Chapter 3

Gradient-based optimization

Contents (class version)

3.0 Introduction

To solve a problem like

$$\hat{\boldsymbol{x}} = \arg\min_{\boldsymbol{x} \in \mathbb{F}^N} \Psi(\boldsymbol{x})$$

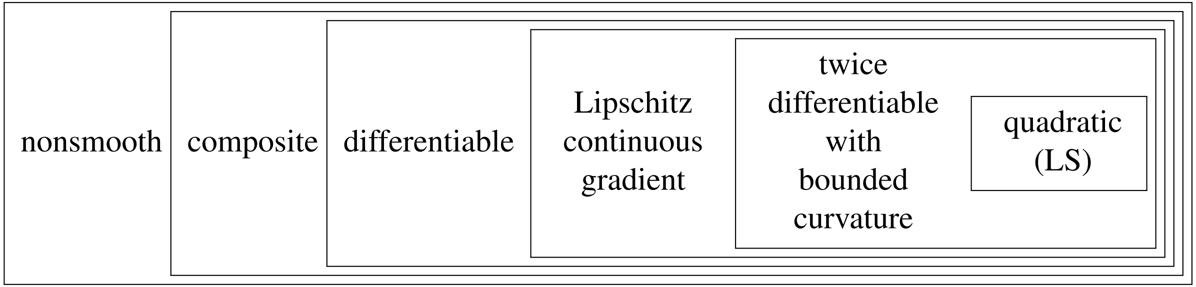
via an iterative method, we start with some initial guess $\boldsymbol{x}_0$, and then the algorithm produces a sequence $\{\boldsymbol{x}_t\}$ where hopefully the sequence **converges** to $\hat{\boldsymbol{x}}$, meaning $\|\boldsymbol{x}_t - \hat{\boldsymbol{x}}\| \to 0$ for some norm $\|\cdot\|$ as $t \to \infty$.

What algorithm we use depends greatly on the properties of the cost function $\Psi : \mathbb{F}^N \mapsto \mathbb{R}$.

EECS 551 explored the **gradient descent** (**GD**) and **preconditioned gradient descent** (**PGD**) algorithms for solving **least-squares problems** in detail.

Here we review the general form of **gradient descent** (**GD**) for **convex** minimization problems; the LS application is simply a special case.

Venn diagram for convex functions:



nonsmooth

composite

differentiable

Lipschitz continuous gradient

twice differentiable with bounded curvature

quadratic (LS)

Motivating application(s)

We focus initially on the numerous SIPML applications where the cost function is **convex** and **smooth**, meaning it has a **Lipschitz continuous** gradient.

A concrete family of applications is **edge-preserving image recovery** where the measurement model is

$$\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{\varepsilon}$$

for some matrix $\boldsymbol{A}$ and we estimate $\boldsymbol{x}$ using

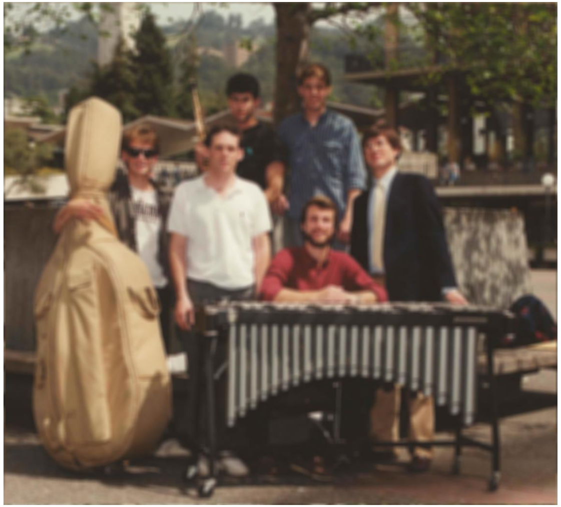
$$\hat{\boldsymbol{x}} = \operatorname*{arg\,min}_{\boldsymbol{x} \in \mathbb{F}^N} \frac{1}{2} \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{y}\|_2^2 + \beta R(\boldsymbol{x})$$

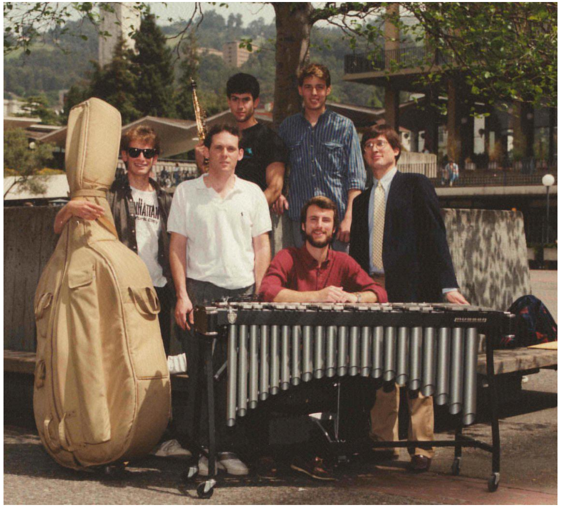
where the regularizer is convex and smooth, such as

$$R(\boldsymbol{x}) = \sum_k \psi([\boldsymbol{C}\boldsymbol{x}]_k)$$

for a potential function ψ that has a Lipschitz continuous derivative, such as the **Fair potential**.

Example. Here is an example of **image deblurring** or **image restoration** that was performed using such a method. The left image is the blurry noisy image $\boldsymbol{y}$, and the right image is the restored image $\hat{\boldsymbol{x}}$.





Step sizes and Lipschitz constant preview

For gradient-based optimization methods, a key issue is choosing an appropriate **step size** (aka **learning rate** in ML). Usually the appropriate range of step sizes is determined by the **Lipschitz constant** of $\nabla \Psi$, so we focus on that next.

3.1 Lipschitz continuity

The concept of **Lipschitz continuity** is defined for general metric spaces, but we focus on vector spaces.

Define. A function $g : \mathbb{F}^N \mapsto \mathbb{F}^M$ is **Lipschitz continuous** if there exists $L < \infty$, called a **Lipschitz constant**, such that

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

In general the norms on $\mathbb{F}^N$ and $\mathbb{F}^M$ can differ and L will depend on the choice of the norms. We will focus on the Euclidean norms unless otherwise specified.

Define. The smallest such L is called the **best Lipschitz constant**. (Often just “the” LC.)

Algebraic properties

Let f and g be Lipschitz continuous functions with (best) Lipschitz constants L_f and L_g respectively.

$h(\boldsymbol{x})$	L_h	
$\alpha f(\boldsymbol{x}) + \beta$	$\lvert\alpha\rvert L_f$	scale/shift
$f(\boldsymbol{x} - \boldsymbol{x}_0)$	L_f	translate
$f(\boldsymbol{x}) + g(\boldsymbol{x})$	$\leq L_f + L_g$	add
$f(g(\boldsymbol{x}))$	$\leq L_f L_g$	compose (HW)
$\boldsymbol{A}\boldsymbol{x} + \boldsymbol{b}$	$\\|\boldsymbol{A}\\|$	affine (for same norm on $\mathbb{F}^M$ and $\mathbb{F}^N$)
$f(\boldsymbol{x})g(\boldsymbol{x})$	?	multiply

If f and g are Lipschitz continuous functions on $\mathbb{R}$, then $h(x) = f(x)g(x)$ is a Lipschitz continuous function on $\mathbb{R}$. (?)

A: True B: False **??**

If $f : \mathbb{F}^N \mapsto \mathbb{F}$ and $g : \mathbb{F}^N \mapsto \mathbb{F}$ are **Lipschitz continuous** functions and $h(\boldsymbol{x}) \triangleq f(\boldsymbol{x})g(\boldsymbol{x})$,

and

then

Proof

$$|h(\boldsymbol{x}) - h(\boldsymbol{z})| =$$
$$=$$
$$\leq$$
$$\leq$$

Is boundedness of both f and g a necessary condition? group

For our purposes, we especially care about cost functions whose *gradients* are Lipschitz continuous. We call these **smooth** functions. The definition of gradient is subtle for functions on $\mathbb{C}^N$ so here we focus on $\mathbb{R}^N$.

Define. A differentiable function $f(\boldsymbol{x})$ is called **smooth** iff it has a **Lipschitz continuous gradient**, *i.e.*, iff $\exists L < \infty$ such that

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

Lipschitz continuity of ∇f is a stronger condition than mere continuity, so any differentiable function whose gradient is Lipschitz continuous is in fact a **continuously differentiable** function.

The set of differentiable functions on $\mathbb{R}^N$ having L-Lipschitz continuous gradients is sometimes denoted $\mathcal{C}_L^{1,1}(\mathbb{R}^N)$ [1, p. 20].

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 $L_{\nabla f} = \|\boldsymbol{A}'\boldsymbol{A}\|_2 = \|\boldsymbol{A}\|_2^2 = \sigma^2(\boldsymbol{A}) = \rho(\boldsymbol{A}'\boldsymbol{A})$.

The value $L_{\nabla f} = \|\boldsymbol{A}'\boldsymbol{A}\|_2$ is the *best* Lipschitz constant for $\nabla f(\cdot)$. (?)

A: True B: False **??**

??

