Chapter 8

Optimization basics

Contents (class version)

8.0 Introduction

Many of the previous topics have involved **optimization** formulations: linear LS, Procrustes, low-rank approximation, multidimensional scaling. In all these cases we derived analytical solutions, like the pseudo-inverse for minimum-norm LS problems and the truncated SVD for low-rank approximation.

But often we need iterative optimization algorithms, *e.g.*,

- if no closed-form minimizer exists, or
- if the analytical solution requires too much computation and/or memory, *e.g.*, SVD for large problems.

To solve a problem like $\hat{\boldsymbol{x}} = \arg\min_{\boldsymbol{x}} f(\boldsymbol{x})$ via an iterative method, we start with some initial guess $\boldsymbol{x}_0$, and then the algorithm produces a sequence $\{\boldsymbol{x}_k\}$ where hopefully the sequence **converges** to $\hat{\boldsymbol{x}}$, meaning $\|\boldsymbol{x}_k - \hat{\boldsymbol{x}}\| \to 0$ for some norm $\|\cdot\|$ as $k \to \infty$, as discussed in Ch. 5.

The homework has introduced a few such optimization algorithms. This chapter elaborates on those. Along the way we introduce several new matrix concepts: **matrix square root**, **matrix powers**, more about **positive (semi)definite matrices**, **commuting matrices**, and **majorization**.

8.1 Preconditioned gradient descent (PGD) for LS

For the (possibly regularized) LS problem with $\boldsymbol{A} \in \mathbb{F}^{M \times N}$ the **cost function** is **quadratic**:

$$\hat{\boldsymbol{x}} = \arg\min_{\boldsymbol{x}} f(\boldsymbol{x}), \quad \underbrace{f(\boldsymbol{x}) = \frac{1}{2} \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{y}\|_2^2}_{f \,:\, \mathbb{F}^N \mapsto \mathbb{R}}, \quad \underbrace{\nabla f(\boldsymbol{x}) = \boldsymbol{A}' (\boldsymbol{A}\boldsymbol{x} - \boldsymbol{y})}_{\nabla f \,:\, \mathbb{F}^N \mapsto \mathbb{F}^N}.$$

The homework problems focused on the classical **gradient descent** (**GD**) iterative method:

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \alpha \nabla f(\boldsymbol{x}_k) = \boldsymbol{x}_k - \alpha \boldsymbol{A}'(\boldsymbol{A}\boldsymbol{x}_k - \boldsymbol{y}),$$

where convergence of the sequence $\{\boldsymbol{x}_k\}$ is ensured (according to HW) by choosing the **step size** α to satisfy

$$0 < \alpha < \frac{2}{\sigma_1(\boldsymbol{A}'\boldsymbol{A})} = \frac{2}{\sigma_1^2(\boldsymbol{A})}. \tag{8.1}$$

Now consider a generalization called **preconditioned gradient descent** (**PGD**):

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \boldsymbol{P} \nabla f(\boldsymbol{x}_k) = \boldsymbol{x}_k - \boldsymbol{P}\boldsymbol{A}'(\boldsymbol{A}\boldsymbol{x}_k - \boldsymbol{y}), \tag{8.2}$$

where $\boldsymbol{P}$ is a $N \times N$ **preconditioning matrix**. Classical gradient descent simply uses $\boldsymbol{P} = \alpha \boldsymbol{I}_N$. But a single step size α may be suboptimal (and hard to choose), especially if different elements of $\boldsymbol{x}$ have different units.

What conditions on $\boldsymbol{P}$ ensure convergence? Will some other $\boldsymbol{P} \neq \alpha \boldsymbol{I}$ provide faster convergence? Why does (8.1) suffice? Answering these questions will use many matrix methods!

Tool: Matrix square root

We need another linear algebra tool first. (And more will come before we finish this topic.)

Define. We call $\boldsymbol{S}$ a **(matrix) square root** of a square matrix $\boldsymbol{A}$ iff $\boldsymbol{S}\boldsymbol{S} = \boldsymbol{A}$.

It is not unique: if $\boldsymbol{S}$ is a square root of $\boldsymbol{P}$, then $-\boldsymbol{S}$ is also square root of $\boldsymbol{P}$.

Example. For the rotation matrix $\boldsymbol{R} = \begin{bmatrix} \cos\phi & \sin\phi \\ -\sin\phi & \cos\phi \end{bmatrix}$ a square root is $\boldsymbol{S} = \begin{bmatrix} \cos(\phi/2) & \sin(\phi/2) \\ -\sin(\phi/2) & \cos(\phi/2) \end{bmatrix}$ because $\boldsymbol{S}\boldsymbol{S} = \boldsymbol{R}$. This example has an intuitive geometric interpretation.

If $\boldsymbol{P}$ is **positive semidefinite**, *i.e.*, $\boldsymbol{P} \succeq 0$, then $\boldsymbol{P}$ has a positive semidefinite square root.
Proof: $\boldsymbol{P} \succeq 0 \implies \boldsymbol{P} = \boldsymbol{V}\boldsymbol{\Lambda}\boldsymbol{V}'$, where the eigenvalues are all real and nonnegative,
so let $\boldsymbol{S} = \boldsymbol{V}\boldsymbol{\Lambda}^{1/2}\boldsymbol{V}'$ where $\boldsymbol{\Lambda}^{1/2} \triangleq \mathsf{diag}\left\{\sqrt{\lambda_i}\right\}$. Clearly $\boldsymbol{S}\boldsymbol{S} = \boldsymbol{P}$ and $\boldsymbol{S} \succeq \boldsymbol{0}$.

If $\boldsymbol{P}$ is **positive definite**, $\boldsymbol{P} \succ \boldsymbol{0}$, then $\boldsymbol{P}$ has a positive definite (and hence invertible) square root.

Proof: Same as above only now the eigenvalues are all positive.

Define. We write $\boldsymbol{P}^{1/2}$ to denote the positive (semi)definite square root of $\boldsymbol{P}$; we use this notation only if $\boldsymbol{P}$ is positive (semi)definite. When being careful, we call $\boldsymbol{P}^{1/2}$ the **principal square root**, but often we just call it "the square root" of $\boldsymbol{P}$, just like people say "the square root of 9 is 3."

Every **diagonalizable** matrix has a **matrix square root**.
A: True B: False $\boxed{??}$

Example. If $\boldsymbol{A} = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}$ then $\boldsymbol{S} = \begin{bmatrix} 1 & 1/2 \\ 0 & 1 \end{bmatrix}$ is a matrix square root of $\boldsymbol{A}$.

If a square matrix has a **matrix square root**, then it is **diagonalizable**.
A: True B: False $\boxed{??}$

More generally, let $\boldsymbol{A}$ be any 2×2 matrix. Even if $\boldsymbol{A}$ is not diagonalizable, we can still factor it using the **Jordan normal form**: as follows:

$$\boldsymbol{A} = \boldsymbol{V} \begin{bmatrix} \lambda & 1 \\ 0 & \lambda \end{bmatrix} \boldsymbol{V}^{-1}$$

where $\boldsymbol{V}$ is an invertible matrix consisting of **generalized eigenvectors**. One can verify that if $\lambda \neq 0$, then a square root of this matrix is

$$\boldsymbol{S} = \boldsymbol{V} \begin{bmatrix} \sqrt{\lambda} & \frac{1}{2\sqrt{\lambda}} \\ 0 & \sqrt{\lambda} \end{bmatrix} \boldsymbol{V}^{-1}.$$

This **Jordan form** approach to find a matrix square root generalizes to any square matrix having positive eigenvalues. But not every matrix has a square root.

Example. The matrix $\begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}$ has no square root.

Example.

For $\boldsymbol{x} \neq \boldsymbol{0}$, consider $\boldsymbol{A} = \boldsymbol{x}\boldsymbol{x}' \succeq \boldsymbol{0}$. Which of the following is a matrix square root of $\boldsymbol{A}$?

A: $\boldsymbol{x}\boldsymbol{x}'$ B: $\boldsymbol{x}\boldsymbol{x}' / \|\boldsymbol{x}\|_2$ C: $\boldsymbol{x}\boldsymbol{x}' / \|\boldsymbol{x}\|_2^2$ D: $\boldsymbol{x}\boldsymbol{x}' \|\boldsymbol{x}\|_2$ E: $\boldsymbol{x}\boldsymbol{x}' \|\boldsymbol{x}\|_2^2$

??

Example.

If $\boldsymbol{Q}$ is a unitary matrix, then $\boldsymbol{Q}$ has a matrix square root.

A: True B: False ??

??

Practical implementation of matrix square root

For a square matrix, the command for finding a **matrix square root** is: `sqrt(A)`

This operation differs *completely* from element-wise root: `sqrt.(A)`

In MATLAB (and old JULIA versions) the operation is `sqrtm(A)`

Caution. If $\boldsymbol{A}$ is square but not diagonalizable then the output of `sqrt` can be meaningless.
Try `sqrt([0 1; 0 0])` in JULIA or `sqrtm([0 1; 0 0])` in MATLAB.

Hereafter we assume the **preconditioner** $\boldsymbol{P}$ in PGD (8.2) is **positive definite**, *i.e.*, $\boldsymbol{P} \succ \mathbf{0}$, so that $\boldsymbol{P}^{1/2}$ exists and is invertible. We denote its inverse by $\boldsymbol{P}^{-1/2}$.

