

COMPUTER SESSION # 2

The quadratic case. We consider the minimisation of a strictly convex quadratic function $f : \mathbb{R}^d \rightarrow \mathbb{R}$:

$$f(x) = \frac{1}{2} \langle x, Ax \rangle - \langle b, x \rangle$$

where A is symmetric, positive definite, and $b \in \mathbb{R}^d$. We denote by $\lambda_{\min}$ and $\lambda_{\max}$ the smallest and largest eigenvalues of A . The conditioning number of A is $\kappa = \lambda_{\max}/\lambda_{\min}$.

Exercise 0.1 (Fixed Step Size and conditioning number). Let $b = (0, 0)^\top \in \mathbb{R}^2$. We consider two matrices:

$$A_1 = \begin{pmatrix} 2 & 0 \\ 0 & 2 \end{pmatrix}, \quad A_2 = \begin{pmatrix} 10 & 0 \\ 0 & 1 \end{pmatrix}$$

- (1) What is the global minimiser x^* of f in both cases? What are the conditioning numbers κ_1 and κ_2 ?
- (2) Write a function `GD_fixed(Q, b, x0, tau, N_iter)` that performs N_iter iterations of gradient descent with a fixed step size `tau`, starting from `x0`. It should return the sequence of iterates as an array.
- (3) For both A_1 and A_2 , run the algorithm for $N = 50$ iterations starting from $x_0 = (2, 2)^\top$, using the theoretical optimal fixed step size $\tau^* = \frac{2}{\lambda_{\max} + \lambda_{\min}}$.
- (4) Plot the contour lines of f and the trajectory of the iterates for both matrices. How does the conditioning number affect the geometry of the level sets and the trajectory?
- (5) Plot the distance to the optimum $\|x_k - x^*\|$ in logarithmic scale. Compare the empirical convergence rate with the theoretical bound $\left(\frac{\kappa-1}{\kappa+1}\right)^k$.

Exercise 0.2 (Exact Line Search). Instead of a fixed step size, we can choose at each step the optimal size τ_k that minimises the function along the search direction $d_k = \nabla f(x_k)$. That is, $\tau_k = \operatorname{argmin}_{\tau > 0} f(x_k - \tau d_k)$.

- (1) Analytically compute the value of τ_k as a function of d_k and A .
- (2) Show that for this exact line search, consecutive gradients are orthogonal: $\langle d_{k+1}, d_k \rangle = 0$. What does this imply for the shape of the trajectory?
- (3) Write a function `GD_exact(A, b, x0, N_iter)` implementing this method.
- (4) Run it on the matrix A_2 from the previous exercise. Plot the trajectory on the contour lines and compare the empirical convergence rate with the fixed optimal step size algorithm. Is exact line search faster on ill-conditioned problems?

The general case.

Exercise 0.3 (Discretisation of the Gradient). When the gradient is not available analytically, we must approximate it. The centred finite difference approximation of $\partial f / \partial x_i$ is given by:

$$[\nabla_{\text{num}} f(x)]_i = \frac{f(x + h e_i) - f(x - h e_i)}{2h}$$

where e_i is the i -th basis vector and $h > 0$ is the discretisation step.

- (1) Write a function `approx_grad(f, x, h)` that computes this numerical gradient for a function $f : \mathbb{R}^d \rightarrow \mathbb{R}$.
- (2) Let $g(x) = x_1^3 + x_2^3 - 3x_1x_2$. Evaluate the absolute error $\|\nabla_{\text{num}}g(x) - \nabla g(x)\|$ at $x = (1, 2)^\top$ for various values of $h \in [10^{-16}, 10^{-1}]$ (using `np.logspace`).
- (3) Plot the error as a function of h in log-log scale. What is the optimal value of h ?

Exercise 0.4 (The Rosenbrock Function and Armijo Backtracking). The Rosenbrock function is the canonical “bad function” for optimisation algorithms:

$$f(x, y) = (1 - x)^2 + 100(y - x^2)^2$$

For non-quadratic functions, exact line search is usually impossible. Instead, we use **Backtracking Line Search (Armijo Rule)**. At iteration k , given $d_k = \nabla f(x_k)$, we start with a step $\tau = \tau_{\max}$ and repeatedly multiply τ by a factor $\beta \in (0, 1)$ until the following condition is met:

$$f(x_k - \tau d_k) \leq f(x_k) - c\tau \|d_k\|^2$$

where $c \in (0, 1)$ is a small tolerance.