Here is an interesting geometric property of functions in $\mathcal{C}_L^{1,1}(\mathbb{R}^N)$ [1, p. 22, Lemma 1.2.3]:

$$|f(\boldsymbol{x}) - f(\boldsymbol{z}) - \nabla f(\boldsymbol{z})(\boldsymbol{x} - \boldsymbol{z})| \leq \frac{L}{2} \|\boldsymbol{x} - \boldsymbol{z}\|_2^2, \ \forall \boldsymbol{x}, \boldsymbol{z} \in \mathbb{R}^N.$$

In other words, for any point $\boldsymbol{z}$, the function $f(\boldsymbol{x})$ is bounded between the two quadratic functions:

$$q_{\pm}(\boldsymbol{x}) \triangleq f(\boldsymbol{z}) + \langle \nabla f(\boldsymbol{z}), \boldsymbol{x} - \boldsymbol{z} \rangle \pm \frac{L}{2} \|\boldsymbol{x} - \boldsymbol{z}\|_2^2 .$$

Convex functions with Lipschitz continuous gradients

See [1, p. 56] for many equivalent conditions for a **convex** differentiable function f to have a Lipschitz continuous gradient, such as the following holding for all $\boldsymbol{x}, \boldsymbol{z} \in \mathbb{R}^N$:

$$\underbrace{f(\boldsymbol{z}) + \langle \nabla f(\boldsymbol{z}), \boldsymbol{x} - \boldsymbol{z} \rangle \leq}_{\text{tangent plane property}} f(\boldsymbol{x}) \underbrace{\leq f(\boldsymbol{z}) + \langle \nabla f(\boldsymbol{z}), \boldsymbol{x} - \boldsymbol{z} \rangle + \frac{L}{2} \|\boldsymbol{x} - \boldsymbol{z}\|_2^2}_{\text{quadratic majorization property}} .$$

The left inequality holds for all differentiable convex functions.

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

$$\left|\!\left|\!\left| \nabla^2 f(\boldsymbol{x}) \right|\!\right|\!\right|_2 \leq L, \quad \forall \boldsymbol{x} \in \mathbb{R}^N, \tag{3.1}$$

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**:

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

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

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

$\nabla^2 f = 2\boldsymbol{z}\boldsymbol{z}'$ where $\boldsymbol{z}' = [1\ 2]$ so $\left|\!\left|\!\left| \nabla^2 f \right|\!\right|\!\right|_2 = 2 \|\boldsymbol{z}\|_2^2 = 10$.

Boundedness of 2nd derivative is not a necessary condition in general, because Lipschitz continuity of the derivative of a function does not require the function to be twice differentiable.

Example. Consider $f(x) = \frac{1}{2}([x]_+)^2$. The derivative of this function is $\dot{f}(x) = [x]_+$ which has Lipschitz constant $L = 1$, yet f is not twice differentiable.

However, if a 1D function from $\mathbb{R}$ to $\mathbb{R}$ is twice differentiable, then its derivative is Lipschitz iff its second derivative is bounded.

Proof. The "if" follows from (3.1). For the "only if" direction, suppose $\ddot{f}$ is unbounded. Then for any $L < \infty$ there exists a point $x \in \mathbb{R}$ such that $\ddot{f}(x) > L$. Now consider $z = x \pm \epsilon$ and let $g(x) = \dot{f}(x)$. Then $\left|\frac{g(x)-g(z)}{x-z}\right| = \left|\frac{g(x)-g(x\pm\epsilon)}{\epsilon}\right| \to \ddot{f}(x) > L$ as $\epsilon \to 0$, so g cannot be L-Lipschitz continuous. This property holds for every $L < \infty$. □

Challenge. Generalize this partial converse of (3.1) to twice differentiable functions from $\mathbb{R}^N$ to $\mathbb{R}$, *i.e.*, prove or disprove this conjecture: if $f : \mathbb{R}^N \mapsto \mathbb{R}$ is twice differentiable, then ∇f is Lipshitz continuous iff the bounded Hessian norm property (3.1) holds.

(Read)

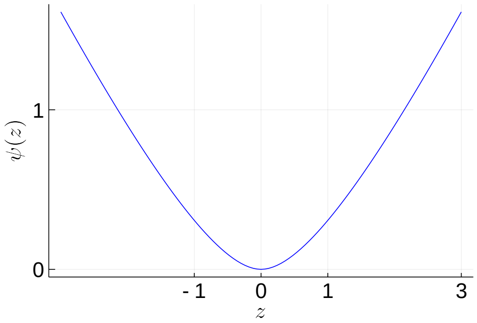
Example. The **Fair potential** used in many imaging applications [2] [3] is

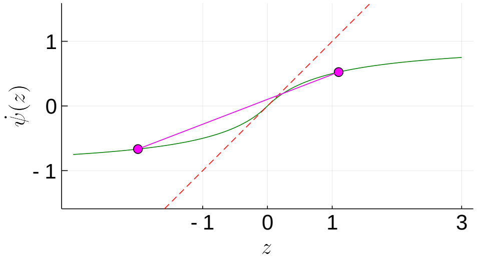
$$\psi(z) = \delta^2 \left(|z/\delta| - \log(1 + |z/\delta|) \right), \qquad (3.2)$$

for some $\delta > 0$ and has the property of being roughly quadratic for $z \approx 0$ and roughly like $|z|$ for $\delta\, |z| \gg 0$. When the domain of ψ is $\mathbb{R}$, we can differentiate (carefully treat $z > 0$ and $z < 0$ separately):

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

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





Example. Is the Fair potential ψ itself **Lipschitz continuous**? Yes: $\left|\dot{\psi}(z)\right| \leq 1$.

Edge-preserving regularizer and Lipschitz continuity

Example. Determine "the" **Lipschitz constant** for the gradient of the **edge-preserving regularizer** in $\mathbb{R}^N$, when the derivative $\dot{\psi}$ of potential function ψ has Lipschitz constant $L_{\dot{\psi}}$:

$$R(\boldsymbol{x}) = \sum_{k=1}^{K} \psi([\boldsymbol{C}\boldsymbol{x}]_k) \Longrightarrow \nabla R(\boldsymbol{x}) = \tag{3.3}$$

where

$$\|g(\boldsymbol{u}) - g(\boldsymbol{v})\|_2^2 =$$

$$\Longrightarrow L_g \leq \qquad \Longrightarrow L_{\nabla R} \leq \tag{3.4}$$

Thus when $\ddot{\psi} \leq 1$, a Lipschitz constant for the gradient of the above $R(\boldsymbol{x})$ is:

A: 1 B: $\|\boldsymbol{C}\|_2$ C: $\|\boldsymbol{C}'\boldsymbol{C}\|_2$ D: $\|\boldsymbol{C}\|_2^4$ E: None of these $\boxed{??}$

Showing this $L_{\nabla R}$ is the best Lipschitz constant is a HW problem.

3.2 Gradient descent for smooth convex functions

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

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

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

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

- the **step size** α is chosen such that $0 < \alpha < 2/L,$

then the GD iteration

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

has the following convergence properties [4, p. 207].

- The cost function is non-increasing (**monotone**): $\Psi(\boldsymbol{x}_{k+1}) \leq \Psi(\boldsymbol{x}_k), \ \forall k \geq 0.$
- The distance to any minimizer $\hat{\boldsymbol{x}}$ is non-increasing (**monotone**): $\|\boldsymbol{x}_{k+1} - \hat{\boldsymbol{x}}\|_2 \leq \|\boldsymbol{x}_k - \hat{\boldsymbol{x}}\|_2, \ \forall k \geq 0.$
- The sequence $\{\boldsymbol{x}_k\}$ **converges** to a minimizer of $\Psi(\cdot)$.
- The gradient norm converges to zero [4, p. 22] $\|\nabla \Psi(\boldsymbol{x}_k)\|_2 \to 0.$
- For $0 < \alpha \leq 1/L$, the cost function decrease is bounded by [5]:

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

  This upper bound is conjectured to also hold for $1/L < \alpha < 2/L$ [6].

Optimal asymptotic step size for GD (Read)

The above step size range $0 < \alpha < 2/L$ is a wide range of values, and one might ask what is the best choice?

For a LS cost function $f(\boldsymbol{x}) = \frac{1}{2} \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{y}\|_2^2$, the EECS 551 notes show that the asymptotically optimal choice of the step size is:

$$\alpha_* = \frac{2}{\sigma_{\max}(\boldsymbol{A}'\boldsymbol{A}) + \sigma_{\min}(\boldsymbol{A}'\boldsymbol{A})} = \frac{2}{\sigma_{\max}(\nabla^2 f) + \sigma_{\min}(\nabla^2 f)},$$

because $\nabla^2 f = \boldsymbol{A}'\boldsymbol{A}$.