Convergence rate analysis of PGD: first steps

Let $\hat{\boldsymbol{x}}$ denote any minimizer of $f(\boldsymbol{x})$. That $\hat{\boldsymbol{x}}$ must satisfy the normal equations: $\boldsymbol{A}'\boldsymbol{A}\hat{\boldsymbol{x}} = \boldsymbol{A}'\boldsymbol{y}$. For analysis purposes, replace $\boldsymbol{A}'\boldsymbol{y}$ in (8.2) with $\boldsymbol{A}'\boldsymbol{A}\hat{\boldsymbol{x}}$:

$$\begin{aligned}
\boldsymbol{x}_{k+1} &= \boldsymbol{x}_k - \boldsymbol{P}\boldsymbol{A}'(\boldsymbol{A}\boldsymbol{x}_k - \boldsymbol{y}) = \boldsymbol{x}_k - \boldsymbol{P}(\boldsymbol{A}'\boldsymbol{A}\boldsymbol{x}_k - \boldsymbol{A}'\boldsymbol{y}) \\
&= \boldsymbol{x}_k - \boldsymbol{P}(\boldsymbol{A}'\boldsymbol{A}\boldsymbol{x}_k - \boldsymbol{A}'\boldsymbol{A}\hat{\boldsymbol{x}}) = \boldsymbol{x}_k - \boldsymbol{P}\boldsymbol{A}'\boldsymbol{A}(\boldsymbol{x}_k - \hat{\boldsymbol{x}}) \\
\Longrightarrow \boldsymbol{x}_{k+1} - \hat{\boldsymbol{x}} &= \boldsymbol{x}_k - \hat{\boldsymbol{x}} - \boldsymbol{P}\boldsymbol{A}'\boldsymbol{A}(\boldsymbol{x}_k - \hat{\boldsymbol{x}}) \\
&= \qquad\qquad \\
\Longrightarrow \underbrace{\boldsymbol{P}^{-1/2}(\boldsymbol{x}_{k+1} - \hat{\boldsymbol{x}})}_{\boldsymbol{\delta}_{k+1}} &= \boldsymbol{P}^{-1/2}(\boldsymbol{I} - \boldsymbol{P}\boldsymbol{A}'\boldsymbol{A})(\boldsymbol{x}_k - \hat{\boldsymbol{x}}) \\
&= \left(\boldsymbol{I} - \boldsymbol{P}^{1/2}\boldsymbol{A}'\boldsymbol{A}\boldsymbol{P}^{1/2}\right)\underbrace{\boldsymbol{P}^{-1/2}(\boldsymbol{x}_k - \hat{\boldsymbol{x}})}_{\boldsymbol{\delta}_k}
\end{aligned}$$

$$\Longrightarrow \boldsymbol{\delta}_{k+1} = \left(\boldsymbol{I} - \boldsymbol{P}^{1/2}\boldsymbol{A}'\boldsymbol{A}\boldsymbol{P}^{1/2}\right)\boldsymbol{\delta}_k = \boldsymbol{G}\boldsymbol{\delta}_k, \qquad \begin{aligned} \boldsymbol{\delta}_k &\triangleq \boldsymbol{P}^{-1/2}(\boldsymbol{x}_k - \hat{\boldsymbol{x}}) \ \text{(mod. error vector)} \\ \boldsymbol{G} &\triangleq \boldsymbol{I} - \boldsymbol{P}^{1/2}\boldsymbol{A}'\boldsymbol{A}\boldsymbol{P}^{1/2} \end{aligned}$$

$$\Longrightarrow \boldsymbol{\delta}_k = \qquad .$$

The matrix $\boldsymbol{G}$ "governs" the convergence of PGD.

Tool: Matrix powers

Now we need **matrix powers** to proceed, another easy byproduct of **eigendecomposition**.
Here $\boldsymbol{G} = \boldsymbol{G}'$, so $\boldsymbol{G} = \boldsymbol{V}\boldsymbol{\Lambda}\boldsymbol{V}'$ for some unitary $\boldsymbol{V}$ and $\boldsymbol{\Lambda}$ is real (but *not* nonnegative in general).
So $\boldsymbol{G}^2 = \boldsymbol{G}\boldsymbol{G} = \boldsymbol{V}\boldsymbol{\Lambda}\boldsymbol{V}'\boldsymbol{V}\boldsymbol{\Lambda}\boldsymbol{V}' = \boldsymbol{V}\boldsymbol{\Lambda}^2\boldsymbol{V}'$ and more generally $\boldsymbol{G}^k = \boldsymbol{V}\boldsymbol{\Lambda}^k\boldsymbol{V}', \ \forall k \in \mathbb{N}$.

If $\boldsymbol{A}$ is **diagonalizable** and **invertible**, is $\boldsymbol{A}^k$ diagonalizable and invertible for all $k \in \mathbb{N}$?

A: Yes B: No C: Depends **??**

PGD convergence condition using eigenvalues

So we can express the modified error vector $\boldsymbol{\delta}_k$ in terms of the initial error:

$$\boldsymbol{\delta}_k =$$

Define $\tilde{\boldsymbol{\delta}}_k \triangleq \boldsymbol{V}'\boldsymbol{\delta}_k$ to be the error coordinates in the $\boldsymbol{V}$ basis, then

$$\tilde{\boldsymbol{\delta}}_k =$$

We have reduced the question of convergence of PGD down to seeing whether $\{\lambda_i^k\}$ is a decreasing geometric series. We need $-1 < \lambda_i < 1$ for all $i = 1, \ldots, N$ to ensure that all error components diminish to zero.

Put concisely, a necessary and sufficient condition to ensure convergence of PGD from any initializer $\boldsymbol{x}_0$ is:

$$\rho(\boldsymbol{G}) = \rho\left(\boldsymbol{I} - \boldsymbol{P}^{1/2}\boldsymbol{A}'\boldsymbol{A}\boldsymbol{P}^{1/2}\right) < 1, \quad \textit{i.e.}, \tag{8.3}$$

Classical GD: step size bounds

Consider the classical case where $\boldsymbol{P} = \alpha \boldsymbol{I}$. Then the above condition (8.3) simplifies to

$$-1 < \mathsf{eig}\{\boldsymbol{I} - \alpha \boldsymbol{A}'\boldsymbol{A}\} < 1, \quad \Longleftrightarrow \qquad\qquad \Longleftrightarrow \qquad\qquad$$

where $\{\sigma_i\}$ denotes the singular values of $\boldsymbol{A}$.

The lower bound condition $0 < \alpha\sigma_i^2$ holds for all i if $0 < \alpha$ and if $\boldsymbol{A}$ has full column rank (*e.g.*, when Tikhonov regularization is used). So even though the analysis did not assume full rank at the start, if we want to make sure that all error modes diminish to zero, we must have full (column) rank matrix $\boldsymbol{A}$.

The upper bound condition $\alpha\sigma_i^2 < 2$ holds for all i if $\alpha\sigma_1^2 < 2$ because σ_1 is the largest, so we need $\alpha < 2/\sigma_1^2$.

In summary, we derived the step-size condition (8.1) for convergence of classical GD for LS problem:

$$0 < \alpha < \frac{2}{\sigma_1^2(\boldsymbol{A})} = \frac{2}{\sigma_1(\boldsymbol{A}'\boldsymbol{A})}.$$

Optimal step size for GD

For classical GD, where $\boldsymbol{P} = \alpha \boldsymbol{I}$, the *optimal* step size (for fastest convergence) is: (HW)

$$\alpha_* = \frac{2}{\sigma_1^2(\boldsymbol{A}) + \sigma_N^2(\boldsymbol{A})}. \tag{8.4}$$

This step size makes $\rho(\boldsymbol{I} - \alpha \boldsymbol{A}' \boldsymbol{A})$ as small as possible, *i.e.*, so that its largest absolute eigenvalue is as close to zero as possible. That largest eigenvalue will be the mode that converges to zero the slowest, so it will ultimately govern the convergence rate.

However, the step-size conditions (8.1) and (8.4) are unsatisfying practically because if a LS problem is so large that we cannot use the SVD to compute the pseudo-inverse, then probably we do not really want to use an SVD (or `svds`) to find $\sigma_1(\boldsymbol{A})$ and perhaps also $\sigma_N(\boldsymbol{A})$.

If $\boldsymbol{A}$ is unitary, what is the best step size α when $\boldsymbol{P} = \alpha \boldsymbol{I}$?

A: 0 B: 1 C: 1.99 D: 2 E: None of these ??

Suppose $\boldsymbol{P} = \alpha (\boldsymbol{A}' \boldsymbol{A})^{-1}$ for some $\alpha \geq 0$. What is the best value of α, so that the eigenvalues of $\boldsymbol{G}$ are as small (in magnitude) as possible, so PGD converges as fast as possible?

A: 0 B: 1/2 C: 1 D: $\sqrt{2}$ E: $2 - \epsilon$??

Practical step size for GD

One option for avoiding the SVD is to use the bound from HW5:

$$\sigma_1(\boldsymbol{A}) = \|\boldsymbol{A}\|_2 \leq \|\boldsymbol{A}\|_{\mathrm{F}}.$$

So choosing $0 < \alpha < 2/\|\boldsymbol{A}\|_{\mathrm{F}}^2$ always provides a valid step size. This is useful because it is easy to compute $\|\cdot\|_{\mathrm{F}}$ because no SVD is needed. However, the upper bound above is often quite loose. If $\boldsymbol{A} = \boldsymbol{I}_N$, then $\|\boldsymbol{A}\|_2 = 1$ but $\|\boldsymbol{A}\|_{\mathrm{F}} = \sqrt{N}$. So using this bound, the step size α would be much smaller than necessary.

