbb{R}^3$, starting from $x^0 = (1, 0, 0)$. (a) Compute $\nabla f(x^0)$ and

run the linear oracle; which vertex comes back? (b) Take the step with $\gamma_0 = \frac{2}{0+2} = 1$; where is x^1 ? (c) Compute the gap g_0 ; what does it certify about $f(x^0) - f^*$? (d) Repeat once more and compute g_1 . You will find $g_1 > g_0$; explain why this is consistent with [Proposition 13.5](#): the gap is always a *valid* bound on the suboptimality, not a monotone one. How many vertices can x^k involve after k steps, and why does that matter to someone who has to act on the answer?

Exercise 13.3 (The barrier and its certificate). Take $\min c^\top x$ subject to $Ax \leq b$ with $m = 4$ inequalities in $\mathbb{R}^2$. (a) Write the barrier objective at parameter t and explain why it is convex. (b) Give λ_i in terms of the slacks at $x^*(t)$ and verify that $\lambda \geq 0$. (c) What value of t guarantees a solution within 10^{-4} of optimal? (d) Why does the central path approach the optimum from strictly inside the feasible set? The iteration count of an interior-point method barely grows with problem size, but the time per iteration does: which worries you for $d = 10^6$?

Exercise 13.4 (Choose an escape). For each model, choose between a linear oracle, dualise-and-split and interior point, defending the choice by cost, certificate and feasibility during the run; name the primitive first. (a) A portfolio on the simplex with sector caps and a turnover limit, where the committee wants a tradeable portfolio at any point during the optimisation. (b) A blending model with 200 variables and 600 constraints, needed to six figures for an audit. (c) A network-flow model where the linear subproblem is a shortest-path computation.

Exercise 13.5 (Course-wide review). Without looking at [Table 13.3](#), fill in the cheap primitive, the structure exploited and the headline rate for gradient descent, the subgradient method, projected gradient, proximal gradient, FISTA, SGD, coordinate descent, ADMM, Frank–Wolfe and interior point. Which rates in this unit cannot be improved within their class, even by momentum, and what is the reason in each case?

Exercise 13.6 (Coding task). Start from the companion notebook, `week13_constraints.ipynb`. (a) Implement the Frank–Wolfe loop for the ℓ_1 ball, whose oracle is one `argmax` with a sign; (b) verify that the gap certificate holds at every iteration and plot it against the true suboptimality; (c) add projected gradient with an inner QP for the projection and compare the two in wall-clock time; (d) solve the model in RSOME with an interior-point solver, comparing accuracy and time. Justify by measurement, not preference, when to iterate and when to call a solver.

Appendix: Discrepancies between the notes and the lecture slides

The notes treat the lecture slides as the primary source. Wherever the notes depart from a slide—because the slide appears to contain an error, an ambiguity, or an inconsistency with another week—the departure is recorded here, so that nothing is changed silently. Entries marked *boxed* are also flagged in the text in a *Note on the lecture slides* box; the others are small matters (a missing side condition, a term used before its definition, a figure drawn slightly off) that the notes simply state correctly. Frame numbers refer to the footline of the slides ($N.M$ = Week N , frame M).

Week 1

- **1.2, 1.3, 1.5** *boxed* The finance problem is stated three ways (minimum risk for a return target; maximum $\mu^\top w$ under a risk limit, with undefined data “ R ”; the weighted trade-off), and the λ of the 1.3 table is undefined. *In the notes*: the trade-off is the running model, R is read as a risk limit (taken up in Week 4) and λ as a regularisation weight.
- **1.13, 1.14, 1.21** The optimal value is defined twice ($f(x^*)$ and an infimum or supremum), the three statuses are not exhaustive in general, and the empty-set convention is missing. *In the notes*: the definition points to $z^* = \inf f$ or $\sup f$, a remark gives $\max x$ subject to $0 \leq x < 5$, and $\sup \emptyset = -\infty$, $\inf \emptyset = +\infty$ are stated.
- **1.15–1.18 vs 1.34 (and 1.17)** “Convex” is never defined, and local and global minima and nonsmoothness are used before 1.34 defines them loosely. *In the notes*: convexity gets an informal meaning (graph never above a chord) pointing to Weeks 2–3; local minima are defined with a ball of radius r ; nonsmooth means having kinks.
- **1.19 (and 1.18)** The semester table calls Week 13 a synthesis only, whereas its deck first treats optimisation with complex constraints; 1.18 says Weeks 2–5 build the language of the central promise. *In the notes*: the course plan describes Week 13 as both, and says that the promise is made precise and proved over the next three weeks, in Week 4.
- **1.12 (vs Weeks 4–5)** Week 1 writes the objective f and constraints $g_i(x) \leq 0$, $h_j(x) = 0$, while the standard form of Week 4 and the Lagrangian of Week 5 write f_0 , f_i and affine equalities. *In the notes*: f , g_i , h_j are kept, and one sentence announces the f_0 , f_i notation of Weeks 4–5, with the same meaning.
- **1.32** The letter b is both the offset of the affine function $a^\top x + b$ and the right-hand side of the halfspace $a^\top x \leq b$; “equality constraints define hyperplanes” needs “linear”; $a \neq 0$ is not required. *In the notes*: the definition keeps the deck’s forms with $a \neq 0$, the halfspace is $g(x) \leq 0$ for the affine $g(x) = a^\top x - b$, and the text speaks of linear equality constraints.
- **1.6, 1.24, 1.31** Least squares switches from β , $1/N$ to x , $\frac{1}{2}$ unbridged; the normal equations precede their gradient; on 1.31 f is both outer and composite function and the offset changes sign; the chain rule is not argued. *In the notes*: a bridge shows $\frac{1}{N} \sum_i (b_i - a_i^\top \beta)^2 = \frac{1}{N} \|A\beta - b\|_2^2$; a chain-rule sketch yields the normal equations.
- **1.28 (REGISTRY note)** The course registry lists the Week 1 directional derivative as a use of the direction symbol d^k , but the deck writes $Df(x; v)$ with direction v . *In the notes*: v is kept and no overload is mentioned in Week 1.
- **1.28 (gradient figure), 1.26, 1.13, 1.32, 1.27, 1.33, 1.14, 1.24** Cosmetic: schematic gradient ellipses, a misplaced tick label, a $z^* = 13$ label, no normal on the halfspace, schematic

contour labels, omitted pictures. *In the notes:* true level sets of $x_1^2 + 2x_2^2$ with $\nabla f(1, 1) = (2, 4)$, labels by equation, normal a drawn, prose for the rest.

- **1.4, 1.5, 1.13, 1.15–1.16, 1.23, 1.24, 1.26, 1.44** Additions, not errors: the deck has no media-mix solution, no proofs, no least-squares instance, no convexity definition. *In the notes:* (all checked) $x^* = (10000, 0, 0)$, $z^* = 40,000$, the portfolio read through γ , an LP certificate, short proofs, a three-point example, a convexity gloss, setup URLs in a footnote.

Week 2

- **2.4, 2.7–2.10 vs 2.12, 2.26, 2.32–2.33** *boxed* x_1, x_2 denote two points of $\mathbb{R}^d$ in the definitions of convex, conic and affine combinations, but two coordinates of one point in the quick check, the RSOME model and the tutorial; the 2.10 figure labels both its axes and its points x_1, x_2 . *In the notes:* the symbols are kept, each use says which reading is meant, and proofs write x, y for two points.
- **2.8 vs 2.12, 2.32 (Q1)** *boxed* The quick check and tutorial Q1 ask whether a set is a “cone”, but only “convex cone” is defined, and the takeaway reads cones as sets closed under nonnegative scaling. *In the notes:* “cone” means convex cone, shown equivalent to convex and closed under nonnegative scaling, so every answer is the same under either reading.
- **2.16 (also 2.12, 2.20, 2.21)** *boxed* The norm-cone argument invokes the triangle inequality, introduced in neither Week 2 nor Week 1, and the convexity of balls is asserted and relied on but never argued. *In the notes:* a lemma derives homogeneity and the triangle inequality from Week 1’s Cauchy–Schwarz inequality, and both claims are proved.
- **2.26 (listing)** *boxed* The comment `# 2 = optimal` is Gurobi’s status code, and the import of `grb_solver` fails unless Gurobi is installed. *In the notes:* the listing is kept with its real Gurobi output; a box adds that Week 1 recommends the default solver first, which prints 0 (MOSEK prints `Optimal`), with the same solution.
- **2.6, 2.13, 2.14** The condition $a \neq 0$ is missing from the halfspace recall and the hyperplane, halfspace and slab definitions (with $a = 0$ the sets are $\mathbb{R}^d$ or empty, and “a convex cone iff $b = 0$ ” fails); slab, box and ball lack side conditions. *In the notes:* $a \neq 0$ is added with a sentence on the degenerate case, as are $\alpha \leq \beta$, $\ell \leq u$ and $r \geq 0$.
- **2.14 (and 2.3 figure), 2.20–2.21** C is both a generic convex set and the matrix of $Cx = e$; A, b are polyhedron data, an affine map and least-squares data; the inverse-image rule is stated for $Ax + b$ but applied to $Ax - b$. *In the notes:* the symbols are kept, a sentence at each use says what they denote, and the offset $-b$ is noted.
- **2.23 figure, 2.16 figure** The perspective figure draws the slice labelled $t = 3$ at height 2.8, and the norm-cone rays have slope 1.048 rather than 1. *In the notes:* the adapted figures draw that slice at height 3 and the rays at 45° .
- **2.19–2.23, 2.27, 2.30** The closure rules are stated without proof, and 2.23 never says that the perspective cone of a convex set is a convex cone. *In the notes:* each rule and the perspective-cone claim get a short proof, and the norm cone is identified as the perspective cone of the unit ball.
- **2.7, 2.11** The convex hull is defined as all convex combinations of a set’s points, omitting “finitely many”, and the link between the two-point definition and “closed under convex combinations” is left implicit. *In the notes:* the definition says finitely many, and an induction proves k -point closure (affine sets too), so the hull is the smallest convex superset.