For more general cost functions that are twice differentiable, one can apply similar analyses to show that the asymptotically optimal choice is

$$\alpha_* = \frac{2}{\sigma_{\max}(\nabla^2 f(\hat{\boldsymbol{x}})) + \sigma_{\min}(\nabla^2 f(\hat{\boldsymbol{x}}))}.$$

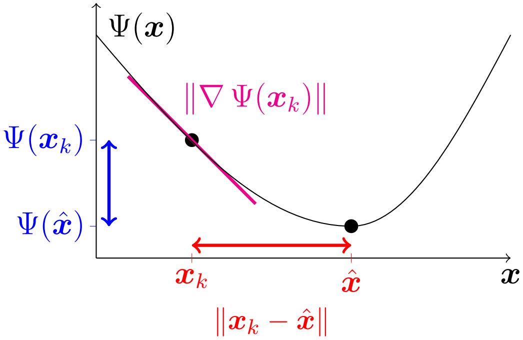
Although this formula is an interesting generalization, it is of little practical use because we do not know the minimizer $\hat{\boldsymbol{x}}$ and the Hessian $\nabla^2 f$ and its SVD are infeasible for large problems.
Furthermore, the asymptotically optimal choice of α_* may not be the best step size in the early iterations when the iterates are far from $\hat{\boldsymbol{x}}$.

Convergence rates (Read)

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

- $\Psi(\boldsymbol{x}_k) \to \Psi(\hat{\boldsymbol{x}})$
- $\|\nabla \Psi(\boldsymbol{x}_k)\| \to 0$
- $\|\boldsymbol{x}_k - \hat{\boldsymbol{x}}\| \to 0$

  both globally and locally...


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

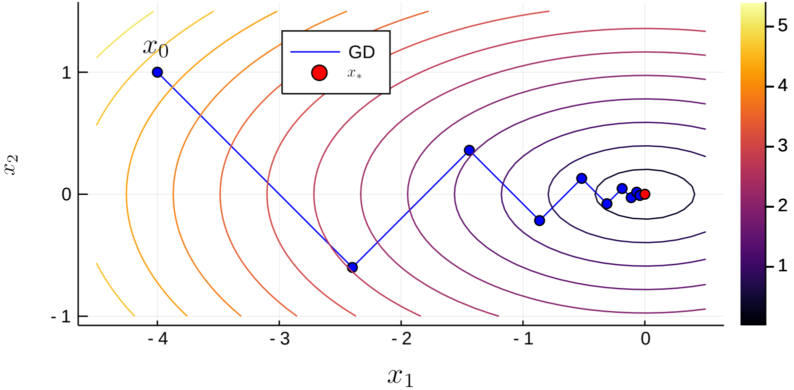
Quantifying bounds on the rates of decrease of these quantities is an active research area. Even classical GD has relatively recent results [5] that tighten up the traditional bounds. The tightest possible worst-case bound for GD for the decrease of the cost function (with a fixed step size $\alpha = 1/L$) is $O(1/k)$:

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

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

In contrast, Nesterov's fast gradient method (p. 3.40) 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 [7].

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 $\|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{y}\|$.

Two ways to try to accelerate convergence are to use a **preconditioner** and/or a **line search**.

3.3 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) [8]. Here is how **preconditioned steepest descent** for a linear LS problem works:

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

- Finding α_k analytically quadratic cases is a HW problem
- By construction, this iteration is guaranteed to decrease the cost function monotonically, with strict decrease unless $\boldsymbol{x}_k$ is already a minimizer, provided the preconditioner $\boldsymbol{P}$ is **positive definite**. Expressed mathematically:

$$\nabla\,\Psi(\boldsymbol{x}_k) \neq \boldsymbol{0} \Longrightarrow \Psi(\boldsymbol{x}_{k+1}) < \Psi(\boldsymbol{x}_k)\,.$$

- Computing α_k takes some extra work, especially for non-quadratic problems. Often Nesterov's fast gradient method or the **optimized gradient method** (**OGM**) [7] are preferable because they do not require a line search (if the Lipschitz constant is available).

Preconditioning: overview

(Read)

Why use the preconditioned search direction $\boldsymbol{d}_k = -\boldsymbol{P}\nabla\,\Psi(\boldsymbol{x}_k)$?

Consider the least-squares cost function $\Psi(\boldsymbol{x}) = \frac{1}{2}\left\|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{y}\right\|_2^2$, and define a "preconditioned" cost function using a change of coordinates:

$$f(\boldsymbol{z}) \triangleq \Psi(\boldsymbol{T}\boldsymbol{z}) = \frac{1}{2}\left\|\boldsymbol{A}\boldsymbol{T}\boldsymbol{z} - \boldsymbol{y}\right\|_2^2 .$$

The **Hessian matrix** of $f(\cdot)$ is $\nabla^2 f(\boldsymbol{z}) = \boldsymbol{T}'\boldsymbol{A}'\boldsymbol{A}\boldsymbol{T}$.
Applying GD to f yields

$$\begin{aligned} \boldsymbol{z}_{k+1} &= \boldsymbol{z}_k - \alpha\nabla f(\boldsymbol{z}_k) = \boldsymbol{T}'\boldsymbol{A}'(\boldsymbol{A}'\boldsymbol{T}\boldsymbol{z}_k - \boldsymbol{y}) \\ \Longrightarrow \boldsymbol{T}\boldsymbol{z}_{k+1} &= \boldsymbol{T}\boldsymbol{z}_k - \alpha\boldsymbol{T}\boldsymbol{T}'\boldsymbol{A}'(\boldsymbol{A}'\boldsymbol{T}\boldsymbol{z}_k - \boldsymbol{y}) \\ \Longrightarrow \boldsymbol{x}_{k+1} &= \boldsymbol{x}_k - \alpha\boldsymbol{P}\boldsymbol{A}'(\boldsymbol{A}'\boldsymbol{x}_k - \boldsymbol{y}) = \boldsymbol{x}_k - \alpha\boldsymbol{P}\nabla\,\Psi(\boldsymbol{x}_k), \end{aligned}$$

where $\boldsymbol{x}_k \triangleq \boldsymbol{T} z_k$ and $\boldsymbol{P} \triangleq \boldsymbol{T}\boldsymbol{T}'$. So ordinary GD on f is the same as preconditioned GD on Ψ.

If $\alpha = 1$ and $\boldsymbol{T} = (\boldsymbol{A}'\boldsymbol{A})^{-1/2}$, then $\boldsymbol{P} = (\boldsymbol{A}'\boldsymbol{A})^{-1}$ and $\boldsymbol{x}_1 = (\boldsymbol{A}'\boldsymbol{A})^{-1}\boldsymbol{A}'\boldsymbol{y}$.
In this sense $\boldsymbol{P} = (\boldsymbol{A}'\boldsymbol{A})^{-1} = [\nabla^2\,\Psi(\boldsymbol{x}_k)]^{-1}$ is the **ideal preconditioner**.

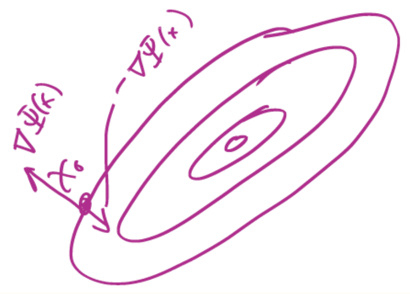
To elaborate, when $\boldsymbol{T} = (\boldsymbol{A}'\boldsymbol{A})^{-1/2}$, then $f(\boldsymbol{z})$ simplifies as follows:

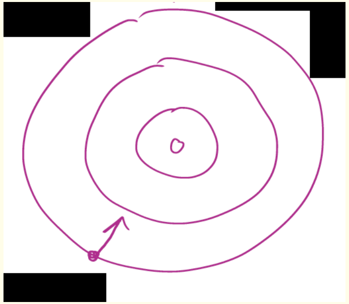
$$\begin{aligned} f(\boldsymbol{z}) &= \frac{1}{2}(\boldsymbol{A}\boldsymbol{T}\boldsymbol{z} - \boldsymbol{y})'(\boldsymbol{A}\boldsymbol{T}\boldsymbol{z} - \boldsymbol{y}) = \frac{1}{2}\left(\boldsymbol{z}'\boldsymbol{T}'\boldsymbol{A}'\boldsymbol{A}\boldsymbol{T}\boldsymbol{z} - 2\,\mathrm{real}\{\boldsymbol{y}'\boldsymbol{A}\boldsymbol{T}\boldsymbol{z}\} + \|\boldsymbol{y}\|_2^2\right) \\ &= \frac{1}{2}\left(\boldsymbol{z}'\boldsymbol{I}\boldsymbol{z} - 2\,\mathrm{real}\{\boldsymbol{y}'\boldsymbol{A}\boldsymbol{T}\boldsymbol{z}\} + \|\boldsymbol{y}\|_2^2\right) = \frac{1}{2}\left(\|\boldsymbol{z} - \boldsymbol{T}'\boldsymbol{A}'\boldsymbol{y}\|_2^2 - \|\boldsymbol{T}'\boldsymbol{A}'\boldsymbol{y}\|_2^2 + \|\boldsymbol{y}\|_2^2\right). \end{aligned}$$

The next figures illustrate this property of converging in 1 iteration for a quadratic cost with the ideal preconditioner.

Example. Effect of ideal preconditioner on quadratic cost function contours.