Another option is to recall that because $\boldsymbol{A}'\boldsymbol{A}$ is symmetric:

$$\|\boldsymbol{A}'\boldsymbol{A}\|_2 = \sigma_1(\boldsymbol{A}'\boldsymbol{A}) = \rho(\boldsymbol{A}'\boldsymbol{A}) \leq \|\boldsymbol{A}'\boldsymbol{A}\|,$$

for any matrix norm $\|\cdot\|$. In particular, as discussed in Ch. 5, $\|\boldsymbol{A}'\boldsymbol{A}\|_1 \leq \|\boldsymbol{A}'\|_1 \|\boldsymbol{A}\|_1 = \|\boldsymbol{A}\|_\infty \|\boldsymbol{A}\|_1$.
So convergence is always ensured by choosing:

$$0 < \alpha < \frac{2}{\|\boldsymbol{A}'\boldsymbol{A}\|_1} \quad \text{or} \quad 0 < \alpha < \frac{2}{\|\boldsymbol{A}\|_\infty \|\boldsymbol{A}\|_1}.$$

Why did I choose $\|\boldsymbol{A}'\boldsymbol{A}\|_1$ instead of $\|\boldsymbol{A}'\boldsymbol{A}\|_\infty$?

Which norm is bigger:
A: $\|\boldsymbol{A}'\boldsymbol{A}\|_1$ usually B: $\|\boldsymbol{A}'\boldsymbol{A}\|_1$ always C: $\|\boldsymbol{A}'\boldsymbol{A}\|_\infty$ usually D: $\|\boldsymbol{A}'\boldsymbol{A}\|_\infty$ always E: They are the same.

$\boxed{\textbf{??}}$

Ideal preconditioner for PGD

Before looking at more general choices for the preconditioner $\boldsymbol{P}$, we first explore the "ideal" $\boldsymbol{P}$.

The ideal preconditioner is $\boldsymbol{P}_{\text{ideal}} = (\boldsymbol{A}'\boldsymbol{A})^{-1}$,
because for that choice:

$$\boldsymbol{G} = \boldsymbol{I} - \boldsymbol{P}_{\text{ideal}}^{1/2}\boldsymbol{A}'\boldsymbol{A}\boldsymbol{P}_{\text{ideal}}^{1/2} =$$

so $\boldsymbol{\delta}_{(1)} = \boldsymbol{G}^1\boldsymbol{\delta}_0 = \mathbf{0}\boldsymbol{\delta}_0 = \mathbf{0}$, *i.e.*, PGD converges in one iteration:

$$\boldsymbol{x}_1 = \boldsymbol{x}_0 - \boldsymbol{P}_{\text{ideal}}\boldsymbol{A}'(\boldsymbol{A}\boldsymbol{x}_0 - \boldsymbol{y}) = \boldsymbol{x}_0 - (\boldsymbol{A}'\boldsymbol{A})^{-1}\boldsymbol{A}'(\boldsymbol{A}\boldsymbol{x}_0 - \boldsymbol{y}) = (\boldsymbol{A}'\boldsymbol{A})^{-1}\boldsymbol{A}'\boldsymbol{y} = \boldsymbol{A}^+\boldsymbol{y}.$$

Why not always use this ideal preconditioner?
Inverting $\boldsymbol{A}'\boldsymbol{A}$ is expensive: it is as hard as the original LS problem. So we look for other preconditioners.
Specifically we would like to find a preconditioner $\boldsymbol{P} \approx (\boldsymbol{A}'\boldsymbol{A})^{-1}$ or $\boldsymbol{P}^{-1} \approx \boldsymbol{A}'\boldsymbol{A}$ but where the inverse is much easier to compute.
For that we need another tool.

Tool: Positive (semi)definiteness properties

Recall that we define positive (semi)definiteness only for Hermitian matrices and:

- **positive semidefinite**: $\boldsymbol{A} \succeq \mathbf{0} \iff \boldsymbol{x}'\boldsymbol{A}\boldsymbol{x} \geq 0, \ \forall \boldsymbol{x}$
- **positive definite**: $\boldsymbol{A} \succ \mathbf{0} \iff \boldsymbol{x}'\boldsymbol{A}\boldsymbol{x} > 0, \ \forall \boldsymbol{x} \neq \mathbf{0}$

Properties ($\boldsymbol{A}$ and $\boldsymbol{B}$ are all Hermitian here):

- $\boldsymbol{X}'\boldsymbol{X} \succeq \mathbf{0}$ for *any* matrix $\boldsymbol{X}$, even non-square (shown previously)
- $\boldsymbol{A} \succeq \mathbf{0} \implies \boldsymbol{X}'\boldsymbol{A}\boldsymbol{X} \succeq \mathbf{0}$, for any matrix $\boldsymbol{X}$ with `size(A,2) == size(X,1)`
- $\boldsymbol{A} \succeq \mathbf{0}$ and $\alpha \geq 0 \implies \alpha\boldsymbol{A} \succeq \mathbf{0}$
- $\boldsymbol{A} \succeq \mathbf{0} \implies$ all eigenvalues of $\boldsymbol{A}$ are real and nonnegative
  Proof 1: if $\boldsymbol{x}$ is an eigenvector of $\boldsymbol{A}$ then $0 \leq \boldsymbol{x}'(\boldsymbol{A}\boldsymbol{x}) = \boldsymbol{x}'\lambda\boldsymbol{x} = \lambda \|\boldsymbol{x}\|_2^2 \implies \lambda \geq 0$
  Proof 2: $\boldsymbol{A}$ Hermitian implies it has an orthogonal eigendecomposition: $\boldsymbol{A} = \boldsymbol{V}\boldsymbol{\Lambda}\boldsymbol{V}'$ so $\boldsymbol{V}\boldsymbol{\Lambda}\boldsymbol{V}' \succeq \mathbf{0}$.
  Using 2nd property above: $\implies \boldsymbol{V}'(\boldsymbol{V}\boldsymbol{\Lambda}\boldsymbol{V}')\boldsymbol{V} \succeq \mathbf{0} \implies \boldsymbol{\Lambda} \succeq \mathbf{0} \implies \boldsymbol{e}_j'\boldsymbol{\Lambda}\boldsymbol{e}_j \geq 0 \implies \lambda_j \geq 0$.
- $\boldsymbol{A} \succ \mathbf{0} \implies \boldsymbol{A}$ invertible and $\boldsymbol{A}^{-1} \succ \mathbf{0}$
- $\boldsymbol{B} \succeq \boldsymbol{A} \iff \boldsymbol{B} - \boldsymbol{A} \succeq \mathbf{0}$ (trivial from definition of $\succeq$)
- $\boldsymbol{B} \succ \boldsymbol{A} \iff \boldsymbol{B} - \boldsymbol{A} \succ \mathbf{0}$
- $\boldsymbol{B} \succeq \boldsymbol{A} \succ \mathbf{0} \implies \boldsymbol{A}^{-1} \succeq \boldsymbol{B}^{-1} \succ \mathbf{0}$
- $\boldsymbol{B} \succeq \boldsymbol{A} \succ \mathbf{0} \implies \gamma\boldsymbol{B} \succ \boldsymbol{A}, \ \forall \gamma > 1$
- $\boldsymbol{A} \succ \mathbf{0}, \ \boldsymbol{B} \succ \mathbf{0} \implies \boldsymbol{B}\boldsymbol{A}\boldsymbol{B} \succ \mathbf{0}$ (HW)

Notice the pattern: start with a definition, then develop properties (especially addition and multiplication).

The above properties relate to multiplication; now consider **matrix addition**.

If $\boldsymbol{A} \succeq 0$ and $\boldsymbol{B} \succeq 0$ have the same size, then $\boldsymbol{A} + \boldsymbol{B} \succeq 0$. (?)

A: True B: False $\boxed{\textbf{??}}$

If $\boldsymbol{A} \succ 0$ and $\boldsymbol{B} \succeq 0$ have the same size, then $\boldsymbol{A} + \boldsymbol{B} \succ 0$. (?)

A: True B: False $\boxed{\textbf{??}}$

The above properties are all useful and important for GD for LS because:

- we choose $\boldsymbol{P} \succ \mathbf{0}$
- and we assume $\boldsymbol{A}'A \succ \mathbf{0}$.

General preconditioners for PGD

Repeating (8.3), for convergence we want

$$-1 < \mathsf{eig}\left\{ \boldsymbol{I} - \boldsymbol{P}^{1/2} \boldsymbol{A}' \boldsymbol{A} \boldsymbol{P}^{1/2} \right\} < 1. \tag{8.5}$$

Now the RHS of (8.5) is easy; assuming $\boldsymbol{A}'\boldsymbol{A}$ and $\boldsymbol{P}$ are positive definite:

$$\mathsf{eig}\left\{ \boldsymbol{I} - \boldsymbol{P}^{1/2} \boldsymbol{A}' \boldsymbol{A} \boldsymbol{P}^{1/2} \right\} = 1 - \underbrace{\mathsf{eig}\left\{ \boldsymbol{P}^{1/2} \boldsymbol{A}' \boldsymbol{A} \boldsymbol{P}^{1/2} \right\}}_{>0} < 1.$$

For the LHS of (8.5), we want $-1 < \mathsf{eig}\left\{ \boldsymbol{I} - \boldsymbol{P}^{1/2} \boldsymbol{A}' \boldsymbol{A} \boldsymbol{P}^{1/2} \right\} = 1 - \mathsf{eig}\left\{ \boldsymbol{P}^{1/2} \boldsymbol{A}' \boldsymbol{A} \boldsymbol{P}^{1/2} \right\}$, *i.e.*, (HW)

$$\mathsf{eig}\left\{ \boldsymbol{P}^{1/2} \boldsymbol{A}' \boldsymbol{A} \boldsymbol{P}^{1/2} \right\} < 2 \iff 2\boldsymbol{I} \succ \boldsymbol{P}^{1/2} \boldsymbol{A}' \boldsymbol{A} \boldsymbol{P}^{1/2} \iff 2\boldsymbol{P}^{-1} \succ \boldsymbol{A}'\boldsymbol{A}.$$

Matrix majorizers

Define. We say that a (Hermitian) symmetric matrix $\boldsymbol{A}$ is a **majorizer** of (or **majorizes**) a (Hermitian) symmetric matrix $\boldsymbol{B}$ iff $\boldsymbol{A} \succeq \boldsymbol{B}$, *i.e.*, iff $\boldsymbol{A} - \boldsymbol{B}$ is positive semidefinite.

