Learning objectives.

1. Compute the *gradient* and *Hessian* of simple multivariate functions.
2. Apply *gradient descent*, *stochastic gradient descent*, and *SGD with momentum* from a formula sheet.
3. Know why *ADAM* is basically just a variant of GD.
4. Apply *Newton's method* by solving the Newton linear system.
5. Convert equality constraints into a quadratic-penalty objective and inequality constraints into a log-barrier objective.

The emphasis is deliberately plug-and-chug. At this stage, students should recognize general patterns and know how to execute these methods correctly. Convergence analysis comes later in the course.

1 Linear algebra review

1.1 Vectors

Vector $x \in \mathbb{R}^n$ lies in n-dimensional Euclidean space:

$$x = (x_1, \dots, x_n) \coloneqq \begin{bmatrix} x_1 & x_2 & \cdots & x_n \end{bmatrix}^T = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}.$$

Inner product

$$\begin{aligned} \langle a, x \rangle \coloneqq a^T x = \begin{bmatrix} a_1 & a_2 & \cdots & a_n \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \\ = a_1 x_1 + a_2 x_2 + \cdots + a_n x_n \end{aligned}$$

Euclidean norm.

$$\|x\| = \sqrt{\langle x, x\rangle} = \sqrt{x^T x} = \left(\sum_{i=1}^{n} x_i^2\right)^{1/2}.$$

Two important results for the Euclidean norm (not used today, but will be important for next lecture):

1. *Pythagorean theorem:*
$$x^T y = 0 \iff \|x+y\|^2 = \|x\|^2 + \|y\|^2 .$$
2. *Cauchy–Schwarz inequality:*
$$|x^T y| \le \|x\| \, \|y\| .$$
Equality iff x and y are linearly dependent.

ℓ_p norms (for $p \geq 1$).

$$\|x\|_p = \left(\sum_{i=1}^{n} |x_i|^p\right)^{1/p}, \qquad \|x\|_\infty = \max_{i\in\{1,\dots,n\}} |x_i|.$$

All of these norms satisfy the following properties:

1. $\|x\|_p \geq 0$,
2. $\|x\|_p = 0 \iff x = 0$,
3. $\|cx\|_p = |c| \, \|x\|_p$ for $c \in \mathbb{R}$,
4. $\|x+y\|_p \leq \|x\|_p + \|y\|_p$ (triangle inequality).

1.2 Matrices

Matrix $A \in \mathbb{R}^{m\times n}$ lies in an mn-dimensional Euclidean space

$$A_{m\times n} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix} = \begin{bmatrix} - & a_1^T & - \\ - & a_2^T & - \\ & \vdots & \\ - & a_m^T & - \end{bmatrix}.$$

Matrix-vector product

$$Ax = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} - & a_1^T & - \\ - & a_2^T & - \\ & \vdots & \\ - & a_m^T & - \end{bmatrix} \begin{bmatrix} | \\ x \\ | \end{bmatrix}$$

$$= \begin{bmatrix} a_1^T x \\ a_2^T x \\ \vdots \\ a_m^T x \end{bmatrix} = \begin{bmatrix} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 \\ a_{21}x_1 + a_{22}x_2 + a_{23}x_3 \\ a_{31}x_1 + a_{32}x_2 + a_{33}x_3 \end{bmatrix}$$

Matrix inversion. A square matrix A is invertible if there is a unique matrix A^{-1} satisfying $A^{-1}A = AA^{-1} = I$. In that case,

$$Ax = b \iff x = A^{-1}b.$$

Two-by-two matrix inversion (invertible iff $ad \neq bc$)

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix}^{-1} = \frac{1}{ad - bc} \begin{bmatrix} d & -b \\ -c & a \end{bmatrix}.$$

Question 1. Why do they say to **never** form the inverse A^{-1}?

Solution: It is usually faster and more numerically stable to implicitly solve $Ax = b$ than to multiply against the explicit inverse $x = A^{-1}b$.