- (1) Compute analytically the gradient and the Hessian of f . What is the unique global minimiser x^* ?
- (2) Evaluate the Hessian at x^* . Compute its conditioning number. What does this suggest about the asymptotic convergence rate of gradient descent near the minimum?
- (3) Write a function `GD_armijo(f, grad_f, x0, N_iter, tau_max=1.0, c=1e-4, beta=0.5)` implementing gradient descent with this Armijo rule.
- (4) Run it on the Rosenbrock function starting from $x_0 = (-1.5, 11.5)^\top$ for $N = 5000$ iterations. Plot the contour lines of f (use a log scale for the levels) and overlay the trajectory. Plot the convergence rate.

Error estimates and Chebyshev polynomials. We now move back to the quadratic case and we assume that

$$f : x \mapsto \frac{1}{2} \langle x, Ax \rangle.$$

Regarding A , we assume that its extremal eigenvalues $\lambda_{\min}$ and $\lambda_{\max}$ are known *exactly*. In practice, however, this is rarely the case. Fortunately, several robust techniques exist for estimating lower and upper bounds on these extremal eigenvalues. These include: - [Gershgorin-type bounds](https://en.wikipedia.org/wiki/Gershgorin_circle_theorem), which provide crude yet inexpensive a priori estimates of the spectrum, - [power-type iteration](https://en.wikipedia.org/wiki/Power_iteration), which efficiently approximates a given eigenvalue from repeated matrix–vector products, - the [Lanczos algorithm](https://en.wikipedia.org/wiki/Lanczos_algorithm), a Krylov-space method that rapidly approximates both extremal eigenvalues of a symmetric matrix, - estimations from the behaviour of the gradient descent itself, which exploit the smoothness inequality using finite differences of gradients (for an upper bound on $\lambda_{\max}$) and the strong convexity inequality (for a lower bound on $\lambda_{\min}$).

For an illustrative purpose, we consider a *two-dimensional* problem. In this case, the construction of a matrix A with prescribed condition number $\kappa(A)$ will be

achieved by specifying the values of its two eigenvalues $\lambda_{\min} = 1$ and $\lambda_{\max} = \kappa(A)$, and using a change of orthonormal basis in the form of a rotation.

```

1 def make_spd_from_kappa(kappa, theta_deg = 30.):
2     # build a two-by-two SPD matrix with prescribed condition number
      from a diagonal matrix using a rotation
3     theta = np.radians(theta_deg)
4     Q = np.array([[np.cos(theta), -np.sin(theta)],
5                   [np.sin(theta),  np.cos(theta)]])
6     A = Q @ np.diag([1.0, float(kappa)]) @ Q.T
7     return A
    
```

We apply a gradient descent for f and, at each step k , we take a step size α_k . One easily verifies that

$$\forall k \in \mathbb{N}, x_{k+1} - x = (I - \alpha_k A)(x_k - x)$$

and, by induction,

$$\forall k \in \mathbb{N}, x_{k+1} - x = P_k(A)(x_0 - x),$$

where P_k is the polynomial of degree k defined by

$$P_k(t) = \prod_{j=0}^{k-1} (1 - \alpha_j t).$$

Thus, an upper bound for the norm of the error is given by

$$\forall k \in \mathbb{N}, \|x_k - x\| \leq \|P_k(A)\| \|x_0 - x\|.$$

Since the matrix A is real symmetric positive definite, with eigenvalues in the interval $[\lambda_{\min}, \lambda_{\max}]$, it follows that:

$$\|P_k(A)\| = \max_{\lambda_{\min} \leq t \leq \lambda_{\max}} |P_k(t)|.$$

In practice, the extremal eigenvalues may not be known exactly, but positive bounds $m \leq \lambda_{\min}$ and $M \geq \lambda_{\max}$ can generally be obtained using the techniques mentioned earlier. This leads to the following problem: *how should one choose the steps $\alpha_0, \dots, \alpha_{k-1}$ so that the resulting polynomial P_k has, among all polynomials having value 1 at the origin, the smallest possible magnitude over the interval $[m, M]$?*

Here, since the exact values of the extremal eigenvalues are known, the bounds m and M are sharp.