Contours of $\Psi(\boldsymbol{x})$ contours of $f(\boldsymbol{z})$




[redacted]
[redacted]
[redacted]

Descent direction

Define. A vector $\boldsymbol{d} \in \mathbb{F}^N$ **descent direction** for a cost function $\Psi : \mathbb{F}^N \mapsto \mathbb{R}$ at a point $\boldsymbol{x}$ iff moving *locally* from $\boldsymbol{x}$ along the direction $\boldsymbol{d}$ decreases the cost, *i.e.*,

$$\exists\, c = c(\boldsymbol{x}, \boldsymbol{d}, \Psi) > 0 \text{ s.t. } \forall \epsilon \in [0, c) \;\; \Psi(\boldsymbol{x} + \epsilon \boldsymbol{d}) \leq \Psi(\boldsymbol{x})\,. \tag{3.5}$$

With this definition $\boldsymbol{d} = \boldsymbol{0}$ is always a (degenerate) descent direction.

Fact. For $\mathbb{R}^N$, if $\Psi(\boldsymbol{x})$ is differentiable at $\boldsymbol{x}$ and $\boldsymbol{P}$ is **positive definite**, then the following vector, if nonzero, is a **descent direction** for Ψ at $\boldsymbol{x}$:

$$\boldsymbol{d} = -\boldsymbol{P} \nabla \Psi(\boldsymbol{x})\,. \tag{3.6}$$

Proof sketch. **Taylor's theorem** yields (Read)

$$\Psi(\boldsymbol{x}) - \Psi(\boldsymbol{x} + \alpha \boldsymbol{d}) = -\alpha \langle \nabla \Psi(\boldsymbol{x}),\, \boldsymbol{d} \rangle + o(\alpha) = \alpha \left[\boldsymbol{d}' \boldsymbol{P}^{-1} \boldsymbol{d} + \frac{o(\alpha)}{\alpha} \right],$$

which will be positive for sufficiently small α, because $\boldsymbol{d}' \boldsymbol{P}^{-1} \boldsymbol{d} > 0$ for $\boldsymbol{P} \succ \boldsymbol{0}$, and $o(\alpha)/\alpha \to 0$ as $\alpha \to 0$.

From this analysis we can see that designing/selecting a **preconditioner** that is **positive definite** is crucial. The two most common choices are:

- $\boldsymbol{P}$ is **diagonal** with positive diagonal elements,
- $\boldsymbol{P} =$ where $\boldsymbol{D}$ is diagonal with positive diagonal elements and $\boldsymbol{Q}$ is unitary.
  In this case $\boldsymbol{Q}$ is often circulant so we can use FFT operations to perform $\boldsymbol{P}\boldsymbol{g}$ efficiently.

Complex case

(Read)

The definition of **descent direction** in (3.5) is perfectly appropriate for both $\mathbb{R}^N$ and $\mathbb{C}^N$. However, the direction $\boldsymbol{d}$ specified in (3.6) is problematic in general on $\mathbb{F}^N$ because many cost functions of interest are not **holomorphic** so are not differentiable on $\mathbb{F}^N$.

However, despite not being differentiable, we can still find a **descent direction** for most cases of interest.

Example. The most important case of interest here is $\Psi : \mathbb{C}^N \mapsto \mathbb{R}$ defined by $\Psi(\boldsymbol{x}) = \frac{1}{2} \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{y}\|_2^2$ where $\boldsymbol{A} \in \mathbb{C}^{M \times N}$ and $\boldsymbol{y} \in \mathbb{C}^M$. This function is not **holomorphic**. However, one can show that

$$\boldsymbol{d} = -\boldsymbol{P}\boldsymbol{A}'(\boldsymbol{A}\boldsymbol{x} - \boldsymbol{y})$$

is a **descent direction** for Ψ at $\boldsymbol{x}$ when $\boldsymbol{P}$ is a positive definite matrix. (HW)

In the context of optimization problems, when we write $\boldsymbol{g} = \nabla \Psi(\boldsymbol{x}) = \boldsymbol{A}'(\boldsymbol{A}\boldsymbol{x} - \boldsymbol{y})$ for the complex case, we mean that $-\boldsymbol{g}$ is a descent direction for Ψ at $\boldsymbol{x}$, not a derivative.

Furthermore, one can show (HW) that the set of minimizers of $\Psi(\boldsymbol{x})$ is the same as the set of points that satisfy $\nabla \Psi(\boldsymbol{x}) = \boldsymbol{A}'(\boldsymbol{A}\boldsymbol{x} - \boldsymbol{y}) = \boldsymbol{0}$. So again even though a derivative is not defined here, the descent direction sure walks like a duck and talks like a **duck**, I mean like a (negated) derivative.

3.4 Descent direction for edge-preserving regularizer: complex case

Now consider an **edge-preserving regularizer** defined on $\mathbb{C}^N$:

$$R(\boldsymbol{x}) = \sum_{k=1}^{K} \psi([\boldsymbol{C}\boldsymbol{x}]_k), \text{ where } \psi(z) = f(|z|) \tag{3.7}$$

for some some **potential function** $\psi : \mathbb{C} \mapsto \mathbb{R}$ defined in terms of some function $f : \mathbb{R} \mapsto \mathbb{R}$.

If $f(r) = r^2$ then it follows from p. 3.22 that $\boldsymbol{C}'\boldsymbol{C}\boldsymbol{x}$ is a **descent direction** for $R(\boldsymbol{x})$ on $\mathbb{C}^N$. But it seems unclear in general how to define a descent direction, due to the $|\cdot|$ above.

To proceed, make the following assumptions about the function f:

- $f : \mathbb{R} \mapsto \mathbb{R}$.
- $0 \leq s \leq t \implies f(s) \leq f(t)$ (monotone).
- f is differentiable on $\mathbb{R}$.
- $\omega_f(t) \triangleq \dfrac{\dot{f}(t)}{t}$ is well-defined for all $t \in \mathbb{R}$, including $t = 0$.
- $0 \leq \omega_f(t) \leq \omega_{\max} < \infty$.

Example. For the **Fair potential** $\omega_f(t) = 1/\left(1 + |t/\delta|\right) \in (0, 1]$.

Claim. For these assumptions, a **descent direction** for the **edge-preserving regularizer** (3.7) for $\boldsymbol{x} \in \mathbb{F}^N$ is

$$-\nabla R(\boldsymbol{x}) = -\boldsymbol{C}' \operatorname{diag}\{\omega_f .(|\boldsymbol{C}\boldsymbol{x}|)\} \, \boldsymbol{C}\boldsymbol{x}, \tag{3.8}$$

where the $|\cdot|$ is evaluated element-wise, like `abs.()` in JULIA. Intuition: $\dot{f}(t) = \omega_f(t)\, t$.

To prove this claim, we focus on just one term in the sum: (Read)

$$r(\boldsymbol{x}) \triangleq \psi(\boldsymbol{v}'\boldsymbol{x}) = f(|\boldsymbol{v}'\boldsymbol{x}|), \qquad \boldsymbol{d}(\boldsymbol{x}) = -\boldsymbol{v}\, \omega_f(|\boldsymbol{v}'\boldsymbol{x}|)\, \boldsymbol{v}'\boldsymbol{x}.$$

for some nonzero vector $\boldsymbol{v} \in \mathbb{C}^N$. (A row of $\boldsymbol{C}$ in (3.7) corresponds to $\boldsymbol{v}'$.)

Letting $\omega = \omega_f(|\boldsymbol{v}'\boldsymbol{x}|) \leq \omega_{\text{max}}$ we have

$$r(\boldsymbol{x} + \epsilon\boldsymbol{d}) = f(|\boldsymbol{v}'(\boldsymbol{x} + \epsilon\boldsymbol{d})|) = f(|\boldsymbol{v}'(\boldsymbol{x} - \epsilon\omega\boldsymbol{v}\boldsymbol{v}'\boldsymbol{x})|) = f(|\boldsymbol{v}'\boldsymbol{x}(1 - \epsilon\omega\boldsymbol{v}'\boldsymbol{v})|) = f\left(|\boldsymbol{v}'\boldsymbol{x}| \left|1 - \epsilon\omega \|\boldsymbol{v}\|_2^2\right|\right).$$

Now when $0 \leq \epsilon \leq 1/(\omega \|\boldsymbol{v}\|_2^2)$ then

$$\left|1 - \epsilon\omega \|\boldsymbol{v}\|_2^2\right| = 1 - \epsilon\omega \|\boldsymbol{v}\|_2^2 \in [0, 1],$$

$$\Longrightarrow r(\boldsymbol{x} + \epsilon\boldsymbol{d}) = f\left(|\boldsymbol{v}'\boldsymbol{x}| \left(1 - \epsilon\omega \|\boldsymbol{v}\|_2^2\right)\right) \leq f(|\boldsymbol{v}'\boldsymbol{x}|) = r(\boldsymbol{x}),$$

using the monotone property of $f(\cdot)$. So $\boldsymbol{d}(\boldsymbol{x})$ is a descent direction for $r(\boldsymbol{x})$.
The proof of (3.8) involves the sum of K such terms. □

So with our usual reuse of ∇ to denote a (negated) **descent direction**, we will not write (3.3) for $\mathbb{C}^N$ but rather we will define $\omega_\psi = \omega_f$ and write:

$$\nabla R(\boldsymbol{x}) = \boldsymbol{C}' \, \mathsf{diag}\{\omega_\psi .(|\boldsymbol{C}\boldsymbol{x}|)\} \, \boldsymbol{C}\boldsymbol{x}. \tag{3.9}$$

If $\psi(z) = \psi(|z|), \ \forall z \in \mathbb{C}$, then we can define ω_ψ in terms of $\dot{\psi}$ for nonnegative real arguments.