- **2.26** The feasible set is called the variable-budget cone “sliced by a box”; the box is $0 \leq t \leq 10\,000$ in the t -coordinate, and $t \geq 0$ already holds in the cone, so only $t \leq 10\,000$ cuts anything. *In the notes:* the deck’s wording is kept with a half-sentence reconciling it, and the set is noted to be a polyhedron.
- **2.26, 2.33 (Q6)** The unit of the charge of 3.5 “per dollar” is not stated, yet $4x_1 + 3x_2 + 2x_3 - 3.5t$ is a single quantity only if the charge is measured like the returns. *In the notes:* the charge is read in conversions per dollar, so the objective counts net conversions and a budget dollar pays only if it buys more than 3.5 conversions.
- **2.22 (vs Week 8)** “Projection” here means coordinate projection, dropping auxiliary variables, whereas from Week 8 the same word means the nearest-point projection onto a convex set. *In the notes:* a one-sentence vocabulary warning points ahead to Week 8.
- **2.31–2.33 (tutorial)** The exercises adapt the tutorial, several of whose questions the lecture already answers (Q1(a) and (d), the residual ball of Q4, the model of Q6). *In the notes:* Q1 asks for closure-rule or counterexample justifications, Q3 names the standard sets, Q4 adds $x \geq 0$, Q5 adds a Python check (from 2.31), and Q6 adds budget costs 4.5 and 2.5 (verified).
- **REGISTRY running-examples table** The registry says that Week 1’s small LP ($\max 2x_1 + 3x_2$ subject to $x_1 \leq 3$, $x_2 \leq 3$, $x_1 + x_2 \leq 5$, $x \geq 0$) reappears in Week 2, but the current deck does not contain it. *In the notes:* that LP is not revisited, and other weeks should not cite Week 2 for it.

Week 3

- **3.16 (also 3.18, 3.20, tutorial A4; 3.9–3.10) boxed** The Hessian test is stated on a convex domain $\mathcal{X}$, but “convex $\Rightarrow \nabla^2 f \succeq 0$ ” needs an open domain ($x_1^2 - x_2^2$ on the line $x_2 = 0$), and only $Q \succeq 0$ is defined although $Q \succ 0$ and strict convexity are used. *In the notes:* both matrix conditions are defined, the derivative tests are stated on open domains with their strict versions, and a box gives the counterexample.
- **3.19 (vs 3.17–3.18) boxed** The two-dimensional test writes $ax_1^2 + 2bx_1x_2 + cx_2^2 + \dots$ with $Q = \begin{bmatrix} a & b \\ b & c \end{bmatrix}$, that is $x^\top Qx$ without the factor $\frac{1}{2}$, so the Hessian is $2Q$; the letters q, b, d_1, d_2 clash with earlier meanings, and only $a > 0$ is covered. *In the notes:* the convention $\frac{1}{2}x^\top Qx$ is kept, the case $a = 0$ is proved, and a box records the factor 2.
- **3.8 boxed** Strict convexity is said to make minimisers unique, “an argument we run in Week 5”, but the midpoint argument is run in Week 4 and Week 5 only states the conclusion. *In the notes:* the midpoint argument is given where strict convexity is discussed, and a box points to Week 4.
- **3.30, 3.35 boxed** Two forward references lead nowhere: robust losses “in Week 7” (Week 7 keeps $|x|$ kinked and uses subgradients; Huber never returns), and conjugacy as a lead-in to duality and regularisation (no later deck uses a conjugate). *In the notes:* both topics are self-contained, the Huber example is linked to Week 9’s soft-thresholding, its inner minimiser, and a box states both points.
- **3.4 (vs 3.7, 3.12)** “Convex function” and “the familiar convexity inequality” are used in the epigraph equivalence before the chord definition is given, and no earlier week defines a convex function. *In the notes:* the chord definition (with strict convexity, concavity and affine functions) comes first, and both directions of the equivalence are proved.
- **3.1, 3.2, 3.15** “No spurious local minima” is promised from the first-order inequality, but 3.15 proves only “stationary $\Rightarrow$ global” for differentiable f , and “local $\Rightarrow$ global” rests on a

picture. *In the notes:* part (a) of the theorem uses the first-order inequality, part (b) the chord argument without differentiability, and its limits are stated.

- **3.14, 3.15 (also 3.9)** The first-order results are stated for differentiable f on a convex $\mathcal{X}$ without saying where f is differentiable when $\mathcal{X}$ is not open. *In the notes:* they are stated on an open convex domain and noted to hold whenever f is differentiable on an open set containing $\mathcal{X}$; only the Hessian test needs $\mathcal{X}$ open.
- **3.3, 3.12, 3.35** Two domain conventions coexist, $f: \text{dom } f \rightarrow \mathbb{R}$ and $f: \mathbb{R}^d \rightarrow \mathbb{R} \cup \{+\infty\}$. *In the notes:* two sentences reconcile them (a point with $f(x) = +\infty$ never satisfies $f(x) \leq t$, so the epigraph is the same), and $\text{dom } f$ is used throughout.
- **3.22** The subgradient inequality is required for all y , with no domain, and “at a differentiable point the only subgradient is the gradient” is stated without qualification or reason. *In the notes:* the inequality is required for $y \in \text{dom } f$, and the statement is made at interior points of $\text{dom } f$, with a one-line proof of existence and uniqueness.
- **3.27** The perspective is called the analogue of Week 2’s “perspective mapping”, but Week 2 defines a perspective cone, and that the edge $t = 0$ adds only $(0, 0)$ and keeps the epigraph convex is not justified. *In the notes:* rule (c) is proved directly, its epigraph is the perspective cone of $\text{epi } f$ without the adjoined origin, and the edge is checked.
- **3.29, 3.30** “ $\text{epi } f$ is the projection of $\text{epi } g$ ” is exact only if the infimum over z is attained, and the rule needs $\inf_z g(x, z) > -\infty$ for every x ; infimal convolution has the same proviso. *In the notes:* rule (e) assumes a finite infimum and is proved by an ε -argument, with the attainment caveat; the infimal convolution is convex whenever finite.
- **3.32 (vs 3.25, 3.27)** The hull formula allows $t_i = 0$, where $t_i f_i(x_i/t_i)$ is undefined, as the example needs; the x_i are vectors splitting x ; N counts functions here but records on 3.25. *In the notes:* a term with $t_i = 0$ is read as 0 and requires $x_i = 0$, the reading through $y_i = x_i/t_i$ and the roles of x_i and N are stated, and the example is derived step by step.
- **3.35, 3.36** “The best affine lower bound with slope y ” presupposes that $f^*(y)$ is finite, and the example sets a derivative to zero without saying why the stationary point is a maximiser. *In the notes:* the supremum may be $+\infty$ and $\text{dom } f^*$ is defined, the reading is stated for $y \in \text{dom } f^*$, and the example notes that the function maximised is concave.
- **3.37** The status print is said to confirm optimality, but no output is shown; with `grb_solver` the status is the integer 2, and the solution is never reported. *In the notes:* the output is reported, 2 is explained as Gurobi’s code for “optimal”, and the exact solution $x^* = (29/35, 26/35)$, $t^* = 38/35$ is given, all three pieces active.
- **3.25** The sign condition $\lambda \geq 0$ on the penalty weight in the loss-plus-penalty example is left implicit. *In the notes:* $\lambda \geq 0$ is stated explicitly.
- **3.40** Step 4 of the recognition checklist omits the Hessian test, and step 3 says “projection” for partial minimisation, which could be confused with Week 8’s projection onto a set. *In the notes:* the Hessian test is added to step 4, and step 3 reads “projection (partial minimisation)”.
- **3.43–3.44 (tutorial A3, A6, B1, B4, B5, mini-lab)** Some tutorial items are worked in the lecture, and two ask only for a statement. *In the notes:* A3 is proved in the body and kept as an exercise; B4 and B5 use new data and point to the worked examples; A6, B1 and the mini-lab gain instances or a follow-up question.
- **3.22–3.23 (vs REGISTRY and Week 5)** The registry dates subgradients and subdifferentials to Week 7, and Week 5 presents them as new, yet 3.22–3.23 already define both. *In the notes:* the mathematics is unchanged; Week 3 gives a first look and points forward to Week 5’s optimality condition and Week 7’s calculus.

Week 4

- **4.23 (and 1.5, 5.31)** *boxed* γ is a risk-aversion weight in Week 1 but a risk budget here; the Week 5 slides call the penalised form “Week 4’s”; the weights are x , not w ; and the slide caps the variance although γ bounds $\|Fx\|_2$. *In the notes:* a slidenote records both roles of γ (Week 5 flags its attribution); the text says that γ caps the volatility and explains x versus w .
- **4.25 (with 4.24)** *boxed* The slide says the labels are b_i , not y_i , “because b is the SVM intercept”; that clash explains the y_i of the SVM, whereas logistic regression writes b_i because (a_i, b_i) is the unit-wide data convention. *In the notes:* the text gives the correct reason for both symbols, and a short slidenote says that the slide states it the other way round.
- **4.31** *boxed* The MNL frame never states the sign of β , although concavity of $R(s) = \frac{1}{\beta}[\dots]$ needs $\beta > 0$ (for $\beta < 0$ it is convex), as does the fall of a product’s share when its price rises. *In the notes:* $\beta > 0$ (price sensitivity) and α_i (attractiveness) are stated with the demand model, and a slidenote explains why the sign matters.
- **4.12 and 4.47 vs 4.29 and 4.33** *boxed* Frames 4.12 and 4.47 count four families, GP the fourth, in “exactly” the MOSEK Cookbook’s order, whereas 4.29 and 4.33 (and the Cookbook) put GP in the exponential-cone class. *In the notes:* three solver families, refined into a four-rung ladder; GP is a modelling class that lands on the exponential-cone rung after $y = \log x$; a slidenote states the discrepancy, and the first exercise keeps “GP” as an answer.
- **4.39** *boxed* The atoms table gives the single call `rso.norm(u)` for both $\|u\|_2 \leq t$ and $\|u\|_1 \leq t$; that call is the Euclidean norm, so copying it for an ℓ_1 constraint builds a different model. *In the notes:* the table has separate rows for `rso.norm(u)`, `rso.norm(u, 1)` and `rso.norm(u, np.inf)` (and $Q \succeq 0$ for `rso.quad`), and a slidenote explains the difference.
- **4.12, 4.21–4.24, 4.27, 4.32–4.33, 4.40** “QP”, “SOCP” and “exponential cone” are family names that students must classify into and whose solver counts they must read, but they are never defined. *In the notes:* short definitions of the QP (with $Q \succeq 0$), the SOCP and the exponential cone; the softplus epigraph becomes two exponential-cone constraints and a linear one, and $\|u\|_2^2 \leq t \iff \|(2u, t-1)\|_2 \leq t+1$ explains why `sumsq` gives an SOC constraint.
- **4.9 vs 1.12** The constraints are $f_i(x) \leq 0$ and $a_j^\top x = b_j$ here (and in Week 5), but $g_i(x) \leq 0$ and $h_j(x) = 0$ in Week 1. *In the notes:* the deck’s f_i are kept, with one sentence linking them to Week 1 and noting that Week 5 stacks the equalities as $Ax = b$; the text also says why equalities must be affine (a circle counter-example) and what the standard form forbids.
- **4.10** The slide writes the feasible set as an intersection of sublevel sets and affine sets only, leaving out the domain restrictions that 4.9 makes part of the model. *In the notes:* the proof that the feasible set is convex also intersects with the common domain of the f_i , which is convex because the domain of a convex function is.
- **4.8** “Have a unique minimiser” is right for positive-definite quadratics, but strict convexity alone gives uniqueness, not existence, no feasible set is named, and the graphic’s two marked points are at different heights. *In the notes:* the proposition says “at most one minimiser”, recalling Week 3’s midpoint argument; a quadratic with $Q \succ 0$ has exactly one minimiser over $\mathbb{R}^d$ and over any non-empty closed convex set, e^x has none, and the graphic is not reproduced.
- **4.11 and 4.15** The epigraph display appears twice (with f_0 , then with f) and without a feasible set. *In the notes:* it is stated once as a proposition, with $x \in \mathcal{X}$ on both sides, a proof and the tightness statement $t^* = f_0(x^*)$, and is recalled at the start of the reformulation patterns.
- **4.17 (with 4.16, 4.18)** The slide writes $\|r\|_1 = \sum_{i=1}^m |r_i|$, although m counts affine pieces on 4.16 and constraints on 4.9, and gives only the ingredients of the ℓ_1 model, not the LP. *In the*

notes: $A \in \mathbb{R}^{N \times d}$, $b \in \mathbb{R}^N$ throughout, m counts only the pieces of pattern (i), the ℓ_1 LP is written out, and fitting $b = (1, 2, 6)$ gives median 2, mean 3, mid-range 3.5.