Summarizing: if $\beta \boldsymbol{P}^{-1}$ **majorizes** $\boldsymbol{A}'\boldsymbol{A}$ for some $0 < \beta < 2$, then convergence of PGD is ensured, because $\beta \boldsymbol{P}^{-1} \succeq \boldsymbol{A}'\boldsymbol{A} \Longrightarrow 2\boldsymbol{P}^{-1} \succ \boldsymbol{A}'\boldsymbol{A}$.

Diagonal majorizer

Fact (diagonal majorizer). If $\boldsymbol{B}$ is (Hermitian) symmetric and $N \times N$, then: (HW)

$$\operatorname{diag}\{|\boldsymbol{B}|\,\mathbf{1}_N\} \succeq \boldsymbol{B}, \tag{8.6}$$

where here $|\boldsymbol{B}|$ denotes the element-wise absolute value of $\boldsymbol{B}$, *i.e.*, `abs.(B)` in JULIA.

Thus, a valid preconditioner for PGD is $\boldsymbol{P}^{-1} = \alpha^{-1} \operatorname{diag}\{|\boldsymbol{A}'\boldsymbol{A}|\,\mathbf{1}_N\}$ for any $0 < \alpha < 2$, leading to the **diagonally preconditioned GD** iteration:

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \alpha \boldsymbol{D} \boldsymbol{A}' (\boldsymbol{A}\boldsymbol{x}_k - \boldsymbol{y}), \quad \boldsymbol{D} \triangleq \operatorname{diag}\{|\boldsymbol{A}'\boldsymbol{A}|\,\mathbf{1}_N\}^{-1}, \quad 0 < \alpha < 2. \tag{8.7}$$

Typically we use $1 \leq \alpha \leq 1.99$ and it is easy to adjust α in this range to find good value.

If $\boldsymbol{A} = \begin{bmatrix} 1 & 0 \\ 0 & 2 \\ 1 & -1 \end{bmatrix}$ then what is the diagonal preconditioner $\boldsymbol{D}$ in (8.7) ?

A: $\begin{bmatrix} 1/3 & 0 \\ 0 & 1/6 \end{bmatrix}$ B: $\begin{bmatrix} 1/2 & 0 \\ 0 & 1/3 \end{bmatrix}$ C: $\begin{bmatrix} 1/2 & 0 \\ 0 & 1/5 \end{bmatrix}$ D: $\begin{bmatrix} 2 & 0 \\ 0 & 5 \end{bmatrix}$ E: None of these.

$\boxed{\textbf{??}}$

The "normalization" by the inverse matrix $\boldsymbol{D}$ in (8.7) makes finding α much easier than in classical GD where changing the units of the problem (units of $\boldsymbol{x}$ or $\boldsymbol{y}$ and hence $\boldsymbol{A}$) necessitate finding a new α value.

The following code compares eig$\left\{\boldsymbol{I} - \boldsymbol{P}^{-1/2}\boldsymbol{A}'\boldsymbol{A}\boldsymbol{P}^{-1/2}\right\}$ for

- classical GD with $\boldsymbol{P} = \alpha_* \boldsymbol{I}$ for the optimal step size α_*,
- versus diagonally preconditioned GD with $\alpha = 1.3$ chosen empirically.

The former has eigenvalues ± 0.5151, and the latter has eigenvalues $(-0.3, 0.35)$ so at least in this example the diagonal preconditioner accelerates convergence, without requiring an eigendecomposition to find $\boldsymbol{P}$.

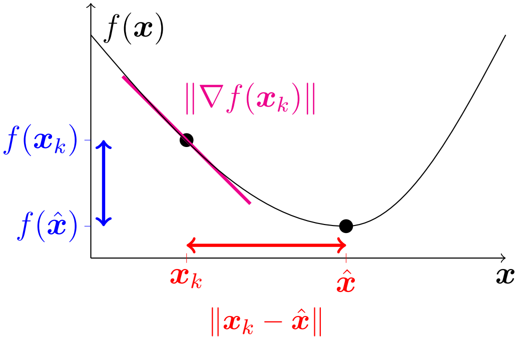
```
using LinearAlgebra
A = [1 0; 0 2; 1 -1]
@show A'A
D = Diagonal(1 ./ (abs.(A'A) * ones(2))) # diagonal preconditioner
alpha1 = 1.3
P1 = alpha1 * D
G1 = I - sqrt(P1) * (A' * A) * sqrt(P1)
@show eigvals(G1)

alphabest = 2 / (sum(svdvals(A'A)[[1, end]])) # optimal GD step size
G2 = I - alphabest * (A' * A)
@show eigvals(G2)
```

We have focused on the quadratic problem here, but the principles of diagonal majorization apply to using GD (and accelerated GD) in general for convex optimization of cost functions having Lipschitz gradients [1].

Convergence rates

There are many ways to assess the convergence rate of an iterative algorithm like PGD. Researchers study:

- $f(\boldsymbol{x}_k) \to f(\hat{\boldsymbol{x}})$
- $\|\nabla f(\boldsymbol{x}_k)\| \to 0$
- $\|\boldsymbol{x}_k - \hat{\boldsymbol{x}}\| \to 0$
- $\|\boldsymbol{\delta}_k\| \to 0, \quad \boldsymbol{\delta}_k \triangleq \boldsymbol{P}^{-1/2}(\boldsymbol{x}_k - \hat{\boldsymbol{x}})$
- $\left\|\tilde{\boldsymbol{\delta}}_k\right\| \to 0, \quad \tilde{\boldsymbol{\delta}}_k \triangleq \boldsymbol{V}'\boldsymbol{\delta}_k$



$f(\boldsymbol{x})$
$\|\nabla f(\boldsymbol{x}_k)\|$
$f(\boldsymbol{x}_k)$
$f(\hat{\boldsymbol{x}})$
$\boldsymbol{x}_k$
$\hat{\boldsymbol{x}}$
$\boldsymbol{x}$
$\|\boldsymbol{x}_k - \hat{\boldsymbol{x}}\|$

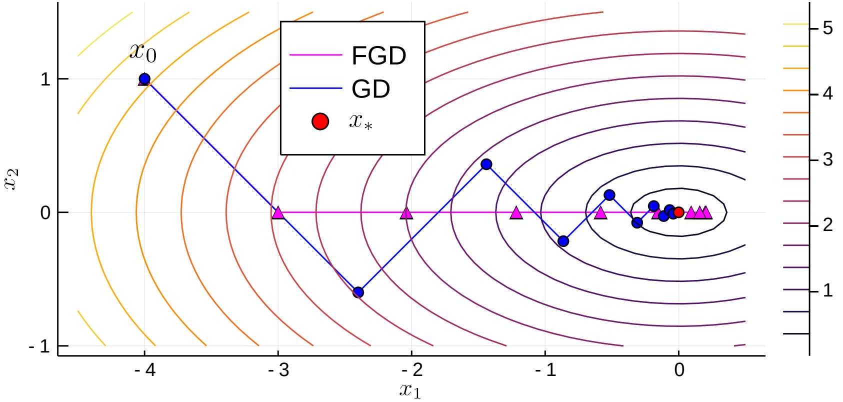
Quantifying bounds on the rates of decrease of these quantities is an active research area. Even classical GD has relatively recent results [2] that tighten up the traditional bounds. The tightest possible worst-case bound for GD for the decrease of the cost function (with a fixed step size $\alpha = 1/L$) is $O(1/k)$:

$$f(\boldsymbol{x}_k) - f(\hat{\boldsymbol{x}}) \leq \frac{\|\boldsymbol{x}_0 - \hat{\boldsymbol{x}}\|_2^2}{L(4k+2)},$$

where L is the Lipschitz constant of the gradient $\nabla f(\boldsymbol{x})$.

In contrast, Nesterov's fast gradient method has a worst-case cost function decrease at rate at least $O(1/k^2)$, which can be improved (and has) by only a constant factor [3].

Example. The following figure illustrates how slow GD can converge for a simple LS problem with $\boldsymbol{A} = \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix}$ and $\boldsymbol{y} = \boldsymbol{0}$. This case used the optimal step size α_* for illustration.
This slow convergence has been the impetus of thousands of papers on faster algorithms!



The ellipses show the contours of the LS cost function $\|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{y}\|$.
Also shown is Nesterov's fast gradient descent (FGD) method for comparison.

What is the optimal step size α_* here?

A: 1/5 B: 2/5 C: 3/5 D: 1/3 E: 2/3 $\boxed{??}$

Tool: commuting (square) matrices

Let $\boldsymbol{X}$ and $\boldsymbol{Y}$ denote square matrices of the same size.
Which of the following conditions guarantee that $\boldsymbol{X}$ and $\boldsymbol{Y}$ commute, *i.e.*, $\boldsymbol{X}\boldsymbol{Y} = \boldsymbol{Y}\boldsymbol{X}$.
Choose the most general correct combination of condition(s).

- 1 $\boldsymbol{X}$ diagonalizable
- 2 $\boldsymbol{Y}$ diagonalizable
- 3 $\boldsymbol{X}$ and $\boldsymbol{Y}$ have same eigenvectors (*i.e.*, are simultaneously diagonalizable)
- 4 $\boldsymbol{X}$ (Hermitian) symmetric
- 5 $\boldsymbol{Y}$ (Hermitian) symmetric

A: 1 & 2
B: 1 & 2 & 3
C: 1 & 2 & 3 & (4 or 5)
D: 1 & 2 & 3 & 4 & 5
E: None of these suffice.

$\boxed{\textbf{??}}$

All $N \times N$ **companion** matrices commute. (?)

A: True B: False $\boxed{\textbf{??}}$

$\boxed{\textbf{??}}$

All $N \times N$ **circulant** matrices commute. (?)