Forward substitution (invertible iff A_{ii} invertible for all i)

$$\begin{bmatrix} A_{11} & & & \\ A_{21} & A_{22} & & \\ \vdots & \vdots & \ddots & \\ A_{n1} & A_{n2} & \cdots & A_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix} \Leftrightarrow \begin{aligned} x_1 &= A_{11}^{-1} b_1 \\ x_2 &= A_{22}^{-1}(b_2 - A_{21}x_1) \\ x_3 &= A_{33}^{-1}(b_3 - A_{32}x_2 - A_{31}x_1) \\ &\vdots \\ x_n &= A_{nn}^{-1}(b_n - \textstyle\sum_{i=1}^{n-1} A_{ni}x_i) \end{aligned}$$

Special case: block diagonal (invertible iff A_{ii} invertible for all i)

$$\begin{bmatrix} A_{11} & & & \\ & A_{22} & & \\ & & \ddots & \\ & & & A_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix} \iff \begin{array}{l} x_1 = A_{11}^{-1} b_1 \\ x_2 = A_{22}^{-1} b_2 \\ x_3 = A_{33}^{-1} b_3 \\ \vdots \\ x_n = A_{nn}^{-1} b_n \end{array}$$

1.3 Gradients and Hessians

Let $f(x_1, \ldots, x_n)$ be real-valued. Then

$$\nabla f(x) = \begin{bmatrix} \partial f / \partial x_1 \\ \partial f / \partial x_2 \\ \vdots \\ \partial f / \partial x_n \end{bmatrix}.$$

Hessian is the Jacobian of the gradient (symmetric for the smooth functions considered here)

$$\nabla^2 f(x) = \begin{bmatrix} \dfrac{\partial^2 f}{\partial x_1^2} & \cdots & \dfrac{\partial^2 f}{\partial x_1 \partial x_n} \\ \vdots & \ddots & \vdots \\ \dfrac{\partial^2 f}{\partial x_n \partial x_1} & \cdots & \dfrac{\partial^2 f}{\partial x_n^2} \end{bmatrix}.$$

2 Gradient descent

For $\min_x f(x)$, gradient descent with stepsize $\eta > 0$ is written

$$x_{k+1} = x_k - \eta \underbrace{\nabla f(x_k)}_{g_k}.$$

We commonly write g_k for the gradient evaluated at the k-th iteration.

Example 1. Gradient descent. Let

$$f(x) = x_1^4 + 2x_2^2 + \exp(-x_3) + \exp(2x_3).$$

1. Derive $\nabla f(x)$.
2. Perform one gradient descent iteration from $x_0 = (1, 2, 0)$ with stepsize $\eta = 1/2$.

Solution:

$$\nabla f(x) = \begin{bmatrix} 4x_1^3 \\ 4x_2 \\ -\exp(-x_3) + 2\exp(2x_3) \end{bmatrix}.$$

Plug in our point

$$\nabla f(1, 2, 0) = \begin{bmatrix} 4(1)^3 \\ 4(2) \\ -\exp(-0) + 2\exp(0) \end{bmatrix} = \begin{bmatrix} 4 \\ 8 \\ 1 \end{bmatrix}.$$

The update is then

$$\begin{aligned} x_1 &= x_0 - \eta \nabla f(x_0) \\ &= \begin{bmatrix} 1 \\ 2 \\ 0 \end{bmatrix} - \frac{1}{2} \begin{bmatrix} 4 \\ 8 \\ 1 \end{bmatrix} = \begin{bmatrix} -1 \\ -2 \\ -1/2 \end{bmatrix}. \end{aligned}$$

3 Newton's method

For $\min_x f(x)$, Newton's method with stepsize $\eta > 0$ is written

$$x_{k+1} = x_k - \eta \underbrace{\nabla^2 f(x_k)^{-1}}_{B_k} \underbrace{\nabla f(x_k)}_{g_k}.$$

The matrix B_k is called the *preconditioner*. (Sometimes, this name also refers to B_k^{-1}.) It is intended to make GD converge faster.

The resulting method is a kind of "preconditioned GD."

Example 2. Newton's method. Let

$$f(x) = x_1^4 + 2x_2^2 + \exp(-x_3) + \exp(2x_3).$$

1. Derive $\nabla^2 f(x)$.
2. Perform one **full** Newton iteration from $x_0 = (1, 2, 0)$.

Solution: From before,

$$\nabla f(x) = \begin{bmatrix} 4x_1^3 \\ 4x_2 \\ -\exp(-x_3) + 2\exp(2x_3) \end{bmatrix}$$

and $\nabla f(1, 2, 0) = (4, 8, 1)$. Therefore,

$$\nabla^2 f(x) = \begin{bmatrix} 12x_1^2 & 0 & 0 \\ 0 & 4 & 0 \\ 0 & 0 & \exp(-x_3) + 4\exp(2x_3) \end{bmatrix}.$$