- **4.19** The transportation LP does not state its feasibility condition $\sum_i s_i \geq \sum_j d_j$, and m , n and d_j overload the standard-form symbols. *In the notes:* the condition is stated with the feasible plan $x_{ij} = s_i d_j / \sum_k s_k$, the numbers of variables and constraints are given, and one parenthetical sentence addresses the overloaded symbols.
- **4.20** The Chebyshev-centre derivation needs $r \geq 0$, which the LP omits (an empty polyhedron then returns a negative radius instead of an infeasible status), and “the” centre need not be unique, even in the deck’s own figure. *In the notes:* the model is kept; the prose states the assumption and recommends adding $r \geq 0$, and a hand-solved example on the deck’s polygon gives $r^* = 1.25$ with every centre $(\xi, 1.75)$, $1.85 \leq \xi \leq 2.68$, optimal.
- **4.29** The posynomial is written in $x_1, \dots, x_n$ (against the course-wide $x \in \mathbb{R}^d$), and a_k , c_k and the area A reuse symbols with other meanings. *In the notes:* the deck’s symbols are kept; one sentence says what n , c_k and a_k are and that they are not the LP’s c or a_i , and the box example says that A is a number, not a matrix.
- **4.31 (and 4.51 B5; 4.9, 4.19, 4.34)** In the MNL frame p and s are prices and market shares, although p counts equality constraints on 4.9 and s denotes supplies on 4.19 and a slack on 4.34; tutorial B5 renames s_0 to t . *In the notes:* the text says once that p and s are prices and shares here; concavity is argued with $\phi(t) = (1 - t) \log t$ and s_0 substituted, as in B5, and the closed form of s_0 is added.
- **4.35** The slide substitutes $y = xt$, $t = 1/(c^\top x + d)$ in a ratio of affine functions and says only that the constraints “carry over”; recovering x needs $t > 0$, and the topic is called enrichment only. *In the notes:* $Gx \leq h$ becomes $Gy \leq ht$, $x = y/t$ when $t^* > 0$, the new objective and constraints are shown to be perspectives, and the enrichment comment is omitted.
- **4.40** The slide says RSOME’s Gurobi interface drops exponential-cone constraints “printing only the warning”; in fact a second warning follows and the error comes later. *In the notes:* the verified behaviour of the logistic listing (status 4 for the unbounded stripped model, then `No solution available`); the table’s “Comment” column becomes “Examples in this week”.
- **4.41 / 4.44** The “independent NumPy check” tests only $Ax \leq b$, not $Cx = e$ or $x \geq 0$, yet 4.44 presents it as the check the decisions must pass. *In the notes:* the listing appears with its real output; the prose adds that a complete check also tests $Cx = e$ and $x \geq 0$ (both hold) and explains the fourth inequality row of `do_math()` (the objective’s epigraph row).
- **4.42 / 4.43** The two listings are fragments that need `d`, `N`, `A`, `b`, `y` and `lam`, and 4.43 stores the labels b_i in a variable `y`. *In the notes:* one merged listing with a shared import block, code lines otherwise verbatim; the prose states the required data, that `y` holds the labels b_i , and the verified `do_math()` counts (one SOC; $2N$ ExpCone and one SOC).
- **4.45 (with 4.5, 4.44)** The slides speak of “infeasible” and “unbounded” statuses, but RSOME prints the raw status: Gurobi’s code 4 (“infeasible or unbounded”) for both, MOSEK a word such as `Undefined`. *In the notes:* one sentence says what is actually printed and that the modeller must settle the diagnosis, as in the two-asset example.
- **4.50 A2–A3, 4.51 B1–B2** Packaging of the tutorial questions. *In the notes:* A2 and A3 form one exercise, A2 also asking for the data c , A , b of the LP template; B1 and B2 form one exercise in set-builder form; the rest are restated without solutions; all exercises are titled; the further-practice pointers (4.19, 4.49) sit in the transportation paragraph and the summary.

Week 5

- **5.4, 5.31 boxed** The penalised model $\max \mu^\top w - \gamma w^\top \Sigma w$ subject to $\mathbf{1}^\top w = 1, w \geq 0$ is called “Week 4’s”; it is Week 1’s, with γ a risk-aversion weight, whereas Week 4 uses γ to cap the standard deviation. *In the notes:* the model is attributed to Week 1, Week 4’s risk-budget form is named as different, and the deck’s KKT system is kept.
- **5.3, 5.21, 5.23, 5.40 (vs 5.22, 5.28) boxed** “Dual feasible” is never defined and only $\lambda \geq 0$ is displayed as the dual constraint, so read literally $\lambda = 0$ is always dual feasible and “primal unbounded $\Rightarrow$ dual infeasible” could never apply. *In the notes:* dual feasibility means $\lambda \geq 0$ and $g(\lambda, \nu) > -\infty$; the KKT block keeps the slide’s $\lambda^* \geq 0$.
- **5.34–5.39 vs 5.3, 5.20, 5.40 boxed** The general theory uses a primal minimisation, but the LP part uses a maximisation without saying that the bounds change sides, and 5.34 joins primal and dual by “ $\Longleftrightarrow$ ” as if they were equivalent. *In the notes:* the pair is shown side by side, LP weak duality $c^\top x \leq \lambda^\top A x \leq b^\top \lambda$ is stated, and the reversal is explained.
- **5.34 (and tutorial C1, 5.46)** The dual constraint $A^\top \lambda \geq c$ is asserted without derivation, although the tutorial asks students to derive an LP dual. *In the notes:* the dual is derived from the Lagrangian of the minimisation form, $L = -b^\top \lambda + (A^\top \lambda - c - s)^\top x$, whose infimum over x is finite only if $A^\top \lambda - c = s \geq 0$.
- **5.8 (vs Week 3 deck)** The subgradient is presented as new, but the Week 3 deck already defines the subgradient and the subdifferential. *In the notes:* the definition is recalled from Week 3 and the subgradient inequality is restated.
- **5.7, 5.10 (and Week 3 frame 3.8)** Uniqueness under strict convexity is only asserted, “unique” presupposes existence, and the “ $\Rightarrow$ ” direction of the gradient test is not argued. *In the notes:* the theorem says “at most one minimiser”, from Week 4’s strict-convexity result with $\mathcal{X} = \mathbb{R}^d$; a descent argument proves “ $\Rightarrow$ ”, and the convex e^x has no minimiser.
- **5.13** The equality-constrained intuition uses $\nabla f(x^*)$ without assuming that f is differentiable, and gives no argument for “orthogonal to every feasible direction, hence parallel to a ”. *In the notes:* differentiability is assumed explicitly and a two-line orthogonal-decomposition argument is given.
- **5.17, 5.30** The sign condition $\lambda \geq 0$ is read off the picture only, and 5.30 writes ∇f and one λ although there are two inequality multipliers and the theory writes f_0 . *In the notes:* $\lambda \geq 0$ is justified with the feasible direction $v = (-1, 1)$, where $\nabla f(x^*)^\top v = \lambda = 2$; the KKT example uses $f_0, \lambda_1, \lambda_2$ and shows the two failing active-set guesses.
- **5.14–5.15** The multiplier of the projection example is never printed. *In the notes:* $\nu^* = 2$ is computed, and the dual function $g(\nu) = 2\nu - \frac{1}{2}\nu^2$ gives $d^* = 2 = p^*$.
- **5.20–5.22** The deck orders dual function, weak duality, dual problem, and never names the dual optimal value. *In the notes:* the dual function and the dual problem are defined together, introducing d^* , before weak duality, which then states $d^* \leq p^*$; the duality gap $p^* - d^*$ is named, and weak duality is noted to need no convexity.
- **5.23** The generic certificate card shows $f_0(x) = 9$ and $g(\lambda, \nu) = 9$, anticipating the LP, whose values in minimisation form would be -9 . *In the notes:* the card is not reproduced; the certificate is illustrated by the projection example ($2 = 2$) and the segment problem ($5 = 5$), and by $9 = 9$ only for the max/min LP pair.
- **5.24–5.25** 5.24 states only a zero duality gap under Slater’s condition, while the chain on 5.25 assumes that a dual optimal pair exists; strong duality and Slater’s condition are described, not defined. *In the notes:* both are defined, the theorem states that the dual optimum is

attained when the common value is finite, and $\min x$ subject to $x^2 \leq 0$ has neither a Slater point nor a KKT multiplier.