A: True B: False $\boxed{\textbf{??}}$

If $\boldsymbol{X}$ and $\boldsymbol{Y}$ are positive semidefinite and are simultaneously diagonalizable then

$$\boldsymbol{X}\boldsymbol{Y} = \boldsymbol{Y}\boldsymbol{X} = \boldsymbol{X}^{1/2}\boldsymbol{Y}\boldsymbol{X}^{1/2} = \boldsymbol{Y}^{1/2}\boldsymbol{X}\boldsymbol{Y}^{1/2}.$$

(?)

A: True B: False $\boxed{\textbf{??}}$

Being simultaneously diagonalizable is a sufficient condition. A necessary (but not sufficient) condition for **commuting matrices** is that they are **simultaneously triangularizable**.

Example. The matrices $\begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}$ and $\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$ commute, but the first is not diagonalizable.

Monotonicity

Some algorithms get closer to the solution every iteration, and/or decrease the cost function every iteration, whereas other algorithms have no such guarantee.

For PGD, there are many quantities we can evaluate to see if they do (or do not) **decrease monotonically**, where to be precise we really mean "non-increasing" but usually we say "decreasing" for brevity:

- $f(\boldsymbol{x}_{k+1}) \overset{?}{\leq} f(\boldsymbol{x}_k)$ cost function
- $\|\nabla f(\boldsymbol{x}_{k+1})\| \overset{?}{\leq} \|\nabla f(\boldsymbol{x}_k)\|$ gradient norm
- $\|\boldsymbol{x}_{k+1} - \hat{\boldsymbol{x}}\| \overset{?}{\leq} \|\boldsymbol{x}_k - \hat{\boldsymbol{x}}\|$ distance to solution in natural coordinates
- $\|\boldsymbol{\delta}_{k+1}\| \overset{?}{\leq} \|\boldsymbol{\delta}_k\|, \quad \boldsymbol{\delta}_k \triangleq \boldsymbol{P}^{-1/2} (\boldsymbol{x}_k - \hat{\boldsymbol{x}})$ distance to solution with a $\boldsymbol{P}$-related coordinate transform
- $\left\|\tilde{\boldsymbol{\delta}}_{k+1}\right\| \overset{?}{\leq} \left\|\tilde{\boldsymbol{\delta}}_k\right\|, \quad \tilde{\boldsymbol{\delta}}_k \triangleq \boldsymbol{V}'\boldsymbol{\delta}_k$ distance to solution under the $\boldsymbol{V}$ transform

When one of the above inequalities hold (typically with strict inequality for $\boldsymbol{x}_k \neq \hat{\boldsymbol{x}}$ we say the algorithm is **monotonic** or **monotone** but to be complete we should also qualify that by saying in which sense it is monotone (*e.g.*, cost function, gradient norm, distance to solution).

The easiest of the above list to analyze for PGD is the last one.

Recall $\tilde{\boldsymbol{\delta}}_{k+1} = \boldsymbol{\Lambda}\tilde{\boldsymbol{\delta}}_k$. If $\tilde{\boldsymbol{\delta}}_k \neq \mathbf{0}$, then because we choose $\boldsymbol{P}$ so that $-1 < \lambda_n < 1$ per (8.5), the distance to the solution diminishes monotonically in these coordinates:

$$\left\|\tilde{\boldsymbol{\delta}}_{k+1}\right\|_2 = \left\|\boldsymbol{\Lambda}\tilde{\boldsymbol{\delta}}_k\right\|_2 \leq \|\boldsymbol{\Lambda}\|_2 \left\|\tilde{\boldsymbol{\delta}}_k\right\|_2 < \left\|\tilde{\boldsymbol{\delta}}_k\right\|_2 .$$

If $\boldsymbol{P} \succ 0$ and $2\boldsymbol{P}^{-1} \succ \boldsymbol{A}'\boldsymbol{A} \succ 0$, does $\|\boldsymbol{\delta}_k\|_2$ decrease monotonically?

A: Yes. B: Not necessarily. $\boxed{??}$

Under the same conditions, when $\boldsymbol{x}_k \neq \hat{\boldsymbol{x}}$ does the distance $\|\boldsymbol{x}_k - \hat{\boldsymbol{x}}\|_2$ decrease monotonically? Choose most general correct answer.

A: Yes.
B: Yes, if the right singular vectors of $\boldsymbol{A}$ are eigenvectors of $\boldsymbol{P}$.
C: Yes, if the left singular vectors of $\boldsymbol{A}$ are eigenvectors of $\boldsymbol{P}$.
D: Yes, if $\boldsymbol{P} = \alpha\boldsymbol{I}$ for suitably chosen α.
E: No.

$\boxed{??}$

Hint: $\|\boldsymbol{x}_{k+1} - \hat{\boldsymbol{x}}\|_2 = \|(\boldsymbol{I} - \boldsymbol{P}\boldsymbol{A}'\boldsymbol{A})(\boldsymbol{x}_k - \hat{\boldsymbol{x}})\|_2 \leq \|\boldsymbol{I} - \boldsymbol{P}\boldsymbol{A}'\boldsymbol{A}\|_2 \|\boldsymbol{x}_k - \hat{\boldsymbol{x}}\|_2$

8.2 Preconditioned steepest descent

(Read)

Instead of using GD with a fixed step size α, an alternative is to do a **line search** to find the best step size *at each iteration*. This variation is called **steepest descent** (or GD with a line search). Here is how **preconditioned steepest descent** for a linear LS problem works:

$$\begin{aligned} \boldsymbol{d}_k &= -\boldsymbol{P}\nabla f(\boldsymbol{x}_k) = -\boldsymbol{P}\boldsymbol{A}'(\boldsymbol{A}\boldsymbol{x}_k - \boldsymbol{y}) && \text{search direction (negative preconditioned gradient)} \\ \alpha_k &= \arg\min_{\alpha} f(\boldsymbol{x}_k + \alpha \boldsymbol{d}_k) && \text{step size} \\ \boldsymbol{x}_{k+1} &= \boldsymbol{x}_k + \alpha_k \boldsymbol{d}_k && \text{update.} \end{aligned}$$

- Finding α_k is a HW problem
- By construction, this iteration is guaranteed to decrease the cost function monotonically: $f(\boldsymbol{x}_{k+1}) \leq f(\boldsymbol{x}_k)$ with strict decrease unless $\boldsymbol{x}_k$ is already a minimizer, provided the preconditioner $\boldsymbol{P}$ is **positive definite**.
- Computing α_k takes some extra work, especially for non-quadratic problems. Often Nesterov's fast gradient method or the **optimized gradient method** (**OGM**) [3] are preferable because they do not require a line search (if the Lipschitz constant is available).

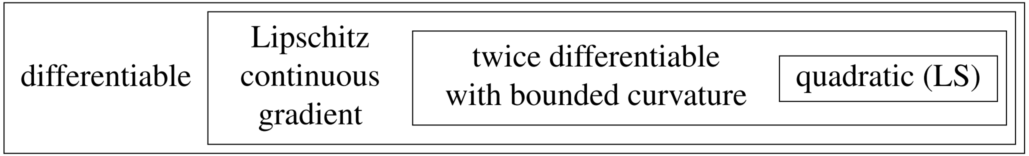
8.3 Gradient descent for smooth convex functions

So far we used (preconditioned) **gradient descent** (**PGD**) only for LS problems.
We often need to solve more general (non LS) unconstrained minimization problems:

$$\hat{\boldsymbol{x}} = \mathop{\arg\min}_{\boldsymbol{x} \in \mathbb{R}^N} f(\boldsymbol{x}).$$

What algorithm we use depends in part on the properties of the cost function $f : \mathbb{R}^N \mapsto \mathbb{R}$.

Venn diagram for convex functions:



differentiable
Lipschitz continuous gradient
twice differentiable with bounded curvature
quadratic (LS)

GD is applicable to the broader family of **convex** functions having **gradients** that are **Lipschitz continuous**.
We call these **smooth convex** functions.

Define. A differentiable function $f(\boldsymbol{x})$ has a **Lipschitz continuous gradient** if there exists $L < \infty$ such that

$$\|\nabla f(\boldsymbol{x}) - \nabla f(\boldsymbol{z})\|_2 \leq L \, \|\boldsymbol{x} - \boldsymbol{z}\|_2 \,, \quad \forall \boldsymbol{x}, \boldsymbol{z} \in \mathbb{R}^N.$$

Example. For $f(\boldsymbol{x}) = \frac{1}{2} \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{y}\|_2^2$ we have $\|\nabla f(\boldsymbol{x}) - \nabla f(\boldsymbol{z})\|_2 = \|\boldsymbol{A}'(\boldsymbol{A}\boldsymbol{x} - \boldsymbol{y}) - \boldsymbol{A}'(\boldsymbol{A}\boldsymbol{z} - \boldsymbol{y})\|_2$
$= \|\boldsymbol{A}'\boldsymbol{A}(\boldsymbol{x} - \boldsymbol{z})\|_2 \leq \|\!|\boldsymbol{A}'\boldsymbol{A}|\!\|_2 \, \|\boldsymbol{x} - \boldsymbol{z}\|_2 \,.$
So the Lipschitz constant of ∇f is $L = \|\!|\boldsymbol{A}'\boldsymbol{A}|\!\|_2 = \|\!|\boldsymbol{A}|\!\|_2^2 = \sigma^2(\boldsymbol{A}) = \rho(\boldsymbol{A}'\boldsymbol{A})$.

Fact. If $f(\boldsymbol{x})$ is **twice differentiable** and if there exists $L < \infty$ such that its **Hessian matrix** has a bounded **spectral norm**:

$$\left\| \nabla^2 f(\boldsymbol{x}) \right\|_2 \leq L, \quad \forall \boldsymbol{x} \in \mathbb{R}^N,$$

then $f(\boldsymbol{x})$ has a **Lipschitz continuous gradient** with Lipschitz constant L.