The result (3.9) is shown in [9] using **Wirtinger calculus**.

Lipschitz constant for descent direction (Read)

Having established the descent direction (3.8) for edge-preserving regularization on $\mathbb{C}^N$, the next step is to determine a **Lipschitz constant** for that function.

Again we can write it as a composition of three functions:

$$\boldsymbol{C}' \operatorname{diag}\{\omega_\psi .(|\boldsymbol{C}\boldsymbol{x}|)\} \boldsymbol{C}\boldsymbol{x} = h(g(f(\boldsymbol{x}))), \quad f(\boldsymbol{x}) = \boldsymbol{C}\boldsymbol{x}, \; g(\boldsymbol{u}) = d.(\boldsymbol{u}), \; h(\boldsymbol{v}) = \boldsymbol{C}'\boldsymbol{v}, \quad d(z) \triangleq \omega_\psi(|z|)\, z.$$

For real z arguments and when $\dot{\psi}$ and hence ω_ψ are symmetric, $d(z) = \dot{\psi}(z)$ so the Lipschitz constant is easy.

When $z \in \mathbb{C}$, I have not yet been able to prove Lipschitz continuity of $d(z)$. My **conjecture**, supported by numerical experiments and [9, App. A], is that if $\omega_\psi(t)$ is a non-increasing function of t on $[0, \infty)$, in addition to the other assumptions on p. 3.23, then

$$L_d = \omega_\psi(0) \tag{3.10}$$

and, akin to (3.4), the Lipschitz constant for the descent direction on $\mathbb{C}^N$ is:

$$L_{\nabla R} = \|\boldsymbol{C}\|_2^2 L_d. \tag{3.11}$$

Challenge. Prove (3.10). Here are some initial steps that might help:

$$\begin{aligned} |d(x) - d(z)| &= |\omega_\psi(|x|)\, x - \omega_\psi(|z|)\, z| = \left|\omega_\psi(|x|)\, |x|\, \mathrm{e}^{\imath \angle x} - \omega_\psi(|z|)\, |z|\, \mathrm{e}^{\imath \angle z}\right| \\ &= \left|\dot{\psi}(|x|)\, \mathrm{e}^{\imath \angle x} - \dot{\psi}(|z|)\, \mathrm{e}^{\imath \angle z}\right| \leq \quad ? \qquad\qquad \leq \omega_\psi(0)\, |x - z|\,. \end{aligned}$$

Practical Lipschitz constant (Read)

In general, computing $\|C\|_2^2$ in (3.4) exactly would require a SVD or the power iteration, both of which are impractical for large-scale problems.

If we use finite differences with periodic boundary conditions, then C is circulant and hence is a **normal matrix** so $\|C\|_2 = \rho(C)$, where $\rho(\cdot)$ denotes the **spectra radius**. For 1D finite differences with N even, the spectral radius is 2 and for N odd is $1 + \cos(\pi (N-1)/N) \approx 2$. (HW)

But for nonperiodic boundary conditions we need a different approach.

Recall that because $C'C$ is symmetric:

$$\|C\|_2^2 = \|C'C\|_2 = \sigma_1(C'C) = \rho(C'C) \leq \|C'C\|,$$

for any matrix norm $\|\cdot\|$. In particular, the matrix 1-norm is convenient:

$$\|C'C\|_1 \leq \|C'\|_1 \|C\|_1 = \|C\|_\infty \|C\|_1$$
$$\Longrightarrow \|C\|_2^2 \leq \|C\|_\infty \|C\|_1 = 2 \cdot 2 = 4$$

for 1D finite differences, because there as most a single $+1$ and -1 in each row or column of C. Interestingly, this 1-norm approach gives us an upper bound on $\|C\|_2^2$ for any boundary conditions that matches the exact value when using periodic boundary conditions.

So the practical choice for 1D first-order finite-differences is to use $L_{\nabla R} = 4 L_{\dot\psi}$, where often we scale the potential functions so that $L_{\dot\psi} = 1$. Bottom line: never use `opnorm()` when working with finite differences!

GD step size with preconditioning

We earlier argued that the preconditioned gradient $-\boldsymbol{P}\nabla\,\Psi(\boldsymbol{x})$ can be preferable to $-\nabla\,\Psi(\boldsymbol{x})$ as a descent direction. But the GD convergence theorem on p. 3.14 had no $\boldsymbol{P}$ in it. So must we resort to PSD on p. 3.18 that requires a line search? No!

Suppose Ψ is convex has a Lipschitz continuous gradient with Lipschitz constant $L_{\nabla\,\Psi}$.
Define a new function in transformed coordinate system: $f(\boldsymbol{z}) \triangleq \Psi(\boldsymbol{T}\boldsymbol{z})$. Using the properties on p. 3.6, this function also has a Lipschitz continuous gradient and

$$\nabla f(\boldsymbol{z}) =$$

Choose a step size $0 < \alpha_f < 2/L_{\nabla f}$ for applying GD to f, yielding

$$\begin{aligned} \boldsymbol{z}_{k+1} &= \boldsymbol{z}_k - \alpha_f \nabla f(\boldsymbol{z}_k) = \boldsymbol{z}_k - \alpha_f \boldsymbol{T}'\nabla\,\Psi(\boldsymbol{T}\boldsymbol{z}_k) \\ \Longrightarrow \boldsymbol{T}\boldsymbol{z}_{k+1} &= \boldsymbol{T}\boldsymbol{z}_k - \alpha_f \boldsymbol{T}\boldsymbol{T}'\nabla\,\Psi(\boldsymbol{T}\boldsymbol{z}_k) \\ \Longrightarrow \boldsymbol{x}_{k+1} &= \boldsymbol{x}_k - \alpha_f \boldsymbol{P}\nabla\,\Psi(\boldsymbol{x}_k), \end{aligned}$$

where $\boldsymbol{x}_k \triangleq \boldsymbol{T} z_k$ and $\boldsymbol{P} = \boldsymbol{T}\boldsymbol{T}'$. So ordinary GD on f is the same as preconditioned GD on Ψ. The step size should satisfy $0 < \alpha_f < 2/L_{\nabla f}$ so it suffices (but can be suboptimal; see HW) to choose

$$0 < \alpha_f <$$

Finite difference implementation

For $\boldsymbol{x} \in \mathbb{F}^N$ we need to compute first-order **finite differences** $d_n = x_{n+1} - x_n,\ n = 1, \ldots, N-1$, which in matrix notation is $\boldsymbol{d} = \boldsymbol{C}\boldsymbol{x} \in \mathbb{F}^{N-1}$. Here are eight (!) different implementations in JULIA:

```julia
function loopdiff(x::AbstractVector)
  N = length(x)
  y = similar(x, N-1)
  for n=1:(N-1)
    @inbounds y[n] = x[n+1] - x[n]
  end
  return y
end

d = diff(x) # built-in
d = loopdiff(x)
d = (circshift(x, -1) - x)[1:(N-1)]
d = [x[n+1]-x[n] for n=1:(length(x)-1)] # comprehension
d = @views x[2:end] - x[1:(end-1)] # indexing
d = conv(x, eltype(x).([1, -1]))[2:end-1] # using DSP
d = diagm(0 => -ones(N-1), 1 => ones(N-1))[1:(N-1),:] * x # big/slow
d = spdiagm(0 => -ones(N-1), 1 => ones(N-1))[1:(N-1),:] * x # SparseArrays
```

Which is fastest? `https://web.eecs.umich.edu/~fessler/course/598/demo/diff1.html`

Use notebook to discuss

`@inbounds`

`@views`

`spdiagm` (column-wise storage...)

`LinearMaps` (cf `fatrix` in MIRT)

Adjoint tests

```
x = randn(N); y = randn(M);
@assert isapprox(y'*(A*x), (A'*y)'*x)
```

A generalization of transpose for linear maps is called the **adjoint**.

Orthogonality for steepest descent and conjugate gradients

Recall that the (preconditioned) steepest descent method has three steps:

- Descent direction $\boldsymbol{d}_k$, *e.g.*, $\boldsymbol{d}_k = -\boldsymbol{P}\nabla\,\Psi(\boldsymbol{x}_k)$
- line search: $\alpha_k = \arg\min_\alpha f_k(\alpha), \; f_k(\alpha) = \Psi(\boldsymbol{x}_k + \alpha \boldsymbol{d}_k)$
- update $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_k \boldsymbol{d}_k$

By construction:

$$0 = \dot{f}_k(\alpha_k) = \langle \boldsymbol{d}_k, \, \nabla\,\Psi(\boldsymbol{x}_k + \alpha_k \boldsymbol{d}_k) \rangle = \langle \boldsymbol{d}_k, \, \nabla\,\Psi(\boldsymbol{x}_{k+1}) \rangle \, .$$

In other words, the gradient $\nabla\,\Psi(\boldsymbol{x}_{k+1})$ at the next iterate is perpendicular to the current search direction $\boldsymbol{d}_k$:

This orthogonality leads to the "zig-zag" nature of PSD iterates seen on p. 3.17.