- **5.4, 5.33, 5.46** The ℓ_1 -regularised least-squares model is called the workhorse of Weeks 7 and 9 and its componentwise condition “derived in Week 7”, but the Week 7 deck has no lasso, only the subdifferential of the ℓ_1 norm. *In the notes:* the model is attributed to Week 9, with the subdifferential calculus supplied by Week 7.
- **5.33, 5.46, 5.49 (C2)** The tutorial refers to the componentwise form of $\lambda \partial \|\beta\|_1$ as given in the lecture, but the lecture never displays it, and the sum rule is used silently. *In the notes:* $\partial|x|$ is derived, the lasso condition is written coordinate by coordinate, and the sum and separability rules are taken on trust until Week 7.
- **5.38** The rule $\partial p^*/\partial u_i = -\lambda_i^*$ uses u_i , which is never defined. *In the notes:* the perturbed problem with $f_i(x) \leq u_i$ and value $p^*(u)$ is defined first, and the formula for the maximisation LP is justified by weak and strong duality: the prediction is never too low and is exact while λ^* stays optimal.
- **5.38** Calling λ_i^* the shadow price presumes a unique dual solution; at a breakpoint of $p^*(b)$ the optimal prices are not unique and the one-sided slopes differ. *In the notes:* the caveat follows the definition of the shadow price, and the perturbation example shows it at $b_1 = 5$, where $(1, 1)$ and $(0, 2)$ are both optimal.
- **5.39** “Active basis” is simplex vocabulary that the course never introduces. *In the notes:* the text says that the same constraints stay active, or that the active set has not changed.
- **5.38 (figure)** The shadow-price figure is a schematic two-piece curve, not the value function of its LP: the kink at $b_1 = 2.5$ is missing and the last piece still rises. *In the notes:* the figure draws the true $p^*(b_1)$, with slopes 3, 1, 0, breakpoints 2.5 and 5, the point $(4, 9)$ and the dashed linear prediction; the perturbation example gives the three pieces.
- **5.10 vs 5.33** Least squares is $\|Ax - b\|_2^2$ on 5.10 and $\frac{1}{2} \|A\beta - b\|_2^2$ on 5.33, each consistent with its own gradient. *In the notes:* both are kept as on the slides, and one sentence says that the factor changes neither the minimiser nor the normal equations.
- **5.8/5.20, 5.10/5.18/5.34, 5.33, 5.14/5.34** Several symbols are overloaded: g (subgradient, then dual function, written $q(\lambda)$ in Weeks 12–13), A and b (regression, equality-constraint and LP data), λ (multiplier and lasso penalty weight) and c (centre point and LP objective). *In the notes:* the deck’s symbols are kept, and the reader is told at each change of meaning.
- **5.43** The slide’s comment mentions only the optimal values; with `grb_solver` the printed status is Gurobi’s integer code 2, not a word. *In the notes:* the listing is reproduced, and the text reports the actual output for Gurobi (2 2, then 9.0 9.0) and MOSEK (**Optimal**) and the rerun with $b_1 = 4.2$ (value 9.2, $x = (0.8, 3.4)$).
- **5.47–5.49 (tutorial A2, A3, B1, B2, C1)** These tutorial questions repeat data or derivations that the lecture works in full. *In the notes:* the deck’s versions are worked examples in the body, and the exercises use variants that point back to them: $x_1^2 + x_1x_2 + x_2^2 - 3x_1 - 3x_2 + 4$, the ridge objective $\|Ax - b\|_2^2 + \rho \|x\|_2^2$, the line $x_1 + 2x_2 = 2$, the centres $(1, 4)$ and $(2, 1)$, and Week 1’s LP with a ranging question.
- **5.8, 5.15, 5.17 (figures), 5.35, 5.39 (tables)** Cosmetic points: the $|x|$ picture draws support lines of slopes 0.6, 0.1, -0.4 rather than the flat one that certifies the minimum, the equality-KKT graphic has a stray filled rectangle, labels are crowded, and two small tables are used. *In the notes:* the figures are redrawn with slopes 0.6, 0, -0.4 , without the rectangle and with shorter labels, and both tables become prose.

Week 6

- **6.41 boxed** The experiment is placed “on 6.30’s elongated quadratic” with no start; its numbers (24 uphill steps, peak $\approx 50\times$ the start, $9.5\times$ below the safe run) hold for $x^0 = (2.4, 0.5)$, not 6.30’s $(2.4, 0.35)$, where the first long step goes downhill and the peak is $\approx 34\times$. *In the notes:* the example states $x^0 = (2.4, 0.5)$.
- **6.29** “Each extra digit costs ten times more iterations” describes the C/k bound, not the plotted run, which gains under one digit in 400 iterations; the run labelled “convex only” is strongly convex with a tiny floor ($L/\mu \approx 6.6 \times 10^3$). *In the notes:* the ten-fold reading is attributed to the $O(1/k)$ guarantee, and the caption explains the label.
- **6.17 (also 6.9)** “Strongly convex” is used ten frames before 6.27 defines it, and no earlier deck defines the term. *In the notes:* at the family $Q = A^\top A + \mu I$ the text says that the directional curvature is at least μ along every unit direction and points forward to the definition of strong convexity.
- **6.3, 6.10–6.11, 6.31** The letter d is both the dimension of x and a search direction (d^k , a bare d , and d_{scaled} next to a “ $d \times d$ system”), contrary to the registry’s claim that no frame carries both. *In the notes:* both uses are kept and explained at first use, “a system of d nonlinear equations” becomes “as many equations as there are variables”, and the two meanings never share a sentence.
- **6.22** The Armijo parameter α and the shrink factor β are unrecorded overloads: from Week 7 on, α_k is the step size and β a vector of regression coefficients or a step-schedule or momentum constant, and the registry’s list for β omits Week 6. *In the notes:* the deck’s letters are kept, with a one-sentence warning after their first use.
- **6.18, 6.26** argmin is used without a gloss at its first use in the course, and the rate headline uses the undefined symbol “ $\lesssim$ ”. *In the notes:* the definition of exact line search glosses argmin as the set of minimisers (hence “ $\in$ ”); the rate is stated with the explicit constant $\|x^0 - x^*\|_2^2 / (2tk)$ and a definition of $O(\cdot)$, without “ $\lesssim$ ”.
- **6.26** The rate headline omits $t \leq 1/L$, the existence of a minimiser and the constant; the proof outline uses the first-order inequality “at x^* ” (vacuous; the tangent at x^k is needed) and presumes that the distance to x^* never increases; “sublinear” is applied to the convergence, not the bound. *In the notes:* a full statement, the three-step telescoping proof, and “sublinear” for the guarantee.
- **6.7, 6.9, 6.11, 6.19, 6.20, 6.27, 6.31** Implicit hypotheses: f twice continuously differentiable (Taylor’s formula, Hessian forms of L -smoothness and strong convexity), $\nabla f(x) \neq 0$ (steepest descent), $(g^k)^\top Q g^k > 0$ (exact line search), differentiability (lower quadratic bound) and $\nabla^2 f(x^k) \succ 0$ (Newton step). *In the notes:* all are stated.
- **6.20** The function under the “upper quadratic model” is a quartic with unbounded curvature, hence not L -smooth; the drawn bound holds only on the plotted window, which the slide does not say. *In the notes:* the picture is kept, its caption says that the bound is shown over the range drawn, and the text notes that x^4 is not L -smooth for any L .
- **6.37** “A point the model already classifies confidently pulls almost not at all” holds only for confident *and correct* classification: a confidently misclassified point ($b_i a_i^\top w \ll 0$) has weight $1 - \sigma_i \approx 1$. *In the notes:* the logistic-loss example says “confidently and correctly” and adds that a confidently misclassified observation pulls with almost full weight.
- **6.12, 6.31 (TikZ)** On 6.12 the dashed “linear model” has slope -0.77 , not the tangent slope -0.7 of $0.35(z - 2)^2 + 0.6$ at $z = 1$ (it is a secant); on 6.31 the $-\nabla f$ arrow is 12°

off vertical instead of 4° . *In the notes:* the 6.12 picture is replaced by the computation $m_k(x^k - s\nabla f(x^k)) = f(x^k) - s \|\nabla f(x^k)\|_2^2$; the 6.31 sketch is redrawn.

Week 7

- **7.35 boxed** The lines labelled least squares ($0.95x + 0.05$) and LAD ($0.75x + 0.3$) are not the fits of the six plotted points; the drawn “LAD” line lies below all six (total absolute deviation 4.35). *In the notes:* the figure shows the computed fits $-0.36 + 1.146a$ and $b = a$ (deviation 1.5, certified by hand), and a box calls the slide’s lines schematic.
- **7.31 boxed** $\Delta_k \leq \nu/(\gamma + k)$ is stated with ν undefined and γ unrestricted, and it can fail: for $f(x) = \frac{1}{2}x^2$, $\beta = 5$, $\gamma = 1$, $x^0 = 1$ one has $\nu = 100$ but $\Delta_2 = 36 > \nu/3$. *In the notes:* the theorem defines $\nu = \max\{\beta^2 G^2/(2\mu\beta - 1), \gamma\Delta_0\}$, adds $\gamma \geq 2\mu\beta$ and proves the bound by induction.
- **7.31–7.32 boxed** The averaging claim leaves $\bar{x}^k$ and the step size undefined, and the distance recursion alone gives only $O(1/\sqrt{k})$ in objective values, so it is not “the same recursion”. *In the notes:* part (c) takes $\alpha_k = 2/(\mu(k+2))$ and weights $i+1$, and proves $f(\bar{x}^k) - f^* \leq 2G^2/(\mu(k+2))$ from a recursion that keeps the objective gap.
- **7.31, 7.40** “Same steps, same G ” is ambiguous: the update is unchanged, but the schedule becomes $\beta/(\gamma + k)$ with $2\mu\beta > 1$, which needs a lower estimate of μ . *In the notes:* the text says that the update is the one analysed before but the step-size rule is not.
- **7.30 (vs 7.39)** “This rate cannot be improved by any method that uses only subgradients” is a worst-case, dimension-free statement about the class of convex G -Lipschitz functions, not about each instance, and is not proved. *In the notes:* a remark states it in that form without proof and cites the Polyak runs as instance-specific speed.
- **7.20** The example for “ $-g$ need not be a descent direction” sits at the minimiser $x = 0$, where no direction descends; the increase on 7.21 comes from the step length. *In the notes:* the observation is kept, and for $f(x) = |x_1| + 2|x_2|$ at $(1, 0)$ the subgradient $g = (1, 2)$ gives $f(x - \alpha g) > f(x)$ for every $\alpha > 0$.
- **7.27** With sums over $i = 0, \dots, k$, K iterations bound f_{best}^{K-1} , not f_{best}^K ; the balanced value $RG/\sqrt{K}$ is exact and minimal, and the bound needs $R \geq \|x^0 - x^*\|_2$. *In the notes:* the corollary is stated so, notes that it also bounds f_{best}^K , and adds $K \geq (RG/\varepsilon)^2$.
- **7.25** “ G -Lipschitz” is used undefined (Week 6 defined Lipschitz continuity only for the gradient), and “equivalently” is global while the assumption concerns the region visited. *In the notes:* $|f(x) - f(y)| \leq G \|x - y\|_2$ is defined, its link with bounded subgradients is explained, and a remark restricts G to the region visited.
- **7.11** Only the sum rule is justified; the affine-composition and active-max rules are left to the tutorial, their dimensions are not stated, and the equality claim is unexplained. *In the notes:* each inclusion is proved in one line, the dimensions ($A \in \mathbb{R}^{n \times d}$, $b \in \mathbb{R}^n$) are given, and equality for finite-valued functions is stated without proof.
- **7.12 (also W5.33, W3.30)** The ℓ_1 subdifferential is stated, not derived, though Week 5 promised a derivation; Weeks 3 and 5 also assign the lasso and Huber-type losses, absent from the deck, to Week 7. *In the notes:* the formula is derived, Week 9’s proximal treatment is previewed, and no lasso or Huber material is added.
- **7.13** The cases display writes $s_i = [-1, 1]$ for a zero residual, meaning $s_i \in [-1, 1]$. *In the notes:* the formula says “any number in $[-1, 1]$ ” and notes that $\text{sign}(0) = 0$ is a valid choice, as in the code.