So twice differentiability with **bounded curvature** is sufficient, but not necessary, for a function to have Lipschitz continuous gradient.

Proof. Using **Taylor's theorem** and the **triangle inequality** and the definition of **spectral norm**:

$$\begin{aligned}
\|\nabla f(\boldsymbol{x}) - \nabla f(\boldsymbol{z})\|_2 &= \left\| \int_0^1 \nabla^2 f(\boldsymbol{x} + \tau(\boldsymbol{z} - \boldsymbol{x})) \, \mathrm{d}\tau \, (\boldsymbol{x} - \boldsymbol{z}) \right\|_2 \\
&\leq \left(\int_0^1 \left\| \nabla^2 f(\boldsymbol{x} + \tau(\boldsymbol{z} - \boldsymbol{x})) \right\|_2 \mathrm{d}\tau \right) \|\boldsymbol{z} - \boldsymbol{z}\|_2 \\
&\leq \left(\int_0^1 L \, \mathrm{d}\tau \right) \|\boldsymbol{x} - \boldsymbol{z}\|_2 = L \, \|\boldsymbol{x} - \boldsymbol{z}\|_2 .
\end{aligned}$$

Example. $f(\boldsymbol{x}) = \frac{1}{2} \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{y}\|_2^2 \Longrightarrow \nabla^2 f = \boldsymbol{A}'\boldsymbol{A}$ so $\|\nabla^2 f\|_2 = \|\boldsymbol{A}'\boldsymbol{A}\|_2$.

The Lipschitz constant for the gradient of $f(\boldsymbol{x}) \triangleq \boldsymbol{x}' \begin{bmatrix} 1 & 2 \\ 2 & 4 \end{bmatrix} \boldsymbol{x}$ is:

A: 1 B: 2 C: 4 D: 5 E: 10 $\boxed{??}$

Convexity and Hessian

If $f(\boldsymbol{x})$ is twice differentiable, then $f(\boldsymbol{x})$ is **convex** iff its Hessian $\nabla^2 f(\boldsymbol{x})$ is **positive semidefinite** for all $\boldsymbol{x}$.

Example. $f(\boldsymbol{x}) = \frac{1}{2} \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{y}\|_2^2 \Longrightarrow \nabla^2 f(\boldsymbol{x}) = \boldsymbol{A}'\boldsymbol{A} \succeq \boldsymbol{0} \Longrightarrow f(\boldsymbol{x})$ is convex for any $\boldsymbol{A}$.

One can also show convexity of this $f(\boldsymbol{x})$ directly from the definition of convex functions.

If $f(\boldsymbol{x})$ is twice differentiable, then $f(\boldsymbol{x})$ is **strictly convex** if its Hessian $\nabla^2 f(\boldsymbol{x})$ is **positive definite** for all $\boldsymbol{x}$.

Why not only if? $f(x) = x^4$ is strictly convex but $\ddot{f}(0) = 0$.

Example. If $\boldsymbol{A}$ has full column rank, then $\boldsymbol{A}'\boldsymbol{A}$ is positive definite so $f(\boldsymbol{x}) = \frac{1}{2} \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{y}\|_2^2$ is strictly convex.

Suppose $\boldsymbol{A}$ is a wide matrix and consider the regularized LS cost function $f(\boldsymbol{x}) \triangleq \frac{1}{2} \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{y}\|_2^2 + \beta \|\boldsymbol{x}\|_2^2$ where $\beta > 0$. Then f is a **strictly convex** function. (?)

A: True B: False $\boxed{??}$

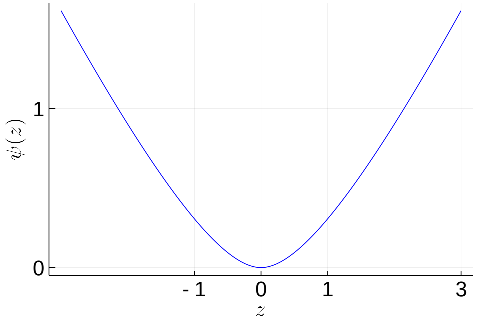
Example. The **Fair potential** used in many imaging applications [4] [5] is

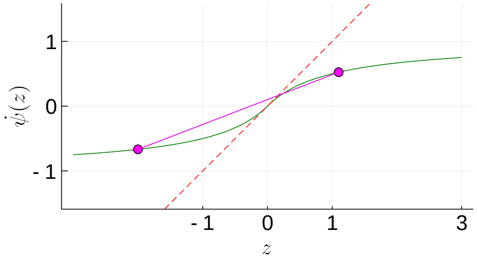
$$\psi(x) = |x| - \log(1 + |x|),$$

and has the property of being roughly quadratic for $x \approx 0$ and roughly like $|x|$ for $|x| \gg 0$.
For this function:

$$\dot{\psi}(x) = \frac{x}{1+|x|} \text{ and } \ddot{\psi}(x) = \frac{1}{(1+|x|)^2} \leq 1,$$

so the Lipschitz constant of the derivative of $\psi(\cdot)$ is 1. Furthermore, its second derivative is nonnegative so it is a convex function.





Is the Fair potential ψ itself **Lipschitz continuous**?

A: No B: Yes with $L = 1/2$ C: Yes with $L = 1$ D: Yes with $L = 2$ $\boxed{??}$

GD convergence theorem

- If **convex** function $f(\boldsymbol{x})$ has a (not necessarily unique) minimizer $\hat{\boldsymbol{x}}$ for which

$$-\infty < f(\hat{\boldsymbol{x}}) \leq f(\boldsymbol{x}), \quad \forall \boldsymbol{x} \in \mathbb{R}^N,$$

- the gradient of $f(\boldsymbol{x})$ is **Lipschitz continuous**, *i.e.*,

$$\|\nabla f(\boldsymbol{x}) - \nabla f(\boldsymbol{z})\|_2 \leq L \, \|\boldsymbol{x} - \boldsymbol{z}\|_2 \, , \forall \boldsymbol{x}, \boldsymbol{z} \in \mathbb{R}^N,$$

- the **step size** α is chosen such that

$$0 < \alpha < 2/L,$$

then the GD iteration

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \alpha \nabla f(\boldsymbol{x}_k)$$

has the following convergence properties [6, p. 207]:

- the cost function is non-increasing: $f(\boldsymbol{x}_{k+1}) \leq f(\boldsymbol{x}_k)$
- for any minimizer $\hat{\boldsymbol{x}}$, the distance to that minimizer is non-increasing: $\|\boldsymbol{x}_{k+1} - \hat{\boldsymbol{x}}\| \leq \|\boldsymbol{x}_k - \hat{\boldsymbol{x}}\|$
- the sequence $\{\boldsymbol{x}_k\}$ **converges** to a minimizer of $f(\cdot)$.
- For $0 < \alpha \leq 1/L$, the cost function decrease is bounded by [2]:

$$f(\boldsymbol{x}_k) - f(\hat{\boldsymbol{x}}) \leq \frac{L \, \|\boldsymbol{x}_0 - \hat{\boldsymbol{x}}\|_2^2}{2} \max\left(\frac{1}{2k\alpha + 1}, (1 - \alpha)^{2k} \right).$$

  This upper bound is conjectured to also hold for $1/L < \alpha < 2/L$ [7].

Convex sets

Define. A nonempty set $\mathcal{C}$ in a vector space $\mathcal{V}$ is a **convex set** iff

$$\boldsymbol{x}, \boldsymbol{z} \in \mathcal{C} \Longrightarrow \alpha \boldsymbol{x} + (1-\alpha) \boldsymbol{z} \in \mathcal{C}, \quad \forall 0 \leq \alpha \leq 1.$$

The linear combination $\alpha \boldsymbol{x} + (1-\alpha) \boldsymbol{z}$ for any $0 \leq \alpha \leq 1$ is a **convex combination**.

Example. Any subspace $\mathcal{S}$ in a vector space $\mathcal{V}$ is convex.

Example. The **nonnegative orthant** $\mathbb{R}_+^N \triangleq \{\boldsymbol{x} \in \mathbb{R}^N : \boldsymbol{x} \geq \boldsymbol{0}\}$ is convex.

Example. For any norm, the ball of radius $r > 0$ $\{\boldsymbol{x} : \|\boldsymbol{x}\| \leq r\}$ is convex. (HW)

Finding the nearest point in a convex set is an important operation. $\mathcal{P}_\mathcal{C}(\cdot)$ denotes the "**projection**" of its argument onto the closest point in $\mathcal{C}$:

$$\mathcal{P}_\mathcal{C}(\boldsymbol{z}) \triangleq \min_{\boldsymbol{x} \in \mathcal{C}} \|\boldsymbol{x} - \boldsymbol{z}\|$$

For $\mathcal{C} = \left\{\boldsymbol{x} \in \mathbb{R}^N : \|\boldsymbol{x}\|_\infty \leq 5\right\}$, which of these is the projection of a point $\boldsymbol{z} \in \mathbb{R}^N$ onto $\mathcal{C}$?

A: `min.(x,5)` B: `min.(abs.(x),5)` C: `min.(abs.(x),5).*sign(x)` D: None of these

$\boxed{??}$

Gradient projection method

In many applications, we seek the minimizer of a convex function $f(\boldsymbol{x})$ over a convex set $\mathcal{C}$:

$$\hat{\boldsymbol{x}} = \operatorname*{arg\,min}_{\boldsymbol{x} \in \mathcal{C}} f(\boldsymbol{x}).$$

This is called **constrained optimization**.

Fact. The conclusions about convergence for GD given above also hold for the more **gradient projection** (**GP**) method

$$\boldsymbol{x}_{k+1} = \mathcal{P}_{\mathcal{C}}(\boldsymbol{x}_k - \alpha \nabla f(\boldsymbol{x}_k)).$$

Example. (HW) NNLS, where $f(\boldsymbol{x}) = \frac{1}{2} \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{y}\|_2^2$ and $\mathcal{C} = \mathbb{R}_+^N$.