The **preconditioned conjugate gradient** (**CG**) method, described in more detail in [10], replaces the standard inner product with a different inner product weighted by the Hessian of the cost function:

$$\langle \boldsymbol{d}_k, \, \nabla\,\Psi(\boldsymbol{x}_{k+1}) \rangle_{\boldsymbol{H}} = \boldsymbol{d}_k' \boldsymbol{H} \nabla\,\Psi(\boldsymbol{x}_{k+1}), \qquad \boldsymbol{H} = \nabla^2\,\Psi(\boldsymbol{x}_k),$$

leading to faster convergence. See [10] for details.

Note that the CG method we want for optimization is the **nonlinear conjugate gradient** (**NCG**) method; in these notes CG means NCG.

3.5 General inverse problems

As mentioned previously, a typical cost function for solving inverse problems has the form

$$\hat{\boldsymbol{x}} = \operatorname*{arg\,min}_{\boldsymbol{x} \in \mathbb{F}^N} \frac{1}{2} \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{y}\|_2^2 + \beta R(\boldsymbol{x}), \quad R(\boldsymbol{x}) = \sum_{k=1}^{K} \psi([\boldsymbol{C}\boldsymbol{x}]_k) = r(\boldsymbol{C}\boldsymbol{x}), \quad r(\boldsymbol{v}) = \sum_{k=1}^{K} \psi(v_k) \,. \tag{3.12}$$

This is just one of many possible special cases of the following fairly general form

(3.13)

where $\boldsymbol{B}_j$ is a $M_j \times N$ matrix and each $f_j : \mathbb{R}^{M_j} \mapsto \mathbb{R}$ is a (typically convex) function.

Example. For the special case (3.12), we use (3.13) with

$$J = 2, \quad \boldsymbol{B}_1 = \boldsymbol{A}, \quad \boldsymbol{B}_2 = \boldsymbol{C}, \quad f_1(\boldsymbol{u}) = \frac{1}{2} \|\boldsymbol{u} - \boldsymbol{y}\|_2^2 \,, \quad f_2(\boldsymbol{v}) = \beta r(\boldsymbol{v}).$$

We will implement several algorithms for minimize cost functions of this general form.

When ψ is the Fair potential, the (best) Lipschitz constant of $\nabla f_2(\boldsymbol{v})$ is:

A: 1 B: β C: $\beta\sqrt{K}$ D: βK E: None **??**

Efficient line search (Read)

A naive implementation of the **line search** step in the PSD algorithm on p. 3.18 minimizes

$$h_k(\alpha) \triangleq \Psi(\boldsymbol{x}_k + \alpha \boldsymbol{d}_k) \,.$$

When applied to a cost function of the general form (3.13) this would involve repeated matrix-vector multiplications of the form $\boldsymbol{B}_j(\boldsymbol{x}_k + \alpha \boldsymbol{d}_k)$, which is expensive. A more efficient approach is to precompute the matrix-vector products prior to performing the line search, noting that for the general (3.13) form:

$$h_k(\alpha) = \Psi(\boldsymbol{x}_k + \alpha \boldsymbol{d}_k) = \sum_{j=1}^{J} f_j(\boldsymbol{B}_j\,(\boldsymbol{x}_k + \alpha \boldsymbol{d}_k)) = \sum_{j=1}^{J} f_j\left(\boldsymbol{u}_j^{(k)} + \alpha \boldsymbol{v}_j^{(k)}\right), \ \boldsymbol{u}_j^{(k)} \triangleq \boldsymbol{B}_j \boldsymbol{x}_k, \ \boldsymbol{v}_j^{(k)} \triangleq \boldsymbol{B}_j \boldsymbol{d}_k.$$

Precomputing $\boldsymbol{u}_j^{(k)}$ and $\boldsymbol{v}_j^{(k)}$ prior to performing the line search avoids redundant matrix-vector products.

Furthermore, algorithms with line searches like PSD and PCG have an **recursive update** of the form:

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_k \boldsymbol{d}_k.$$

Multiplying both sides by $\boldsymbol{B}_j$ yields an efficient **recursive update** for the $\boldsymbol{u}_j$ vector (used in HW problems):

$$\boldsymbol{B}_j \boldsymbol{x}_{k+1} = \boldsymbol{B}_j \boldsymbol{x}_k + \alpha_k \boldsymbol{B}_j \boldsymbol{d}_k \Longrightarrow \boldsymbol{u}_j^{(k+1)} = \boldsymbol{u}_j^{(k)} + \alpha \boldsymbol{v}_j^{(k)}.$$

A key simplification here is that the **Lipschitz constant** of $\dot{h}_k$ does not use operator norms of any $\boldsymbol{B}_j$. (HW)

3.6 Convergence rates

Asymptotic convergence rates

When the cost function Ψ is locally strictly convex and twice differentiable near minimizer $\hat{\boldsymbol{x}}$, one can analyze the **asymptotic convergence rates** of PGD, PSD, and PCG. (See Fessler book Ch. 11.)

All three algorithms satsify inequalities of the following form for different values of c, ρ:

$$\left\|\boldsymbol{P}^{-1/2}(\boldsymbol{x}_{k+1} - \hat{\boldsymbol{x}})\right\|_2 \leq c\rho^k \left\|\boldsymbol{P}^{-1/2}(\boldsymbol{x}_0 - \hat{\boldsymbol{x}})\right\|_2 \Longrightarrow \lim_{k\to\infty} \left\|\boldsymbol{P}^{-1/2}(\boldsymbol{x}_k - \hat{\boldsymbol{x}})\right\|_2^{1/k} \leq \rho.$$

PGD and PSD produce sequences $\{\boldsymbol{x}_k\}$ that **converge linearly** [1, p. 32] to $\hat{\boldsymbol{x}}$ and

$$\sup_{\boldsymbol{x}_0} \lim_{k\to\infty} \left\|\boldsymbol{P}^{-1/2}(\boldsymbol{x}_k - \hat{\boldsymbol{x}})\right\|_2^{1/k} = \rho,$$

where ρ is called the **root convergence factor**. Define $\hat{\boldsymbol{H}} = \boldsymbol{P}^{1/2}\nabla^2\,\Psi(\hat{\boldsymbol{x}})\,\boldsymbol{P}^{1/2}$ and **condition number** $\kappa = \sigma_1(\hat{\boldsymbol{H}})/\sigma_N(\hat{\boldsymbol{H}})$.
This table shows the values of ρ.

Method	ρ	$\kappa = 10^2$
PGD standard step $\alpha = \dfrac{1}{\sigma_1(\hat{\boldsymbol{H}})}$	$\dfrac{\kappa - 1}{\kappa}$	0.99
PGD $\alpha_* = \dfrac{2}{\sigma_1(\hat{\boldsymbol{H}}) + \sigma_N(\hat{\boldsymbol{H}})}$	$\dfrac{\kappa - 1}{\kappa + 1}$	0.98
PSD with perfect line search	$\dfrac{\kappa - 1}{\kappa + 1}$	0.98
PCG with perfect line search	$\dfrac{\sqrt{\kappa} - 1}{\sqrt{\kappa} + 1}$	0.82

PCG converges **quadratically** [1, p. 45] and its ρ above matches a lower bound [1, p. 68].

Heavy ball method

One way to seek faster convergence is to use algorithms that have **momentum**.
An early momentum method is the **heavy ball** method [4, p. 64]. One way to write it is:

$$\begin{aligned} \boldsymbol{d}_k &= -\nabla \Psi(\boldsymbol{x}_k) + \beta_k \boldsymbol{d}_{k-1} \\ \boldsymbol{x}_{k+1} &= \boldsymbol{x}_k + \alpha_k \boldsymbol{d}_k, \end{aligned}$$

where $\alpha_k > 0$ and $\beta_k \geq 0$. The "search direction" $\boldsymbol{d}_k$ depends on both the gradient and the previous direction. Rearranging the 2nd equation to write $\boldsymbol{d}_k = (\boldsymbol{x}_{k+1} - \boldsymbol{x}_k)/\alpha_k$ and the combining yields this form

$$\boldsymbol{x}_{k+1} = \underbrace{\boldsymbol{x}_k - \alpha_k \nabla \Psi(\boldsymbol{x}_k)}_{\text{usual GD}} + \underbrace{\tilde{\beta}_k (\boldsymbol{x}_k - \boldsymbol{x}_{k-1})}_{\text{momentum}}, \qquad \tilde{\beta}_k \triangleq \beta_k \frac{\alpha_k}{\alpha_{k-1}}. \tag{3.14}$$

Convergence rate analysis (Read)

To analyze the convergence rate of this method we make two simplifications.