At $x_0 = (1, 2, 0)$, we solve for Newton direction p

$$\nabla^2 f(x_0)p = \begin{bmatrix} 12 & 0 & 0 \\ 0 & 4 & 0 \\ 0 & 0 & 5 \end{bmatrix} \begin{bmatrix} p_1 \\ p_2 \\ p_3 \end{bmatrix} = - \begin{bmatrix} 4 \\ 8 \\ 1 \end{bmatrix} = -\nabla f(x_0),$$

and find $p = (-1/3, -2, -1/5)$. Full update means $\eta = 1$. We have

$$x_1 = x_0 + p = \begin{bmatrix} 1 \\ 2 \\ 0 \end{bmatrix} + \begin{bmatrix} -1/3 \\ -2 \\ -1/5 \end{bmatrix} = \begin{bmatrix} 2/3 \\ 0 \\ -1/5 \end{bmatrix}.$$

4 Stochastic gradient descent

SGD is designed to optimize a finite-sum, expectation-like objective

$$f(x) = \frac{1}{m}\sum_{i=1}^{m} f_i(x) \approx \mathbb{E}\left[\boldsymbol{f}(x)\right].$$

Recall that GD computes the full-batch gradient

$$\nabla f(x_k) = \frac{1}{m}\sum_{i=1}^{m} \nabla f_i(x_k).$$

This is considered "slow."

Question 2. Why?

Solution: $O(m)$ work at every iteration.

SGD instead computes the *stochastic gradient*. In this lecture, we use random reshuffling: at the start of each epoch, shuffle the m components and iterate through them without replacement. With a single component,

$$x_{k+1} = x_k - \eta g_k \quad \text{where } g_k = \nabla f_i(x_k),$$

or with a *minibatch* B from the shuffled order,

$$g_k = \frac{1}{|B|}\sum_{i\in B} \nabla f_i(x_k), \qquad |B| = N.$$

Exhausting the complete set of m samples this way is called an *epoch*. Without qualifiers, we understand "SGD" to mean $N = 1$.

Example 3. Stochastic gradient descent. Let $f(x) = \frac{1}{2}(f_1(x) + f_2(x))$, where

$$f_1(x) = \frac{1}{2}(x-1)^2, \qquad f_2(x) = \frac{1}{2}(x-3)^2.$$

Starting from $x_0 = 0$ with $\eta = 1/2$:

1. take one gradient-descent step;
2. take one epoch of SGD in the order $1, 2$.

Solution: Since $\nabla f(0) = \frac{1}{2}[(-1) + (-3)] = -2$, one GD step gives $x_1 = 1$.

For SGD, the first gradient is $\nabla f_1(0) = -1$, so $x \leftarrow 1/2$. The second gradient is $\nabla f_2(1/2) = -5/2$, so

$$x \leftarrow \frac{1}{2} - \frac{1}{2}\left(-\frac{5}{2}\right) = \frac{7}{4}.$$

5 SGD with momentum

Let g_k denote the stochastic gradient used at iteration k. Momentum keeps a running average of past gradients, with parameter $\beta \in [0, 1)$

$$
\begin{aligned}
v_k &= \beta v_{k-1} + (1-\beta) g_k, \\
x_k &= x_{k-1} - \eta v_k.
\end{aligned}
$$

This is a classical momentum update. It is inspired by a closely-related scheme that was shown by Nesterov to achieve optimal convergence rate for smooth convex optimization.

Question 3. How to "turn off" momentum?

Solution: Set $\beta = 0$. So β is literally "the amount" of momentum.

> **Example 4. Momentum.** Let $f_1(x) = f_2(x) = \frac{1}{2}(x-2)^2$. Starting from $x_0 = 0$ and $v_0 = 0$, perform one epoch of SGD with momentum using $\eta = 1/2$ and $\beta = 1/2$.

Solution: For the first sample, $g_1 = -2$, hence

$$
\begin{aligned}
v_1 &= \frac{1}{2}(0) + \frac{1}{2}(-2) = -1, \\
x_1 &= 0 - \frac{1}{2}(-1) = \frac{1}{2}.
\end{aligned}
$$

For the second sample, $g_2 = x_1 - 2 = -3/2$, hence

$$
\begin{aligned}
v_2 &= \frac{1}{2}(-1) + \frac{1}{2}\left(-\frac{3}{2}\right) = -\frac{5}{4}, \\
x_2 &= \frac{1}{2} - \frac{1}{2}\left(-\frac{5}{4}\right) = \frac{9}{8}.
\end{aligned}
$$

6 ADAM

Adam is the classic optimizer for neural-network training and was famously used to train GPT-3 and InstructGPT, the precursors to ChatGPT. Since 2024, matrix-aware optimizers such as Muon have become more common. Adam is effectively just a combination of the three ideas discussed so far: (a) preconditioning; (b) stochastic gradients; (c) momentum acceleration.