- **7.10 (graphic)** The arrow labelled $(-1, -1)$ stops at $(-0.85, -0.85)$, and the level sets are nested triangles only for positive levels. *In the notes:* the arrow reaches $(-1, -1)$, and the text describes the level set for $c > 0$ as the boundary of the triangle with vertices (c, c) , $(c, -2c)$, $(-2c, c)$.
- **7.38 (also 7.45)** The minimal loop tracks the best objective value but returns the last iterate, not the best point. *In the notes:* the algorithm returns the best point, and the text before the listing says how the loop can store it (copy the iterate whenever the best value improves).
- **7.39** “Diminishing steps keep improving” is true, but after the 600 iterations shown the diminishing run (gap 1.10×10^{-2}) is marginally behind the fixed step (1.01×10^{-2}); the advantage appears only on longer runs. *In the notes:* both gaps are reported, with the fixed step’s last improvement (iteration 208) and 20 000-iteration gaps from runs marked as not listed.
- **7.33–7.39 (order)** The verdict frame (7.34) sits between stopping (7.33) and the robust-regression application (7.35–7.39). *In the notes:* all the material is kept, and the verdict closes the subsection after the LAD experiments, so that it can draw on them.
- **7.42 (Q3)** The exercise refers to “a point supplied in class”, so it is incomplete for self-study. *In the notes:* the exercise uses $x = (1, 0)$ and $x = (2, 0)$ and adds the convex-hull check of $x^* = (0.2, 0.6)$, $f^* = 1.4$.
- **7.43 (Q4)** Three iterations from $x^0 = 2$ with $\alpha = 0.6$ (2, 1.4, 0.8, 0.2) do not yet show the oscillation asked about; the cycle 0.2, -0.4 starts at the fourth. *In the notes:* the exercise asks for six iterations and for f_{best}^k .
- **7.42 (Q2)** The median question is worked in full in the body, so the exercise as stated would duplicate the text. *In the notes:* the exercise keeps the proof task and adds the even- N data $c = (1, 2, 4, 7)$ (set of minimisers and optimal value).
- **7.17, 7.19 (also 7.7, 7.31, 7.36)** Deliberate overloads: the step size is α_k , not Week 6’s t_k ; t is also the epigraph height and the LP auxiliary vector; β is both the LAD coefficients and a schedule constant. *In the notes:* the deck’s symbols are kept and announced at first use; the LAD model notes that Week 6 wrote the coefficients as x .

Week 8

- **8.9 (used at 8.10, 8.13, 8.27) boxed** The slide states $\partial I_C(x) = N_C(x)$ for every x , but for $x \notin C$ the set contains 0 while $\partial I_C(x) = \emptyset$, so $0 \in \nabla f(x) + N_C(x)$ can hold at an infeasible point ($C = \mathbb{R}_+$, $f(x) = \frac{1}{2}(x+2)^2$, $x = -1$). *In the notes:* $N_C(x)$ is defined for $x \in C$ and is $\emptyset$ elsewhere.
- **8.2, 8.10, 8.15–8.19 boxed** No hypotheses on C are stated. Without closedness a projection may not exist ($C = (0, 1)$, $y = 2$); without convexity it need not be unique, non-expansiveness fails, and so does the “ $\Rightarrow$ ” half of 8.10 ($f(x) = (x - 0.4)^2$ on $\{-1, 1\}$). *In the notes:* C is nonempty, closed and convex throughout; a box gives the counterexamples.
- **8.2, 8.19, 8.31–8.32, 8.38 boxed** “The Week 6 convergence logic is recycled”, but Week 6 bounds $f(x^k) - f^*$, whereas this rate bounds $\|x^k - x^*\|_2$ and rests on a gradient-step contraction that Week 6 never states. *In the notes:* the contraction is a lemma marked as quoted, and a box says what is inherited and what is new.
- **8.8, 8.10** “The Week 3 / Week 7 first-order rule applies directly” skips two steps: Week 7 defines subgradients for finite-valued functions, and for the extended-valued I_C its sum rule

gives only an inclusion, hence only the “ $\Leftarrow$ ” half of 8.10. *In the notes:* the subgradient inequality is extended to I_C on C , and the theorem is proved directly.

- **8.7 (and 8.37)** The KKT recap credits Week 5 but writes the objective f and constraints g_i , whereas Week 5 wrote f_0, f_i and used $g(\lambda, \nu)$ for the dual function; on 8.37 g^k is a subgradient. *In the notes:* the deck’s symbols are kept, with one sentence mapping them to Week 5’s and noting the later meaning of g^k .
- **8.20, 8.41 (with 8.4, 8.19)** The box is written $[\ell, u]^d$ with scalar bounds although its projector uses ℓ_i, u_i ; on 8.19 u is an arbitrary point and p clashes with the number of equality constraints. *In the notes:* the box is $\{x : \ell \leq x \leq u\}$ with $\ell, u \in \mathbb{R}^d$; the text says that u, v and p in the proof are points; the sorted copy of y is called s .
- **8.32** “Same condition-number story as Week 6” uses a name for L/μ that the course introduces only in Week 10. *In the notes:* the iteration count is written in terms of L/μ , “the condition number of Week 10”.
- **8.32, 8.33** The rate is stated for the step $\alpha = 1/L$ only, while 8.33 recommends a safe fixed step $\alpha \leq 1/L$ without saying what a smaller step guarantees. *In the notes:* the theorem is stated for $\alpha = 1/L$, and the text adds that $\alpha < 1/L$ gives the factor $1 - \alpha\mu$, so the largest safe step gives the best guarantee.
- **8.33 (options on 8.30)** Backtracking is offered as one of “the same basic ideas from Week 6” with no test, but Week 6’s Armijo test asks for a decrease proportional to $\|\nabla f(x^k)\|_2^2$, unattainable near a solution on the boundary. *In the notes:* an upper-model test at the projected point is stated, shown to hold for $\alpha \leq 1/L$ and to give a decrease of $\frac{\alpha}{2} \|G_\alpha(x^k)\|_2^2$.
- **8.28, 8.31** Two objects have almost the same name: the projected-gradient *mapping* G_α (a residual) and the projected-gradient *map* T_α (the update). *In the notes:* both names are kept; where T_α is introduced the text says it is the update, not the residual, and gives $G_\alpha(x) = (x - T_\alpha(x))/\alpha$.
- **8.17, 8.19 (figures)** The projection-theorem figure draws $z - y$ although the text calls $y - z$ the outward normal; in the non-expansiveness figure the arrows labelled $u - v$ and $p - q$ show $v - u$ and $q - p$. *In the notes:* both are redrawn: arrows $y - z$ and $x - z$ with the angle between them, and segments labelled by their lengths.
- **8.36 (notebook run)** The NNLS run has no active constraint at its solution: with the notebook’s seed it is the unconstrained least-squares solution (smallest entry 0.1303), and clipping occurs only in the first 22 of 490 iterations. *In the notes:* the run is reported as printed, with a caution; the box and tiny NNLS examples have active constraints.
- **8.36** SciPy’s `nnls` is said to match the projected-gradient solution “to 1e-12”; the measured maximum coordinate difference is 7.3×10^{-12} . *In the notes:* the agreement is given as about 7×10^{-12} .
- **8.35** “The objective is smooth and familiar from Week 4”: the simplex quadratic is the mean-variance portfolio of Weeks 1 and 5 re-lettered; Week 4’s portfolio is a risk-budget SOCP. *In the notes:* the model is tied to the long-only portfolio of Weeks 1 and 5 ($x = w$, $c = \mu$, $\Sigma \rightarrow 2\gamma\Sigma$) and to Week 4’s convex-quadratic model family.
- **8.12–8.13** The component notation $\nabla_i f$ is used without definition, its first use in the course. *In the notes:* it is defined as the i th partial derivative before the orthant example.
- **8.21** The ball projector $\min\{1, r/\|y\|_2\}y$ is undefined at $y = 0$. *In the notes:* the formula is read as y when $y = 0$.

- **8.23–8.24 (TQ2, TQ5)** No method is given for computing the simplex threshold τ , although the tutorial questions require it. *In the notes:* a proposition derives the projection from the KKT conditions and proves τ unique; a sorting algorithm ($O(d \log d)$) and a worked three-vector example follow.
- **8.36 (top figure)** The box example in the figure (the box $[0, 2] \times [0, 0.6]$, $x^0 = (1.9, 0.05)$, $\alpha = 0.25$) is specified only in the notebook. *In the notes:* it is a fully worked example with every iterate, the residual and the optimality certificate, and the caption states what is measured.
- **8.14, 8.23** Δ denotes the simplex, whereas Δ_k is the squared distance to the solution in Weeks 7 and 11, an overload not recorded in the registry. *In the notes:* one sentence at first use says that the plain Δ is unrelated to Δ_k .
- **8.14** The normal-cone form is said to “compress” the simplex KKT system, but $N_\Delta(x)$ is never given. *In the notes:* $N_\Delta(x) = \{\nu \mathbf{1} - \lambda : \nu \in \mathbb{R}, \lambda \geq 0, \lambda_i x_i = 0 \text{ for all } i\}$ is stated and justified by the KKT theorem for a linear objective over the simplex.
- **8.17** The projection theorem is stated without proof. *In the notes:* it is proved in one paragraph by applying the constrained-optimality theorem to $f(z) = \frac{1}{2} \|z - y\|_2^2$ (an addition, not a change).
- **8.32, 8.36** The slide refers to a proof “via eigenvalues” and the code computes L and μ with `eigvalsh`, but eigenvalues are introduced only in Week 10. *In the notes:* L and μ for least squares are the largest and smallest directional curvatures $\|Au\|_2^2$ over unit vectors u , and “eigenvalues” is only a forward pointer to Week 10.
- **8.37** “The same step-size rules and best-iterate bounds from Week 7 carry over” is asserted without a reason. *In the notes:* $x^* = \Pi_C(x^*)$ and non-expansiveness reduce the projected step to Week 7’s one-step inequality, read with the constrained x^* and f^* ; the text adds that the residual stopping test does not carry over.