- We consider the case of constant step sizes $\alpha_k = \alpha$ and $\beta_k = \beta = \tilde{\beta}_k$
- We focus on a quadratic cost function $\Psi(\boldsymbol{x}) = \frac{1}{2} \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{y}\|_2^2$ where $M \times N$ $\boldsymbol{A}$ has full column rank, so there is a unique minimizer $\hat{\boldsymbol{x}}$. Note that $\nabla \Psi(\boldsymbol{x}) = \boldsymbol{A}'(\boldsymbol{A}\boldsymbol{x} - \boldsymbol{y}) = \boldsymbol{H}\boldsymbol{x} - \boldsymbol{b}$, where the Hessian is $\boldsymbol{H} = \boldsymbol{A}'\boldsymbol{A}$ and $\boldsymbol{b} = \boldsymbol{A}'\boldsymbol{y}$. The unique minimizer satisfies the **normal equations**: $\boldsymbol{H}\hat{\boldsymbol{x}} = \boldsymbol{b}$, so $\nabla \Psi(\boldsymbol{x}) = \boldsymbol{H}\boldsymbol{x} - \boldsymbol{H}\hat{\boldsymbol{x}} = \boldsymbol{H}(\boldsymbol{x} - \hat{\boldsymbol{x}})$.

EECS 551 analyzed the convergence rate of GD by relating $\boldsymbol{x}_{k+1} - \hat{\boldsymbol{x}}$ to $\boldsymbol{x}_k - \hat{\boldsymbol{x}}$. Here the recursion (3.14) depends on both $\boldsymbol{x}_k$ and $\boldsymbol{x}_{k-1}$ so we analyze the following two-state recursion:

$$\begin{bmatrix} \boldsymbol{x}_{k+1} - \hat{\boldsymbol{x}} \\ \boldsymbol{x}_k - \hat{\boldsymbol{x}} \end{bmatrix} = \begin{bmatrix} \boldsymbol{x}_k - \alpha \nabla \Psi(\boldsymbol{x}_k) + \beta(\boldsymbol{x}_k - \boldsymbol{x}_{k-1}) - \hat{\boldsymbol{x}} \\ \boldsymbol{x}_k - \hat{\boldsymbol{x}} \end{bmatrix} = \begin{bmatrix} \boldsymbol{x}_k - \hat{\boldsymbol{x}} - \alpha \boldsymbol{H}(\boldsymbol{x}_k - \hat{\boldsymbol{x}}) + \beta(\boldsymbol{x}_k - \boldsymbol{x}_{k-1}) \\ \boldsymbol{x}_k - \hat{\boldsymbol{x}} \end{bmatrix}$$
$$= \begin{bmatrix} \boldsymbol{x}_k - \hat{\boldsymbol{x}} - \alpha \boldsymbol{H}(\boldsymbol{x}_k - \hat{\boldsymbol{x}}) + \beta(\boldsymbol{x}_k - \hat{\boldsymbol{x}}) - \beta(\boldsymbol{x}_{k-1} - \hat{\boldsymbol{x}}) \\ \boldsymbol{x}_k - \hat{\boldsymbol{x}} \end{bmatrix}$$
$$= \boldsymbol{G} \begin{bmatrix} \boldsymbol{x}_k - \hat{\boldsymbol{x}} \\ \boldsymbol{x}_{k-1} - \hat{\boldsymbol{x}} \end{bmatrix}, \quad \boldsymbol{G} \triangleq \begin{bmatrix} (1+\beta)\boldsymbol{I} - \alpha \boldsymbol{H} & \beta \boldsymbol{I} \\ \boldsymbol{I} & \boldsymbol{0} \end{bmatrix}.$$

Because $\boldsymbol{H}$ is Hermitian, it has a unitary eigendecomposition $\boldsymbol{H} = \boldsymbol{V}\boldsymbol{\Lambda}\boldsymbol{V}'$, with eigenvalues $\lambda_i(\boldsymbol{H}) = \sigma_i^2(\boldsymbol{A})$. Writing the governing matrix $\boldsymbol{G}$ using this eigendecomposition:

$$\boldsymbol{G} = \begin{bmatrix} \boldsymbol{V} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{V} \end{bmatrix} \begin{bmatrix} (1+\beta)\boldsymbol{I} - \alpha \boldsymbol{\Lambda} & \beta \boldsymbol{I} \\ \boldsymbol{I} & \boldsymbol{0} \end{bmatrix} \begin{bmatrix} \boldsymbol{V} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{V} \end{bmatrix}' \text{ (diagonal blocks)}$$
$$= \begin{bmatrix} \boldsymbol{V} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{V} \end{bmatrix} \boldsymbol{\Pi} \begin{bmatrix} \boldsymbol{G}_1 & & \boldsymbol{0} \\ & \ddots & \\ \boldsymbol{0} & & \boldsymbol{G}_N \end{bmatrix} \boldsymbol{\Pi}' \begin{bmatrix} \boldsymbol{V} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{V} \end{bmatrix}', \quad \boldsymbol{G}_i = \begin{bmatrix} 1 + \beta - \alpha\lambda_i & \beta \\ 1 & 0 \end{bmatrix},$$

where $\boldsymbol{\Pi}$ is a $2N \times 2N$ permutation matrix. Thus $\mathrm{eig}\{\boldsymbol{G}\} = \cup_{i=1}^N \mathrm{eig}\{\boldsymbol{G}_i\}$, using several eigenvalue properties. The eigenvalues of $\boldsymbol{G}_i$ are the roots of its **characteristic polynomial**:

$$z^2 - (1 + \beta - \alpha\lambda_i)z + \beta.$$

If $\beta = 0$ then the nontrivial root is at $z = 1 - \alpha\lambda_i$, which is an expression see in the EECS 551 notes.

Otherwise the roots (eigenvalues of $\boldsymbol{G}_i$) are:

$$z = \frac{(1 + \beta - \alpha\lambda_i) \pm \sqrt{(1 + \beta - \alpha\lambda_i)^2 - 4\beta}}{2}.$$

For the fastest convergence, we would like to choose α and β to minimize $\max_i \rho(\boldsymbol{G}_i)$. One can show that the best choice is:

$$\alpha_* = \frac{4}{(\sigma_1(\boldsymbol{A}) + \sigma_N(\boldsymbol{A}))^2}, \qquad \beta_* = \frac{\sigma_1(\boldsymbol{A}) - \sigma_N(\boldsymbol{A})}{\sigma_1(\boldsymbol{A}) + \sigma_N(\boldsymbol{A})}.$$

For this choice, one can show that

$$\rho(\boldsymbol{G}) = \beta_* = \frac{\sigma_1(\boldsymbol{A}) - \sigma_N(\boldsymbol{A})}{\sigma_1(\boldsymbol{A}) + \sigma_N(\boldsymbol{A})} = \frac{\sqrt{\kappa} - 1}{\sqrt{\kappa} + 1},$$

where $\kappa = \sigma_1(\boldsymbol{H}) / \sigma_N(\boldsymbol{H}) = \sigma_1^2(\boldsymbol{A}) / \sigma_N^2(\boldsymbol{A})$.

Thus for the simple LS problem, the heavy ball method with its best choice of step-size parameters has the same rate as the conjugate gradient method with a perfect line search.

Of course in practice it is usually too expensive to determine $\sigma_1(\boldsymbol{A})$ and $\sigma_N(\boldsymbol{A})$, so we next seek more practical momentum methods that do not require these values.

Generalized convergence analysis of PGD

Define. The gradient of Ψ is $\boldsymbol{S}$-**Lipschitz continuous** on $\mathbb{R}^N$ for an invertible matrix $\boldsymbol{S}$ iff

$$\left\|\boldsymbol{S}^{-1}\left(\nabla \Psi(\boldsymbol{x}) - \nabla \Psi(\boldsymbol{z})\right)\right\|_2 \leq \left\|\boldsymbol{S}'\left(\boldsymbol{x} - \boldsymbol{z}\right)\right\|_2, \qquad \forall \boldsymbol{x}, \boldsymbol{z} \in \mathbb{R}^N. \tag{3.15}$$

If $\boldsymbol{S} = \sqrt{\mathcal{L}}\boldsymbol{I}$, then the $\boldsymbol{S}$-Lipschitz condition (3.15) simplifies to the classic Lipschitz continuity condition:

$$\|\nabla \Psi(\boldsymbol{x}) - \nabla \Psi(\boldsymbol{z})\|_2 \leq \mathcal{L}\,\|\boldsymbol{x} - \boldsymbol{z}\|_2, \qquad \forall \boldsymbol{x}, \boldsymbol{z} \in \mathbb{R}^N. \tag{3.16}$$

Theorem (PGD convergence). If $\nabla \Psi$ satisfies (3.15) and for some $0 < \alpha$

$$\alpha \boldsymbol{P}'\boldsymbol{S}\boldsymbol{S}'\boldsymbol{P} \prec \boldsymbol{P} + \boldsymbol{P}', \tag{3.17}$$

then (i) the PGD algorithm

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \alpha \boldsymbol{P} \nabla \Psi(\boldsymbol{x}_k) \tag{3.18}$$

monotonically decreases Ψ [10] because for $\boldsymbol{g} = \nabla \Psi(\boldsymbol{x}_k)$:

$$\Psi(\boldsymbol{x}_k) - \Psi(\boldsymbol{x}_{k+1}) \geq \frac{\alpha}{2}\boldsymbol{g}'\left(\boldsymbol{P} + \boldsymbol{P}' - \alpha \boldsymbol{P}'\boldsymbol{S}\boldsymbol{S}'\boldsymbol{P}\right)\boldsymbol{g},$$

and (ii) $\|\nabla \Psi(\boldsymbol{x}_k)\| \to 0$.