A naïve solution is to choose the (optimal) constant step size $\alpha_k = \frac{2}{\lambda_{\max} + \lambda_{\min}}$ in the expression. This ensures that $|P_k(\lambda_{\min})| = |P_k(\lambda_{\max})|$, and the convergence estimate for the stationary descent given above holds.

Exercise 0.5. Choose $m = 1$ and $M = 10$ (so that $\kappa(A) = 10$) and represent the corresponding polynomials P_k for k up to 7 over the relevant interval $[m, M]$.

A better – and in fact optimal – choice is provided by the Chebyshev polynomials of the first kind (https://en.wikipedia.org/wiki/Chebyshev_polynomials). These satisfy the three-term recurrence:

$$\forall k \in \mathbb{N}^*, \forall s \in \mathbb{R}, T_{k+1}(s) = 2sT_k(s) - T_{k-1}(s),$$

with $T_0(s) = 1$ and $T_1(s) = s$.

Exercise 0.6. *Construct and represent the first eight Chebyshev polynomials over the interval $[-1, 1]$.*

A key property of the Chebyshev polynomials is that, for any non-zero natural integer k , among the polynomials with leading coefficient 1 (monic polynomials), the monic polynomial of degree k with minimal maximum absolute value on the interval $[-1, 1]$ is

$$\frac{1}{2^{k-1}} T_k$$

To use them for our purpose, we must apply both a change of variable and an appropriate rescaling.

The following affine transformation is used to map the interval $[m, M]$ to $[-1, 1]$:

$$\forall t \in [m, M], s = \frac{M + m - 2t}{M - m}.$$

Note that $s = 1$ corresponds to $t = m$ and $s = -1$ corresponds to $t = M$. This transformation ensures that the behaviour of the Chebyshev polynomials on $[-1, 1]$ is reflected on the interval $[m, M]$.

In the error analysis above, we have seen that the polynomial P_k is required to have value 1 at the origin. After applying the affine transformation, the value T_k takes at the point corresponding to $t = 0$ might not be 1. Thus, we divide it by the value of T_k at $\frac{M+m}{M-m}$,

$$P_k(t) = T_k\left(\frac{M + m - 2t}{M - m}\right) \cdot T_k\left(\frac{M + m}{M - m}\right)^{-1},$$

ensuring that

$$P_k(0) = T_k\left(\frac{M + m - 0}{M - m}\right) \cdot T_k\left(\frac{M + m}{M - m}\right)^{-1} = 1.$$

Exercise 0.7. *Choose $m = 1$ and $M = 10$ and represent the rescaled Chebyshev polynomials of degree 2 to 7 over the interval $[m, M]$. Observe that they are much better behaved than the polynomials of the preceding naïve approach in terms of magnitude.*

Using that $\kappa(A) = \frac{M}{m}$ when the bounds m and M are sharp, an upper bound on the norm of $P_k(A)$ in terms of $\kappa(A)$ is derived from a lower bound on $|T_k\left(\frac{M+m}{M-m}\right)|$. It can be showed that

$$\|P_k(A)\| \leq 2 \exp\left(-\frac{2k}{\sqrt{\kappa(A)}}\right).$$

This demonstrates that acceleration can be expected by suitably modifying the recurrence relation for the gradient descent.

The accelerated method. From the recursive definition of the Chebyshev polynomials, a recurrence relation for the rescaled Chebyshev polynomials is obtained:

$$\forall k \in \mathbb{N}^*, P_{k+1}(t)c_{k+1} = 2 \frac{M+m-2t}{M-m} P_k(t)c_k - P_{k-1}(t)c_{k-1},$$

where

$$\forall k \in \mathbb{N}, c_k = T_k \left(\frac{M+m}{M-m} \right).$$

We may rearrange this formula into the form:

$$\forall k \in \mathbb{N}^*, P_{k+1}(t) = (\gamma_k + \eta_k t) P_k(t) + \mu_k P_{k-1}(t),$$

where

$$\forall k \in \mathbb{N}^*, \gamma_k = 2 \frac{M+m}{M-m} \frac{c_k}{c_{k+1}}, \eta_k = -\frac{4}{M-m} \frac{c_k}{c_{k+1}} \text{ and } \mu_k = -\frac{c_{k-1}}{c_{k+1}}.$$

We may observe that $\gamma_k + \mu_k = 1$ due to the normalising condition.