Week 9

- **9.11 (used as an equivalence on 9.23, 9.30 and tutorial Q4)** Composite optimality is shown as a one-way implication without proof and tied to Week 7’s $0 \in \partial F(x^*)$ through a sum rule stated only for finite-valued functions, although h may be I_C ; the deck then uses it in both directions. *In the notes:* the theorem is an equivalence with a direct proof that needs no sum rule and covers extended-valued h .
- **9.31 boxed** “Backtracking on the smooth part g exactly as in Week 6”, read literally, tests g at the gradient-step point, not at the prox point x^+ taken, and can accept steps that increase F . *In the notes:* the upper-model test on g at x^+ , Week 8’s projected-gradient test, which passes for every $\alpha \leq 1/L$; the shrink factor is η , as Week 6’s α, β are taken.
- **9.31** $L = \lambda_{\max}(A^\top A)$ uses eigenvalues before Week 10 introduces them and gives λ , the regularisation strength, a second meaning. *In the notes:* $L = \max_{\|u\|_2=1} \|Au\|_2^2$, glossed as the largest eigenvalue of $A^\top A$ with a pointer to Week 10 and a NumPy one-liner; the symbol $\lambda_{\max}$ is avoided.
- **9.35 boxed** The contraction $\left\| (x - y) - \frac{1}{L}(\nabla g(x) - \nabla g(y)) \right\|_2 \leq (1 - \frac{\mu}{L}) \|x - y\|_2$ is called “the Week 6 gradient-step bound”, but Week 6 contains no such bound; it first appears, quoted without proof, in Week 8. *In the notes:* the proof cites Week 8’s lemma and works out the separable quadratic case.

- **9.34, 9.35 (with 9.8, 9.10)** Linear convergence is promised for “ridge-type penalties”, but the argument needs a μ -strongly convex g , whereas the deck’s own split puts a ridge term in h , where it gives only the factor 1. *In the notes:* a remark counts the ridge term in g (elastic-net split); the measured run is strongly convex through g itself.
- **9.28, 9.43** The one-step ISTA example gives only the gradient-step output $(2.1, -0.3, 0.8)$ and $\alpha\lambda = 1$, so $\alpha \leq 1/L$, G_α and F cannot be checked; tutorial Q3 has no data. *In the notes:* the completion $A = I_3$, $b = (3.2, -0.6, 1.6)$, $\lambda = 2$, $\alpha = \frac{1}{2}$, $\beta^k = (1, 0, 0)$ reproduces the slide’s numbers; the exercises use a second verified instance.
- **9.16** Subgradients of h are written g_p, g_q , although g is the smooth part all week and the deck’s own subgradient symbol is s^k . *In the notes:* the proof writes s_p, s_q , and the introduction says once that g is the smooth part and that a subgradient is called s .
- **9.4 vs 9.19; 9.5, 9.32; 9.16, 9.17, 9.21, 9.38** τ is both the complexity budget and the threshold $\alpha\lambda$; s^k is not recorded in the registry; u has four meanings (a point, the box’s upper bound, the trade, a split variable). *In the notes:* the budget τ is confined to one subsection and the change of meaning is announced; u is glossed at each use, and the split variables are declared unrelated to the points u, v .
- **9.5–9.7, 9.13, 9.16, 9.18, 9.22** F , the prox (used before its definition), the condition $v - p \in \alpha \partial h(p)$, hypotheses on h behind “= argmin”, soft-thresholding’s consistency conditions, $\Sigma, c, x^{\text{old}}$, “ISTA” and the group prox at $v = 0$ are used before or without definition. *In the notes:* each is defined or stated at first use.
- **9.16–9.17** The deck proves non-expansiveness before the box-projection example, which illustrates the earlier frame 9.14. *In the notes:* the box example directly follows the proposition that the prox of an indicator is the projection, and non-expansiveness comes after it; no content changes.
- **9.33** The frame is titled “Seeing it on real data”, but the instance is simulated (seeded normal draws, planted 6-sparse coefficients, Gaussian noise). *In the notes:* the example calls it simulated, states what is generated and measured, prints the generating code and names the companion notebook.
- **9.37** The ISTA listing records F after the update, so `hist[k]` is $F(x^{k+1})$, whereas the notebook, and hence the figure, records $F(x^k)$. *In the notes:* the listing is kept verbatim (it prints 117 1.1139290255035426 12) and the text states the offset.
- **9.38** The comment “2 = optimal” is Gurobi’s status code (MOSEK returns `Optimal`), and the function `F` in the last print is defined only in the notebook. *In the notes:* the listing defines `F` and glosses the comment; the text reports both solvers (1.11392922 with Gurobi, 1.11392903 with MOSEK) and that solver zeros are only approximate.
- **9.4** The number-line graphic suggests that each formulation owns one end of the λ scale, and “same trade-off” holds only from penalised to constrained. *In the notes:* both problems are displayed, a proposition proves the penalised-to-constrained direction, and the converse is stated under Slater’s condition with λ the shadow price.
- **9.36** The schematic regularisation path places its dashed guides at $\lambda = 0.95$ and 2.6, while the plotted paths reach zero at $\lambda = 1$ and $18/7$; the deck labels it a schematic. *In the notes:* the path is described in prose with the deck’s caveat, and a table reports measured values on the notebook instance (zero solution once $\lambda \geq \|A^\top b\|_\infty = 6.606$).
- **9.40 vs 9.41–9.45** The tutorial plan and questions disagree: the plan classifies four models but the question lists three, the penalised-versus-constrained activity has no question, and

the mapping activity has no data. *In the notes:* the first exercise adds the logistic-lasso, the second covers the two forms and the roles of λ and τ , and the fourth and fifth use a verified instance on which G_α can be evaluated.

- **REGISTRY running-examples table (affects 9.3, 9.8)** The registry lists the lasso in Week 7, but no deck before Week 9 contains it (only the Week 5 optimality preview); the registry and Weeks 10 and 12 write it in x , the Week 9 slides in β , the Week 9 code in x . *In the notes:* the lasso is introduced from scratch in β , the change of letter from x to β and to neighbouring weeks is stated once, and the code example says it writes x .
- **9.6, 9.27 (convention)** The deck writes the Week 6 update with α (Week 6 wrote t_k), and Week 6's backtracking constants α, β clash with this week's step size and coefficient vector. *In the notes:* a deliberate registry convention, not an error: the introduction announces α once, and the backtracking remark names the shrink factor η .
- **9.17, 9.22** The box example and the table of proximal operators write the lower bound of a box as $l \leq x \leq u$, whereas Week 8 writes $\ell \leq x \leq u$ for the same set and the same projector. *In the notes:* Week 9 writes ℓ , as Week 8 does.

Week 10

- **10.2 (vs 10.5, 10.8, 10.12) boxed** The zig-zag picture uses an exact line search; every other run and the mode analysis use $\alpha = 1/L$, under which gradient descent on this quadratic does not zig-zag but creeps along the flat axis by 0.96 per step. *In the notes:* each picture names its step rule; a box presents zig-zag and crawl as two faces of ill-conditioning.
- **10.14, 10.20, 10.24, 10.26 boxed** The accelerated rates are stated without hypotheses, and smoothness beyond C is not required although y^k can leave C . *In the notes:* the theorems assume convexity or strong convexity, L -smoothness on $\mathbb{R}^d$, a minimiser, $\alpha = 1/L$ and the momentum rule; a box points this out, and a remark gives an infeasible y^k .
- **10.7, 10.16, 10.24, 10.26 (and 10.20) boxed** The deck pairs $\beta_k = (\theta_k - 1)/\theta_{k+1}$ with $y^k = x^k + \beta_k(x^k - x^{k-1})$, whereas the notebook and the theorem quoted on 10.20 attach β_k to $x^{k+1} - x^k$, a one-step shift (both are $O(1/k^2)$). *In the notes:* the algorithms use the published form (β_{k-1} on $x^k - x^{k-1}$, $y^0 = x^0$) and its constants; a box explains the shift.
- **10.37 boxed** “FISTA keeps the same sparse structure as ISTA” is right about the model, but the plotted run shows nothing sparse: with $\lambda = 0.02$ against $\|A^\top b\|_\infty = 16.55$ all 60 coefficients are nonzero ($\lambda = 5$ would leave five). *In the notes:* a box says that the run illustrates speed, not sparsity, and the sparsity statement is made about the model.
- **10.9** “Nesterov is about three digits more accurate” holds at $k = 30$ (gap ratio 1.1×10^3), but the figure runs to $k = 70$, where the ratio is 5.4×10^8 . *In the notes:* the worked example and its table quote the iteration: about three digits at $k = 30$, more than eight at $k = 70$.
- **10.4 (vs 10.8–10.9)** “Tuned for a quadratic it is excellent”, but the plotted run uses $\alpha = 1/L$, $\beta = 4/9$, not Polyak's tuned $\alpha = 4/(\sqrt{L} + \sqrt{\mu})^2$, under which every mode contracts by $2/3$, beating Nesterov's method. *In the notes:* the text gives the plotted parameters and says the tuned pair is faster; an example gives both mode roots.
- **10.21, 10.33** 10.21 states the backtracking test with $g, h, \text{prox}_{\alpha h}$ inside the smooth-case act, 10.33 repeats it, and neither says that the estimate of L must be carried forward for the $O(1/k^2)$ guarantee. *In the notes:* the smooth test uses that act's letters with the carry-forward proviso; the composite test appears once, as Week 9's test at y^k .

- **10.11–10.15** The act is titled “why momentum works”, but its frames analyse only gradient descent mode by mode; no frame computes what momentum does to a mode. *In the notes:* the two-term mode recursion is added for the constant- β accelerated update and for the heavy ball, with the double root $0.8 = 1 - 1/\sqrt{\kappa}$ for $\kappa = 25$ against gradient descent’s 0.96.
- **10.34** The residual table writes the proximal-gradient mapping as $\tilde{G}_\alpha$, whereas Week 9 writes G_α for it, as Week 8 does for the projected-gradient mapping. *In the notes:* the deck’s tilde is kept, and one clause says that Week 9 writes G_α for both, which coincide when h is an indicator.
- **10.35** “Gradient descent decays like its $O(1/k)$ bound over this horizon”, but its gap ends about 20 times below that bound and decays faster (the instance is strongly convex). *In the notes:* gaps and bounds are compared at $k = 100$ (2.14 vs 18.9; 1.91×10^{-2} vs 0.742); the caption says the legend’s orders are guarantees, not fitted slopes.
- **10.7, 10.17, 10.22, 10.25, 10.35, 10.38, 10.42–10.45** Several terms are used without definition: “gradient oracle”, the acronyms AGD and APG, “FISTA” (never expanded), “ill-conditioned” (only “condition number” and “badly conditioned” are tied to κ) and “mode”. *In the notes:* each term is defined or glossed at its first use.
- **10.5 (figure)** In the right panel (“too much momentum oscillates”) the two arrows run at about 26° , along the flat axis of the 28° ellipses, so they show motion along the valley, not the oscillation across it that the label describes. *In the notes:* the arrows are redrawn along the steep axis, alternating in sign; caption and text are unchanged.
- **10.15** The count $\sqrt{\kappa} \ln(1/\varepsilon)$ drops the constant 2 of the bound $2(1 - 1/\sqrt{\kappa})^k (f(x^0) - f^*)$, worth $\sqrt{\kappa} \ln 2$ extra iterations (about 7 for $\kappa = 100$). *In the notes:* the count is stated up to that constant, the extra iterations are named, and the frame’s worked numbers are kept.
- **10.35–10.36** The benchmark frames show pictures but no numbers (data, conditioning, reference optimal values, comparison with the bounds). *In the notes:* the figures are reproduced with the data ($L = 132.1$, $\mu = 0.315$, $\kappa \approx 420$), the reference values ($f^* = 29.93$, 56.51 ; $F^* = 30.62715$, confirmed by MOSEK) and two gap values per curve, all recomputed.