Let g_k be the stochastic gradient from before. Then ADAM is just

$$\begin{aligned} v_k &= \beta_1 v_{k-1} + (1-\beta_1) g_k, \\ x_k &= x_{k-1} - \eta_k B_k v_k, \end{aligned}$$

where the diagonal preconditioner B_k is constructed in a special way by accumulating the previous gradients

$$\begin{aligned} u_k &= \beta_2 u_{k-1} + (1-\beta_2)(g_k \odot g_k), \\ B_k &= \frac{1}{1-\beta_1^k}\left[\operatorname{diag}\left(\frac{u_k}{1-\beta_2^k}\right)^{1/2} + \varepsilon I\right]^{-1}. \end{aligned}$$

Question 4. What is the difference between ADAM and...

- SGD+momentum?
- SGD?
- Classical gradient descent?

Solution:

- SGD+momentum is ADAM with no preconditioning $B_k = I$.
- SGD is SGD+momentum with no momentum $\beta_1 = 0$.
- Classical GD is SGD but with full-batch gradients $g_k = \nabla f(x_k)$.

7 How to deal with constraints?

So far, all algorithms have been unconstrained.

What if we have constraints?

$$\min_{x_1,x_2} f(x_1, x_2) \text{ s.t. } x_1^2 + x_2^2 = 1.$$

Define unconstrained penalized objective.

Quadratic penalty method:

$$F_\mu(x) \coloneqq f(x) + \frac{1}{2\mu}(x_1^2 + x_2^2 - 1)^2$$

Question 5. Why do we square the constraint?

Solution:

- Need to penalize absolute value, or else constraint goes negative.
- Absolute value is not smooth. Squaring makes it smooth.
- Of all smoothing methods, squaring is simplest.

Question 6. What does the parameter μ do?

Solution:

- Decides how "sharp" the penalty is. Smoother as $\mu \to \infty$.
- Decides the amount of *constraint violation*. Smaller as $\mu \to 0$.

Question 7. Why doesn't this "solve" constrained optimization?

Solution:

- Does not find a solution that satisfies constraints $x_1^2 + x_2^2 = 1$, unless $\mu \to 0$.

8 What about inequality constraints?

What if we have *inequality* constraints?

$$\min_{x_1,x_2} f(x_1,x_2) \text{ s.t. } x_1^2+x_2^2 \le 1.$$

Option 1. Quadratic penalty method

$$F_\mu(x) \coloneqq f(x) + \frac{1}{2\mu}[(x_1^2+x_2^2-1)_+]^2$$

where $(t)_+ = \max\{t,0\}$.

Option 2. Logarithmic barrier method

$$F_\mu(x) \coloneqq f(x) - \mu\log(1-x_1^2-x_2^2).$$

Question 8. What is the difference?

Solution:

- Penalty is more permissive / optimistic; allows constraint violation.
- Barrier is stricter / pessimistic; requires strict feasibility and prevents crossing the constraint boundary.

Question 9. What does the parameter μ do?

Solution:

- Larger μ gives a stronger barrier; smaller μ gives a weaker barrier.
- As $\mu \to 0$, the barrier solution can approach the constraint boundary.

Question 10. Why doesn't this "solve" constrained optimization?

Solution:

- For every $\mu > 0$, the barrier solution remains strictly feasible and generally does not equal a boundary optimum. To approach the constrained optimum, decrease $\mu \to 0$.
- How do we find a strictly feasible starting point?

9 Convergence diagnostics

Common quantities to plot are

$$f(x_k), \qquad f(x_k) - f^\star, \qquad \|\nabla f(x_k)\|, \qquad \|x_k - x^\star\|.$$

They answer different questions.

$f(x_k)$	Is the objective decreasing?
$f(x_k) - f^\star$	How far is the objective value from optimal?
$\|\nabla f(x_k)\|$	Is the method approaching a stationary point?
$\|x_k - x^\star\|$	Is the iterate approaching a known reference solution?

For the penalty and barrier formulations above, plot $\|\nabla F_\mu(x_k)\|$; the gradient of the original constrained objective need not vanish at a constrained optimum.

A flat plot of $f(x_k)$ alone is not evidence of failure. If $f^\star$ is large, $f(x_k)$ can look visually stagnant even while the objective gap and gradient norm decay rapidly.