Remembering that $x_k - x = P_k(A)(x_0 - x)$, we have:

(0.1)

$$\begin{aligned} \forall k \in \mathbb{N}^*, x_{k+1} - x &= P_{k+1}(A)(x_0 - x) = (\gamma_k + \eta_k A) P_k(A)(x_0 - x) + \mu_k P_{k-1}(A)(x_0 - x) \\ &= (\gamma_k + \eta_k A)(x_k - x) + \mu_k (x_{k-1} - x), \end{aligned}$$

from which we infer the accelerated iteration:

$$\forall k \in \mathbb{N}^*, x_{k+1} = (\eta_k A + \gamma_k) x_k + \mu_k x_{k-1} - \eta_k b,$$

or, equivalently, using that $\gamma_k + \mu_k = 1$,

$$\forall k \in \mathbb{N}^*, x_{k+1} = x_k + \eta_k (Ax_k - b) - \mu_k (x_k - x_{k-1}),$$

the additional term $-\mu_k (x_k - x_{k-1})$ being interpreted as a *momentum* term, which tends to push the new approximation in the direction of where it was headed at the preceding step.

Exercise 0.8. *Illustrate the convergence behaviour of the Chebyshev-accelerated gradient descent and compare it with the two gradient descent methods used on the example introduced at the beginning of the tutorial. In particular, you will need to tweak the codes proposed for fixed step size in order to incorporate a tolerance and the number of iterations necessary to reach a solution.*

In low dimension, and especially in two dimensions, the steepest descent method often performs surprisingly well: with exact line search, it adapts greedily to the local geometry of the problem and can eliminate dominant error components very quickly. As a result, for moderate condition numbers, steepest descent may converge faster in practice than Chebyshev acceleration, despite its inferior worst-case theoretical rate.

On the other hand, the Chebyshev-accelerated method is designed to minimise the worst-case error over an entire spectral interval, not just the error along the current direction, which makes them more conservative in small dimensions and in the early iterations. Their advantage becomes visible only when the dimension is large

enough for many eigenmodes to interact, and when the condition number is sufficiently high for the minimax polynomial structure to matter. This motivates the next experiment in higher dimension, where Chebyshev acceleration outperforms steepest descent by a wide margin.

A high-dimensional example. In this experiment, the matrix A is built as a random symmetric positive-definite matrix with a prescribed spectrum. First, an orthogonal matrix Q is generated by applying a QR decomposition to a random Gaussian matrix; this ensures that Q is uniformly distributed over the space of orthogonal matrices. A set of eigenvalues covering the interval $[m, M]$ is chose, either linearly or logarithmically spaced, depending on how challenging we want the problem to be. The matrix is then formed as

$$A = Q \text{diag}(\lambda_1, \dots, \lambda_n) Q^T,$$

which produces a symmetric positive-definite matrix whose eigenvectors are random and whose eigenvalues match exactly the desired spectral bounds. The corresponding routines are given hereafter.

```

1
2 rng = np.random.default_rng(2026)
3
4 def random_orthogonal(n, rng):
5     """Random orthogonal matrix via QR of a Gaussian matrix."""
6     X = rng.standard_normal((n, n))
7     Q, R = np.linalg.qr(X)
8     # Ensures a proper rotation (det=+1), but not strictly
9     required
10    Q *= np.sign(np.diag(R))
11    return Q
12
13 def make_spd_with_spectrum(n, m, M, spread='log', rng=rng):
14     """
15     Build an nXn SPD matrix with eigenvalues in [m,M].
16     'spread' can be:
17     - 'lin': linearly spaced
18     - 'log': log-spaced (often a tougher case for SD)
19     """
20     if spread == 'lin':
21         lambdas = np.linspace(m, M, n)
22     elif spread == 'log':
23         lambdas = np.geomspace(m, M, n)
24     else:
25         raise ValueError("spread must be 'lin' or 'log'")
26     Q = random_orthogonal(n, rng)
27     A = (Q * lambdas) @ Q.T # Q diag(lambdas) Q^T without
28     forming diag explicitly
29     return A, lambdas[0], lambdas[-1] # return exact m, M (
30     sharp bounds)

```

Exercise 0.9. *Using the problem parameter values fixed below, compare the number of iterations required by the methods (exact line search, Chebyshev acceleration, constant step size).*