Week 11

- **11.6 boxed** The full gradient is called “the case $s = N$ ” of a mini-batch of s independent draws, but then only about 63% of the records appear and the variance is still $\text{Var}(g_1)/N$; 11.7 leaves the scheme open. *In the notes:* the algorithm draws s independent indices; the full gradient is the limit $s \rightarrow \infty$, the case $s = N$ only without replacement.
- **11.16, 11.17, 11.26 (and 11.21) boxed** Measured floors are read as confirming $A_{k+1} \leq \rho A_k + \alpha^2 B^2$, but every run violates $\alpha < 2\mu/L_g^2 = 2.6 \cdot 10^{-5}$ ($\rho = 1.31, 1.08, 1.02 > 1$) and $B^2/(2\mu) = 115$ is 44 times the measured floor/ α . *In the notes:* the theorem is kept; a box says the runs confirm its shape (a geometric phase, then a floor $\propto \alpha/s$), not its constants.
- **11.18, 11.19 (and 11.21) boxed** $A_k \leq \nu/(\gamma + k)$ assumes only $2\mu\beta > 1$, but the induction needs $\gamma \geq 2\mu\beta$ and can fail without it ($\beta = 1.9$, $\gamma = 1$, $\sigma^2 = 0.05A_0$: $A_1 = 0.9905A_0 > \nu/2$); 11.19’s $\beta = \gamma = \mu = 1$ violates it. *In the notes:* the theorem assumes $\gamma \geq 2\mu\beta$, a box gives the counterexample, and the mean example uses $A_k = \sigma^2/k$.
- **11.28, 11.29 boxed** 11.28 calls the momentum step a weighted “average” (its weights sum to $1/(1 - \beta)$); 11.29 calls SGD’s reversals “the Week 10 zig-zag” (there is none at this step) and blames a factor of 7 on a step “past the stability limit” (the run is stable). *In the notes:* a weighted sum, reversals from sampling noise, a higher noise floor (box).

- **11.15** The frame says “(Both derivations: notebook.)”, but the notebook contains neither derivation of the noise-model constants. *In the notes:* both derivations (least squares $L_g^2 = 2a_{\max}^4$, $B^2 = \frac{2}{N} \sum_i \|a_i\|_2^2 r_i^{*2}$; logistic $L_g = 0$, $B = a_{\max}$) form the proof of the noise-constants proposition.
- **11.15–11.18, 11.20** Global μ -strong convexity forces $L_g \geq \mu$, yet the $L_g = 0$ example (logistic) is not strongly convex, 11.16 calls $L_g = 0$ “7.31 word for word”, and no frame matches theorems to examples. *In the notes:* a remark matches the examples to the theorems and proves $L_g \geq \mu$; $L_g = 0$ recovers Week 7 only formally, with $\alpha < 1/(2\mu)$.
- **11.16 (applied to batches on 11.24–11.26)** The noise model is defined for the single-draw gradient, but the fixed-step result is used for mini-batches. *In the notes:* the single-draw constants serve for any batch, since a batch only lowers the second moment; the mini-batch proposition gives the batch model $((L^2 + L_g^2/s)^{1/2}, B/\sqrt{s})$.
- **11.18** The side condition “valid when $A_k \leq A_0$, i.e. starting above the floor” presupposes that the fixed-step floor exists at the first step, which needs $\alpha_0 < 2\mu/L_g^2$. *In the notes:* $A_k \leq A_0$ is said to hold when $\alpha_0 < 2\mu/L_g^2$ and the run starts at or above the floor at α_0 ; the decaying-step theorem takes the folded recursion as its hypothesis.
- **11.19 and tutorial Q2** ω_{k+1} denotes both the $(k+1)$ th data point and the value drawn at step k , so the boxed running average reads as the average of the first $k+1$ data points. *In the notes:* drawn values are written $\omega_{(k+1)} := \omega_{\xi_k}$, and $\omega_1, \dots, \omega_N$ remain the data.
- **11.20** The averaged iterate $\bar{x}^T$ is never defined, the Jensen step is unnamed, the bound mixes A_0 and $\|x^0 - x^*\|_2$ without saying that x^0 is deterministic, and T is not introduced as a horizon. *In the notes:* the theorem fixes the horizon T , defines $\bar{x}^T$ with weights $\alpha_k / \sum_j \alpha_j$ and sets $A_0 = \|x^0 - x^*\|_2^2$; a lemma states Jensen’s inequality for a finite average.
- **11.13, 11.24** 11.13 calls “freeze the current point, average over one fresh draw” the whole toolkit, but the $1/s$ law of 11.24 also needs the unstated product rule $\mathbb{E}[e_j^\top e_l] = \mathbb{E}[e_j]^\top \mathbb{E}[e_l]$ for independent draws. *In the notes:* the rule is stated after the definition of expectation, the toolkit sentence says “apart from the product rule”, and the mini-batch proof uses it.
- **11.32** Bias correction divides by $1 - \beta_1^k$ and $1 - \beta_2^k$, but in the frame’s indexing d^{k+1} is produced at step k , so this divides by zero at $k = 0$. *In the notes:* the divisors are $1 - \beta_1^{k+1}$ and $1 - \beta_2^{k+1}$, as in the notebook.
- **11.32** “Adam = 11.28’s running direction” overlooks that Adam’s direction $\beta_1 d^k + (1 - \beta_1) g^k$ is normalised (weights summing to 1), whereas 11.28’s $d^{k+1} = \beta d^k + g^k$ has weights summing to $1/(1 - \beta)$. *In the notes:* Adam’s direction is called normalised, so the $\alpha(1 - \beta)$ rescaling is built in.
- **11.33** The raw-design losses (Adam 0.621, SGD 0.637) sit next to “optimum 0.610”, which is the standardised design’s optimum; the raw design’s own optimum is 0.619. *In the notes:* each loss is quoted against its own optimum: raw 0.637 and 0.621 against 0.619 (rare coefficient 0.14 and 2.44 against 3.04), standardised 0.611 for both against 0.610.
- **11.37** The SVRG cost is given as “one full gradient per pass”, omitting the two sampled gradients per inner step that the notebook charges. *In the notes:* two sampled gradients per inner step plus one full gradient per snapshot, that is $3N$ gradient evaluations per epoch of N inner steps.
- **11.24** “To within 2%”: the recomputed maximum deviation from the $1/s$ line is 2.04% (at $s = 32$). *In the notes:* “to within about 2%”.

- **11.36 (and 11.23)** Schematic plots label the rates $O(1/k)$, $O(1/\sqrt{k})$ (and $c/k^{0.9}$) against an axis of passes over the data, although for a sampled method the step counter k is not the number of passes. *In the notes:* the schematics are not reproduced, and rates are stated in steps ($O(1/\sqrt{T})$ after T steps).
- **11.14** The frame cites (7.24) for Δ_k , but 7.24 holds the one-step expansion, while Δ_k is introduced at 7.31. *In the notes:* Week 7's one-step lemma is cited for the expansion and its strongly convex theorem for Δ_k .
- **11.4** “A thousand gradient steps runs for about half a minute” at $N = 3 \cdot 10^6$, but the notebook's timing is 24 ms per gradient, about 24 seconds. *In the notes:* “about 24 seconds”.

Week 12

- **12.11 boxed** The frame writes the global constant as $L = \max_i L_i$ and says the global step is set by the “worst coordinate”; this holds only for diagonal Hessians (the deck's own tilted quadratic has $L = 1$ but $\max_i L_i = 0.776$). *In the notes:* the sandwich $\max_i L_i \leq L \leq \sum_i L_i$ is stated with its hypotheses, and the global step is set by the worst direction.
- **12.14–12.16 (and the 12.15 figure) boxed** No frame says that the constants $2d$ and $1 - \mu_L/d$ need uniform, independent draws and a minimiser with $R_L < \infty$; the “ $\mu = 0$: sublinear” curve has no minimiser and decays slower than $1/k$. *In the notes:* the theorems state every hypothesis; a box says the curve shows what strong convexity buys.
- **12.20 boxed** Exact block minimisation is written $x_{(j)}^{k+1} = \operatorname{argmin}_v f(x^k + U_j v)$, which sets the new block equal to the increment v (for $f = \frac{1}{2} \|x - \mathbf{1}\|_2^2$ at the optimum $x^k = \mathbf{1}$ it returns 0). *In the notes:* $v^* \in \operatorname{argmin}_v f(x^k + U_j v)$ and $x_{(j)}^{k+1} = x_{(j)}^k + v^*$, with a box.
- **12.35 boxed** “Provided ρ is in the right range” reads as a condition for convergence, although consensus ADMM is two-block and converges for every $\rho > 0$; the $\rho = 5000$ run reaches 1.4×10^{-8} by iteration 500. *In the notes:* ρ governs only the speed (a poorly chosen ρ is slow, not wrong), and a box says so.
- **12.32 vs 12.34** Frame 12.32 defines $u = \lambda/\rho$ with λ the multiplier, while 12.34 thresholds at $\text{soft}_{\lambda/\rho}$ with λ the lasso penalty, so the threshold can be mistaken for u . *In the notes:* the split lasso says λ is again the penalty and the multiplier is ρu ; the scalar example shows $u^* = 0.2 \neq \lambda/\rho = 1$ when $z^* = 0$.
- **12.35** The sketch draws three regions while the run uses four, the consensus updates are not on the frame, and “shares only z ” is inaccurate: each region must send its copy x_i and price u_i for the average. *In the notes:* the consensus updates are given, the run has $m = 4$ regions of 1000 rows, and each region shares only its copy and price, never its rows.
- **12.29** The five-unit example's data c , t , R , λ^0 are not on the slide, and at $\alpha = 1.2\alpha^*$ the linearised factor -1.4 predicts divergence, but the measured run stays bounded, with negative prices. *In the notes:* the example states the data, explains the bounded oscillation (units switch off above $c_i t_i$) and notes that an equality multiplier may be negative.
- **12.27** “Under Week 5's conditions equality holds, so maximising q solves the original problem” is loose: the local solutions at λ^* need not be feasible unless unique (linear costs are a counterexample). *In the notes:* for convex f_i and $R > 0$ (Slater point $x_i = R/m$), $\max q = p^*$, and the $x_i(\lambda^*)$ are optimal when the local solutions are unique.
- **12.27** $q'(\lambda) = \sum_i x_i(\lambda) - R$ needs convex f_i and unique local solutions; at a kink the bracket is a supergradient of the concave q ; and $\alpha < 2/S$ presumes a smooth dual. *In the notes:*

the proposition adds these hypotheses; for strongly convex costs every step with $\alpha < 2/S$ increases q (convergence is exact for a quadratic dual); convex costs need subgradient steps.