Example. If $f(\boldsymbol{x}) = \frac{1}{2} \left\| \begin{bmatrix} 4 & 0 \\ 0 & 2 \end{bmatrix} \boldsymbol{x} - \begin{bmatrix} 4 \\ -2 \end{bmatrix} \right\|_2^2$ and we apply GP with $\alpha = 1/L$ and $\boldsymbol{x}_0 = \mathbf{0}$ then $\boldsymbol{x}_1 = $?

A: $(0, 0)$ B: $(1, 0)$ C: $(1, -1)$ D: $(0, -1)$ E: $(1, 1)$ $\boxed{\textbf{??}}$

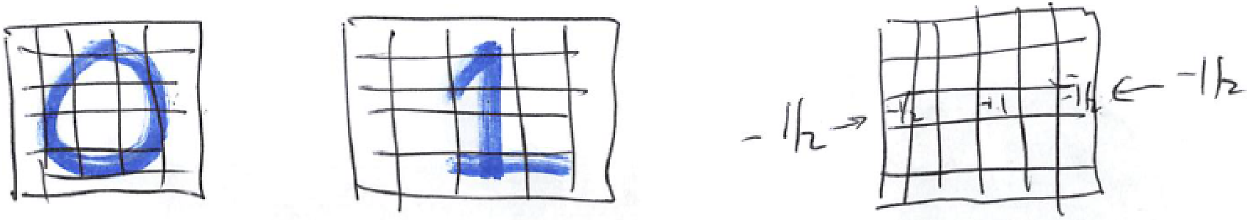
8.4 Machine learning via logistic regression for binary classification

In a **binary classification** problem we are given training feature vectors $\{\boldsymbol{v}_i\} \in \mathbb{R}^N$ and corresponding binary labels $\{y_i = \pm 1 : i = 1, \ldots, M\}$. Given these training pairs $(\boldsymbol{v}_i, y_i)$, the training goal is to learn weights $\boldsymbol{x} \in \mathbb{R}^N$ such that

- $\langle \boldsymbol{x}, \boldsymbol{v}_i \rangle > 0$ if $y_i = +1$ and
- $\langle \boldsymbol{x}, \boldsymbol{v}_i \rangle < 0$ if $y_i = -1$,

i.e., such that $\langle \boldsymbol{x}, y_i \boldsymbol{v}_i \rangle > 0$, so that a reasonable classifier is: $\operatorname{sign}(\langle \boldsymbol{x}, \boldsymbol{v} \rangle)$.

Example. Classifying gray-scale images of hand-written digits 0 & 1.



In this example, we "hand crafted" the weights $\boldsymbol{x}$, hoping that $\operatorname{sign}(\langle \boldsymbol{x}, \boldsymbol{v} \rangle)$ will be correct for a test image $\boldsymbol{v}$. For good statistical performance, we want to learn $\boldsymbol{x}$ from training data. Learning is especially important for harder problems like classifying 5 and 8.

To learn the weights $\boldsymbol{x}$, we can minimize a cost function with a regularization parameter $\beta \geq 0$:

$$\hat{\boldsymbol{x}} = \arg\min_{\boldsymbol{x}} f(\boldsymbol{x}), \qquad f(\boldsymbol{x}) = \sum_{i=1}^{M} h(y_i \langle \boldsymbol{x}, \boldsymbol{v}_i \rangle) + \beta \frac{1}{2} \|\boldsymbol{x}\|_2^2 = \mathbf{1}_M' h.(\boldsymbol{A}\boldsymbol{x}) + \beta \frac{1}{2} \|\boldsymbol{x}\|_2^2 ,$$

where the mth row of the $M \times N$ matrix $\boldsymbol{A}$ is $y_m \boldsymbol{v}_m^T$, *i.e.*, $\boldsymbol{A} \triangleq \begin{bmatrix} y_1 \boldsymbol{v}_1^T \\ \vdots \\ y_M \boldsymbol{v}_M^T \end{bmatrix}$.

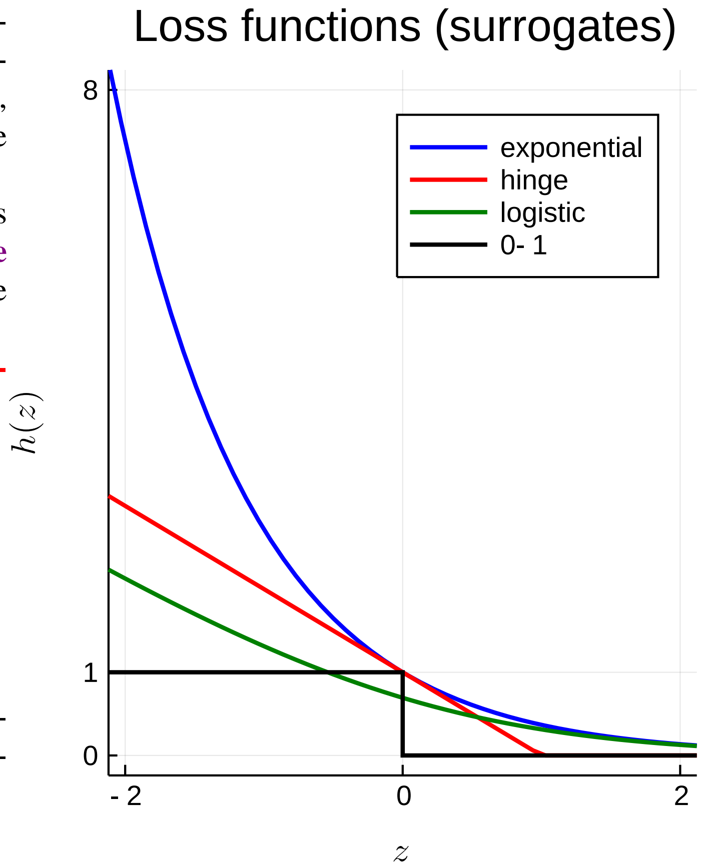
A regularization term like $\|\boldsymbol{x}\|_2^2$ is especially important in the typical case where the feature vector dimension N is large relative to the sample size M. The 1-norm is also often used; see EECS 598.

We want the function h to discourage incorrect classification of the training data.

- The 0-1 loss function $h(z) = \mathbb{I}_{\{z \leq 0\}}$ is natural because it counts how many training samples are misclassified, but it is nonconvex and nondifferentiable, so very difficult to use for optimization. Instead one usually uses **surrogate loss functions**.
- The **hinge loss** function $h(z) = \max(1 - z, 0)$ is related to the soft-margin **support-vector machine** (**SVM**) and is convex, but non-differentiable. (See EECS 598.)
- The **logistic** loss function is convex and has a **Lipschitz continuous** derivative:

$$\begin{aligned} h(z) &= \log\left(1 + \mathrm{e}^{-z}\right) \\ \dot{h}(z) &= -1/\left(\mathrm{e}^{z} + 1\right) \\ \ddot{h}(z) &= \frac{\mathrm{e}^{z}}{\left(\mathrm{e}^{z} + 1\right)^2} \in \left(0, \frac{1}{4}\right]. \end{aligned}$$

  It is suitable for gradient-based methods.
- The exponential loss function is convex and differentiable, but its derivative is not Lipschitz continuous. We will not consider it further.



For the logistic loss, the cost function is not quadratic, but it does have a Lipschitz continuous gradient. For gradient-based optimization, we need the cost function gradient:

$$\underbrace{\nabla f(\boldsymbol{x})}_{N \times 1} = \nabla \left(\mathbf{1}'_M h.(\boldsymbol{A}\boldsymbol{x}) + \beta \frac{1}{2} \|\boldsymbol{x}\|_2^2 \right) = \left(\sum_{m=1}^{M} \nabla h(\boldsymbol{A}_{m,:}\boldsymbol{x}) \right) + \beta \boldsymbol{x}$$

$$= \left(\sum_{m=1}^{M} \boldsymbol{A}'_{m,:} \dot{h}(\boldsymbol{A}_{m,:}\boldsymbol{x}) \right) + \beta \boldsymbol{x} = \boldsymbol{A}' \dot{h}.(\boldsymbol{A}\boldsymbol{x}) + \beta \boldsymbol{x}.$$

The cost function **Hessian matrix** is:

$$\nabla^2 f(\boldsymbol{x}) = \nabla^T \nabla f(\boldsymbol{x}) = \nabla^T \left(\sum_{m=1}^{M} \boldsymbol{A}_{m,:} \dot{h}(\boldsymbol{A}_{m,:}\boldsymbol{x}) + \beta \boldsymbol{x} \right) = \sum_{m=1}^{M} \boldsymbol{A}_{m,:} \ddot{h}(\boldsymbol{A}_{m,:}\boldsymbol{x}) \boldsymbol{A}'_{m,:} + \beta \boldsymbol{I}$$

$$= \boldsymbol{A}' \operatorname{diag}\left\{ \ddot{h}.(\boldsymbol{A}\boldsymbol{x}) \right\} \boldsymbol{A} + \beta \boldsymbol{I} = \underbrace{\boldsymbol{A}' \boldsymbol{D}(\boldsymbol{x}) \boldsymbol{A}}_{} + \beta \underbrace{\boldsymbol{I}}_{}, \qquad \boldsymbol{D}(\boldsymbol{x}) \triangleq \operatorname{diag}\left\{ \ddot{h}.(\boldsymbol{A}\boldsymbol{x}) \right\} .$$

f is a **strictly convex** function when ψ is the **logistic** loss function and $\beta > 0$. (?)