- **12.27** The dual function is written with min and is Week 5's $g(\lambda, \nu)$ under a new name, with the equality multiplier ν renamed λ ; the slide says neither, and a local infimum can be $-\infty$ or unattained. *In the notes:* the definition uses inf and says when it is a minimum; the text says that q is Week 5's g renamed and λ is Week 5's ν .
- **12.9 (and tutorial 12.42)** Coordinate-wise Lipschitz smoothness is never defined (the frame assumes the upper-model inequality, which is a consequence), and neither are e_i (12.4) and L_j (12.20). *In the notes:* the constants are defined by $|\nabla_i f(x + ue_i) - \nabla_i f(x)| \leq L_i |u|$, the upper model is derived from them, and e_i and L_j are defined at first use.
- **12.4** “One coordinate step is less powerful” is specific to the picture: on the same quadratic the exact step along x_1 reaches 11% of the starting value, better than the full step's 25% (with $\alpha = 0.5/L$). *In the notes:* the worked example calls it a rule of thumb, not a theorem, and gives the 11% case.
- **12.4 (figure)** The picture ends the coordinate step at $x_2 = -0.911$ (from the unrounded 35° rotation), whereas the printed three-decimal Q gives $x_2 = -0.912$ and $f(x^k) = 1.649$. *In the notes:* Q is printed, and -0.912 and 1.649 are used consistently in the text and the figure.
- **12.16** The cyclic guarantee assumes only convexity, differentiability and coordinate-wise Lipschitz gradients; the cited results also need a minimiser, in fact a bounded initial sublevel set. *In the notes:* the proposition assumes that $\{x : f(x) \leq f(x^0)\}$ is bounded.
- **12.22** “Proximal-coordinate descent keeps the coordinate-descent rates” is true for the orders but not for the constants in the convex case. *In the notes:* the $O(1/k)$ and linear rates carry over, with modified constants in the convex case (stated without proof).
- **12.7, 12.17, 12.26, 12.30–12.34, 12.42** Symbols change meaning within the week unannounced: A, b, r, u, q (three meanings), L, c, a_i (a column, Week 11's row) and m ; 12.30 also drops the budget model's $x \geq 0$ from $\mathcal{L}_\rho$. *In the notes:* each change is announced at first use, the deck's symbols are kept, and $\mathcal{L}_\rho$ keeps x in a domain such as $x \geq 0$.

Week 13

- **13.12 boxed** The slide says that one projection “spreads weight across almost every asset immediately”, but in the run both methods start from the uniform portfolio, so projected gradient starts with all 60 assets and sheds them slowly (60, 60, 56, 53 at $k = 1, 5, 20, 50$). *In the notes:* the run is reported as measured, and a single-asset start is added for contrast, from which one projected-gradient step with $\alpha = 1/L$ holds 8 to 16 of the 60 assets.
- **13.6, 13.4, 13.23 boxed** Frame 13.6 calls linear questions “the ones optimisation software answers best”, yet on the deck's own polyhedron the LP oracle costs about as much as the projection QP (13.4), and in 13.23 Frank–Wolfe with 59 LPs trails projected gradient with 39 QPs. *In the notes:* escape 1 does not pay on a generic $\{Ax \leq b\}$; it earns its place where the oracle is structurally cheap (simplex, ℓ_1 ball, shortest path).
- **13.14 boxed** The warning “the optimum sits at a vertex the oracle keeps swapping away from” can describe the best case: with $y = (4.5, 2.5)$ in the pentagon example, $x^* = v_3$ and the oracle returns v_3 at every step, so $x^1 = x^*$. *In the notes:* Frank–Wolfe is to be avoided when the optimum lies on a face between several vertices, where the oracle keeps swapping and the iterates zig-zag (away steps repair this).
- **13.8, 13.12** “Start from any vertex” and “ x^k is a combination of at most $k + 1$ vertices” assume a vertex start, which the measured run (uniform start) does not use. *In the notes:* x^k

combines at most k vertices for $k \geq 1$ from any start, since $\gamma_0 = 1$ discards x^0 , and at most $k + 1$ counting a vertex start.

- **13.3** The takeaway “a method is only as cheap as its cheapest primitive” is reversed: the cost per iteration is set by the most expensive primitive. *In the notes:* the key idea reads “only as cheap as its most expensive primitive”.
- **13.4** The figure’s LP-oracle curve is computed over the polyhedron intersected with the box $\|x\|_\infty \leq 50$ (some random polyhedra are unbounded), not over the set of the QP, and the slide does not mention the LP curve. *In the notes:* text and caption say that the LP is solved over the polyhedron intersected with a large box and costs about as much as the QP.
- **13.6, 13.40** Frame 13.6 says that a linear objective over a polyhedron is minimised at a corner, which needs boundedness, and that the ℓ_1 -ball oracle is a single argmin, whereas frame 13.40 makes it one argmax with a sign. *In the notes:* C is compact from the oracle’s definition on, the vertex fact is stated for a compact polyhedron, and the ℓ_1 -ball oracle is $-\tau \text{sign}(v_j)e_j$ with $j \in \text{argmax}_j |v_j|$.
- **13.11** “The ratio $g_k/(f(x^k) - f^*)$ grows from 4.1 to 38”: these are its values at $k = 10$ and $k = 399$, and the ratio is not monotone (minimum 3.1, maximum 50). *In the notes:* the ratio is about 4 at $k = 10$, about 11 at $k = 100$ and about 38 at $k = 399$, fluctuating on the way.
- **13.13** The guarantee is displayed without $k \geq 1$, without convexity of f and without a formula for the diameter D ; at $k = 0$ it can fail for an arbitrary start. *In the notes:* the Frank–Wolfe rate theorem assumes f convex and L -smooth and C compact and convex with $D = \max_{x,y \in C} \|x - y\|_2$, and holds for $k \geq 1$.
- **13.14** “The $O(1/k)$ order cannot be accelerated” is stronger than the known lower bound for linear-oracle methods, which holds for k up to the order of the dimension. *In the notes:* the claim carries the qualifier “in high dimension, for k up to about d ”.
- **13.16** The slide shows the x - and z -steps of ADMM but not the u -step, and does not say that the x -step system needs $Q + \rho A^\top A$ positive definite. *In the notes:* all three updates are displayed, with $u^k = \lambda^k/\rho$ and the condition (Q positive definite, or A with independent columns).
- **13.18** The barrier problem is written without saying that it is posed on $\{Ax < b\}$ and that f is convex. *In the notes:* the definition of the log barrier states the domain $\{x : Ax < b\}$ and assumes f convex and differentiable.
- **13.19** The point labelled “analytic centre” in the figure is $x^*(1)$, the first point of the traced path (t from 1 to 10^4), not the analytic centre (0.52, 0.29). *In the notes:* the caption says so and places the analytic centre further back along the curve, away from the optimum.
- **13.20** The dual function $q(\lambda)$ of the LP is used without being derived, and Week 5’s LP pair has a different form. *In the notes:* the proof derives $q(\lambda) = -b^\top \lambda$ when $A^\top \lambda + c = 0$ (and $-\infty$ otherwise) from the Lagrangian, and the notation paragraph announces c as the cost of the LP.
- **13.21, 13.24** “Every iterate stays strictly inside C ”, “a short phase I” and “feasible during the run: yes, strictly inside” describe the textbook barrier method, not the infeasible-start primal–dual solver (MOSEK) measured in 13.22–13.23. *In the notes:* a remark separates the two, the escapes table points to it, and production solvers are said to avoid phase I.
- **13.23** The instance has $m = 800 = 4d$ rows with $d = 200$, not the $m = 2d$ of frame 13.4: 400 random rows plus 400 box rows $-10 \leq x_i \leq 10$ that make C compact for the linear oracle.

In the notes: the example explains the box in one clause and notes that it does not bind ($\|x^*\|_\infty = 3.37$).

- **13.23** The vertical plunge of the ADMM curve near 2 s is a plotting artefact: the reference p^* is MOSEK's value, 4.6×10^{-6} above the exact optimal value, so the plunge marks ADMM becoming more accurate than the reference. *In the notes:* the caption explains this, and the objective error 2.8×10^{-9} is stated against the exact optimal value.
- **13.29** “What no assumption could buy” overstates: strong convexity improves both $O(1/\sqrt{k})$ and $O(1/\sqrt{T})$ (as the ladder's subgradient row says); the optimality claims hold within their classes. *In the notes:* “two rates no algorithmic device, momentum included, could improve within their class”, and Frank–Wolfe's $O(1/k)$ as a third.
- **13.32** “ $\gamma_k = 2/(k+2)$ often gives the $1/k^2$ behaviour we measured” contradicts the measured slope -1.7 of frame 13.13. *In the notes:* the open-loop step is said to often beat the guaranteed $1/k$, as in the measured run.
- **13.35** Tutorial question 1, item 5, writes the sector caps as $Sw \leq c$, although the set of item 2 is written in x . *In the notes:* the first exercise, part (e), writes $Sx \leq c$.
- **13.6, 13.16, 13.20** The letter c changes role (sector caps, linear term of the QP, cost of an LP) and μ means both the mean returns and the strong-convexity constant; the deck announces only the LP change. *In the notes:* the introduction announces every role of c and μ , that p^* (Week 5's notation) is f^* , and that $a_i^\top$ and m are again rows and constraint counts after Week 12.
- **13.25** The selection map files Newton's method under “a gradient”, has no branch for the subgradient method or coordinate descent, and sends “nothing cheap” to dualise-and-split, although splitting relies on its pieces being cheap. *In the notes:* Newton appears under an affordable Hessian solve, the subgradient method and coordinate descent are added, and splitting is the answer when the pieces, not the whole problem, are cheap.
- **13.28** The headline-rate column of the ladder gives strongly convex rates only for the gradient and subgradient rows. *In the notes:* every row with a strongly convex result in its home week shows it, and the ADMM rate is stated for the averaged iterate.
- **13.39** The review exercise asks for the “two” rates that momentum cannot improve; the week itself proves a third, Frank–Wolfe's $O(1/k)$ for methods that see C only through a linear oracle. *In the notes:* the exercise asks which rates cannot be improved within their class, without fixing their number.