A: True B: False $\boxed{??}$

A Hessian majorizer leads to Lipschitz constant bound for $\nabla f(\boldsymbol{x})$ that is useful for many iterative algorithms: (HW)

$$\nabla^2 f(\boldsymbol{x}) = \boldsymbol{A}'\boldsymbol{D}\boldsymbol{A} + \beta \boldsymbol{I} \preceq \boldsymbol{A}' \left(\frac{1}{4}\boldsymbol{I}\right) \boldsymbol{A} + \beta \boldsymbol{I} = \frac{1}{4}\boldsymbol{A}'\boldsymbol{A} + \beta \boldsymbol{I} \preceq \frac{1}{4} \|\!| \boldsymbol{A}'\boldsymbol{A} |\!\|_2 \boldsymbol{I} + \beta \boldsymbol{I} = \left(\frac{1}{4} \|\!| \boldsymbol{A} |\!\|_2^2 + \beta\right) \boldsymbol{I}.$$

Thus, a Lipschitz constant bound that depends on the training data $\boldsymbol{A}$ and the regularization parameter β is:

$$L \triangleq \frac{1}{4} \|\!| \boldsymbol{A} |\!\|_2^2 + \beta.$$

Notice the process here: typically we do not try to find L numerically. Instead we analyze either ∇f or $\nabla^2 f$ and use properties and inequalities (analytically, on paper) to find a suitable L expressed in terms of the problem variables (in this case $\boldsymbol{A}$ and β).

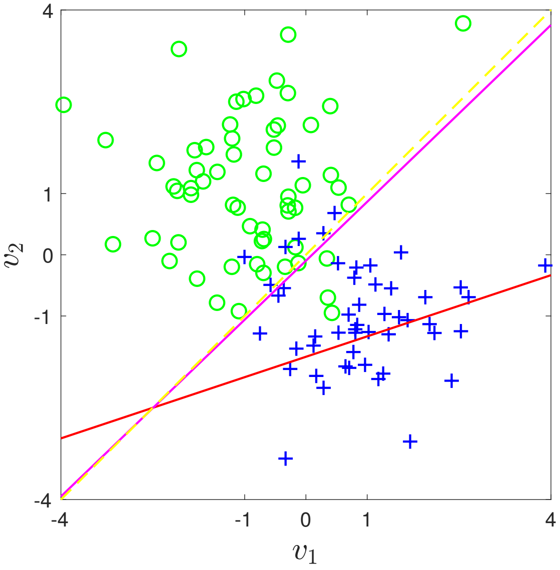
Practical implementation:

- Normalizing each row of $\boldsymbol{A}$ to unit norm can help keep e^z from overflowing.
- Tuning β should use **cross validation** or other such tools from machine learning.
- The cost function is convex with Lipschitz gradient, so it is well-suited for Nesterov's fast gradient method (or OGM).
- When feature dimension N is very large, seeking a sparse weight vector $\boldsymbol{x}$ may be preferable. For that, replace the Tikhonov regularizer $\|\boldsymbol{x}\|_2^2$ with $\|\boldsymbol{x}\|_1$ and then use FISTA (or POGM [8]) for optimization.

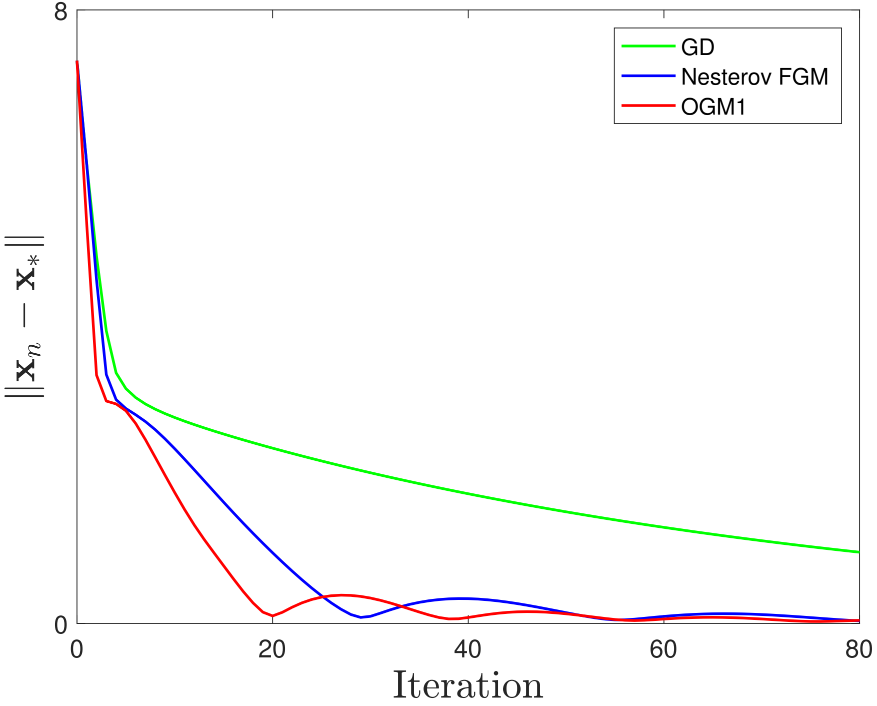
This **logistic regression** approach to classification will be explored in a task sheet.

Numerical Results: logistic regression

Labeled training data (green and blue points);
initial decision boundary (red);
final decision boundary (magenta);
ideal boundary (yellow).

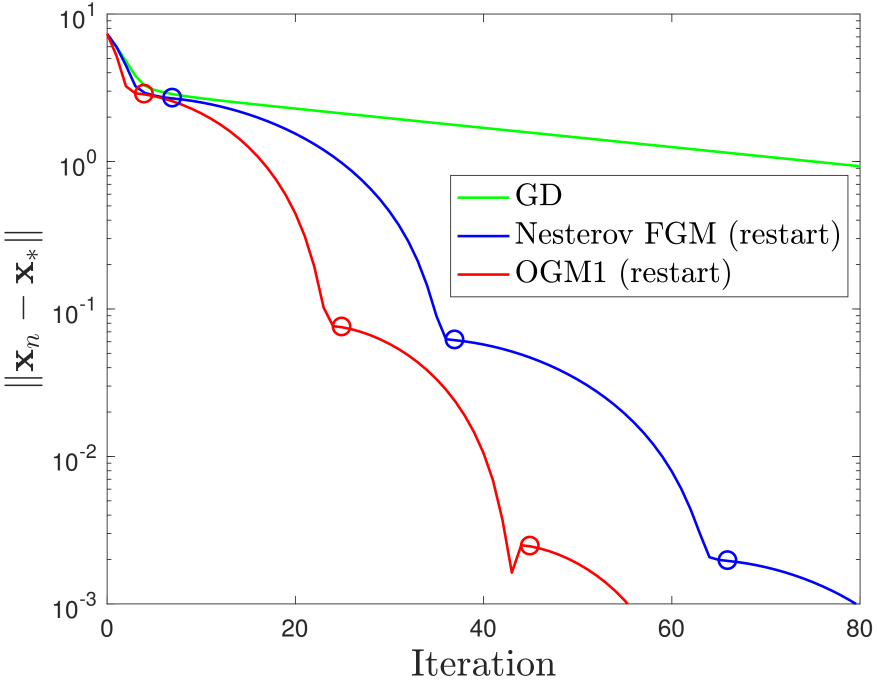


Numerical Results: convergence rates



OGM faster than FGM in early iterations...

Adaptive restart of accelerated GD



FGM restart, O'Donoghue & Candès, 2015. [9]
OGM restart [10]

8.5 Summary

This chapter barely scratches the surface of the field of optimization algorithms, but it illustrates how crucial matrix methods are to that field.

Bibliography

[1] W. Zuo and Z. Lin. “A generalized accelerated proximal gradient approach for total-variation-based image restoration”. In: *IEEE Trans. Im. Proc.* 20.10 (Oct. 2011), 2748–59 (cit. on p. 8.18).

[2] Y. Drori and M. Teboulle. “Performance of first-order methods for smooth convex minimization: A novel approach”. In: *Mathematical Programming* 145.1-2 (June 2014), 451–82 (cit. on pp. 8.19, 8.30).

[3] D. Kim and J. A. Fessler. “Optimized first-order methods for smooth convex minimization”. In: *Mathematical Programming* 159.1 (Sept. 2016), 81–107 (cit. on pp. 8.19, 8.25).

[4] R. C. Fair. “On the robust estimation of econometric models”. In: *Ann. Econ. Social Measurement* 2 (Oct. 1974), 667–77 (cit. on p. 8.29).

[5] K. Lange. “Convergence of EM image reconstruction algorithms with Gibbs smoothing”. In: *IEEE Trans. Med. Imag.* 9.4 (Dec. 1990). Corrections, T-MI, 10:2(288), June 1991., 439–46 (cit. on p. 8.29).

[6] B. T. Polyak. *Introduction to optimization*. New York: Optimization Software Inc, 1987 (cit. on p. 8.30).

[7] A. B. Taylor, J. M. Hendrickx, and Francois Glineur. “Smooth strongly convex interpolation and exact worst-case performance of first- order methods”. In: *Mathematical Programming* 161.1 (Jan. 2017), 307–45 (cit. on p. 8.30).

[8] A. B. Taylor, J. M. Hendrickx, and Francois Glineur. “Exact worst-case performance of first-order methods for composite convex optimization”. In: *SIAM J. Optim.* 27.3 (Jan. 2017), 1283–313 (cit. on p. 8.37).

[9] B. O’Donoghue and E. Candes. “Adaptive restart for accelerated gradient schemes”. In: *Found. Comp. Math.* 15.3 (June 2015), 715–32 (cit. on p. 8.40).

[10] D. Kim and J. A. Fessler. “Adaptive restart of the optimized gradient method for convex optimization”. In: *J. Optim. Theory Appl.* 178.1 (July 2018), 240–63 (cit. on p. 8.40).