In the usual case where $\boldsymbol{P}$ is symmetric positive definite, (3.17) simplifies to $\alpha \boldsymbol{S}\boldsymbol{S}' \prec 2\boldsymbol{P}^{-1}$ and when that condition holds then $\left\|\boldsymbol{P}^{-1/2}(\boldsymbol{x}_k - \hat{\boldsymbol{x}})\right\|_2$ converges monotonically to zero [10].

If we choose $\alpha \boldsymbol{P} = (\boldsymbol{S}\boldsymbol{S}')^{-1}$, then PGD is equivalent to a **majorize-minimize** (**MM**) method (discussed later) and the cost function decrease has the following bound:

$$\Psi(\boldsymbol{x}_k) - \Psi(\hat{\boldsymbol{x}}) \leq \frac{\|\boldsymbol{S}'(\boldsymbol{x}_0 - \hat{\boldsymbol{x}})\|^2}{2k}, \quad k \geq 1. \tag{3.19}$$

The bound above is the "classical" textbook formula. In 2014 the following **tight bound** was found [5]:

$$\Psi(\boldsymbol{x}_k) - \Psi(\hat{\boldsymbol{x}}) \leq \frac{\|\boldsymbol{S}'(\boldsymbol{x}_0 - \hat{\boldsymbol{x}})\|^2}{4k+2}, \quad k \geq 1. \tag{3.20}$$

It is tight because there is a Huber-like function Ψ for which GD meets that rate.

Why generalize? Units of Lipschitz constant

Consider $\Psi(\boldsymbol{x}) = \frac{1}{2}\|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{y}\|_2^2$ where $\boldsymbol{A}$ is 2×2 diagonal with units:

a_{11}	ampere	x_1	ohm	y_1	volt
a_{22}	volt/m	x_2	m	y_2	volt

What are the units of the Lipschitz constant of $\nabla \Psi$?

A: ampere volt / m B: ampere^2 C: volt^2 / m^2 D: ohm m E: none of these $\boxed{??}$

Generalized Nesterov fast gradient method (FGM)

The following is a slight generalization of the **fast gradient method** (**FGM**) of **Nesterov**, also known as **accelerated gradient descent** and **Nesterov accelerated gradient** (**NAG**):
Initialize $t_0 = 1$ and $\boldsymbol{z}_0 = \boldsymbol{x}_0$ then for $k = 0, 1, \ldots$:

$$\begin{aligned} t_{k+1} &= \left(1 + \sqrt{1 + 4t_k^2}\right)/2 & \\ \boldsymbol{x}_{k+1} &= \boldsymbol{z}_k - \left[\boldsymbol{S}\boldsymbol{S}'\right]^{-1} \nabla \Psi(\boldsymbol{z}_k) & \text{gradient step} \\ \boldsymbol{z}_{k+1} &= \boldsymbol{x}_{k+1} + \frac{t_k - 1}{t_{k+1}} (\boldsymbol{x}_{k+1} - \boldsymbol{x}_k) \,. & \text{momentum} \end{aligned}$$

If $t_k = 1$ for all k then FGM reverts to ordinary GD.

Theorem If Ψ is convex and has an $\boldsymbol{S}$-Lipschitz gradient, then this generalized FGM satisfies:

$$\Psi(\boldsymbol{x}_k) - \Psi(\hat{\boldsymbol{x}}) \leq \frac{2 \left\|\boldsymbol{S}' (\boldsymbol{x}_0 - \hat{\boldsymbol{x}})\right\|^2}{k^2}. \tag{3.21}$$

In words, the worst-case rate of decrease of the cost function Ψ is $O(1/k^2)$.
This is a huge improvement over the $O(1/k)$ rate of PGD in (3.19).
However, worst-case analysis may be pessimistic for your favorite application.
For example, if Ψ is quadratic on $\mathbb{R}^N$, then CG converges in N iterations.

3.7 First-order methods

The most famous **second-order** method is **Newton's method**:

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \left(\nabla^2 \Psi(\boldsymbol{x}_k)\right)^{-1} \nabla \Psi(\boldsymbol{x}_k) \,.$$

This method is impractical for large-scale problems so we focus on **first-order methods** [1, p. 7].

General first-order method classes

- General **first-order** (GFO) method:

$$\boldsymbol{x}_{k+1} = \text{function}(\boldsymbol{x}_0, \Psi(\boldsymbol{x}_0), \nabla \Psi(\boldsymbol{x}_0), \ldots, \Psi(\boldsymbol{x}_k), \nabla \Psi(\boldsymbol{x}_k)). \tag{3.22}$$

- First-order (FO) methods with fixed step-size coefficients:

$$\boldsymbol{x}_{n+1} = \boldsymbol{x}_n - \frac{1}{L} \sum_{k=0}^{n} h_{n+1,k} \nabla \Psi(\boldsymbol{x}_k) \,. \tag{3.23}$$

Which of the algorithms discussed so far are FO (fixed-step) methods?

A: PGD, FGM, PSD, PCG B: PGD, FGM, PSD C: PGD, PSD D: PGD, FGM E: PGD $\boxed{??}$

Example: Barzilai-Borwein gradient method

Barzilai & Borwein, 1988: [11]

$$\begin{aligned} \boldsymbol{g}_k &\triangleq \nabla \Psi(\boldsymbol{x}_k) \\ \alpha_k &= \frac{\|\boldsymbol{x}_k - \boldsymbol{x}_{k-1}\|_2^2}{\langle \boldsymbol{x}_k - \boldsymbol{x}_{k-1},\, \boldsymbol{g}_k - \boldsymbol{g}_{k-1} \rangle} \\ \boldsymbol{x}_{k+1} &= \boldsymbol{x}_k - \alpha_k \boldsymbol{g}_k. \end{aligned}$$

- In "general" first-order (GFO) class, but
- not in class FO with fixed step-size coefficients.

Recent research questions

- Analyze convergence rate of FO for any given step-size coefficients $\{h_{n,k}\}$
- Optimize step-size coefficients $\{h_{n,k}\}$
  - fast convergence
  - efficient recursive implementation
  - universal (design *prior* to iterating, independent of L)
- How much better could one do with GFO?

Nesterov's fast gradient method is FO

Nesterov (1983) iteration [12, 13] expressed in efficient recursive form: Initialize: $t_0 = 1$, $\boldsymbol{z}_0 = \boldsymbol{x}_0$

$$\begin{aligned}
\boldsymbol{z}_{n+1} &= \boldsymbol{x}_n - \frac{1}{L}\nabla\,\Psi(\boldsymbol{x}_n) && \text{(usual GD update)} \\
t_{n+1} &= \frac{1}{2}\left(1 + \sqrt{1 + 4t_n^2}\right) && \text{(magic momentum factors)} \\
\boldsymbol{x}_{n+1} &= \boldsymbol{z}_{n+1} + \frac{t_n - 1}{t_{n+1}}(\boldsymbol{z}_{n+1} - \boldsymbol{z}_n) && \text{(update with momentum)} \\
&= (1+\gamma_n)\boldsymbol{z}_{n+1} - \gamma_n \boldsymbol{z}_n, && \gamma_n = \frac{t_n - 1}{t_{n+1}} > 0.
\end{aligned}$$

FGM1 is in class FO [7] (for analysis, not implementation!):

$$\boldsymbol{x}_{n+1} = \boldsymbol{x}_n - \frac{1}{L}\sum_{k=0}^{n} h_{n+1,k}\nabla\,\Psi(\boldsymbol{x}_k)$$

$$h_{n+1,k} = \begin{cases} \dfrac{t_n - 1}{t_{n+1}} h_{n,k}, & k = 0,\ldots,n-2 \\ \dfrac{t_n - 1}{t_{n+1}}\left(h_{n,n-1} - 1\right), & k = n-1 \\ 1 + \dfrac{t_n - 1}{t_{n+1}}, & k = n. \end{cases}
\qquad
\begin{bmatrix}
1 & 0 & 0 & 0 & 0 & 0 \\
0 & 1.25 & 0 & 0 & 0 & 0 \\
0 & 0.10 & 1.40 & 0 & 0 & 0 \\
0 & 0.05 & 0.20 & 1.50 & 0 & 0 \\
0 & 0.03 & 0.11 & 0.29 & 1.57 & 0 \\
0 & 0.02 & 0.07 & 0.18 & 0.36 & 1.62
\end{bmatrix}$$

Nesterov's FGM1 optimal convergence rate

Shown by Nesterov to be $O(1/n^2)$ for "primary" sequence $\{\boldsymbol{z}_n\}$ [7, eqn. (3.5)]:

$$\Psi(\boldsymbol{z}_n) - \Psi(\boldsymbol{x}_\star) \leq \frac{L \left\|\boldsymbol{x}_0 - \boldsymbol{x}_\star\right\|_2^2}{2t_{n-1}^2} \leq \frac{2L \left\|\boldsymbol{x}_0 - \boldsymbol{x}_\star\right\|_2^2}{(n+1)^2}. \tag{3.24}$$

Nesterov [1, p. 59-61] constructed "the worst function in the world," a simple quadratic function Ψ, with a tridiagonal Hessian matrix similar to $\boldsymbol{C}'\boldsymbol{C}$ for first-order differences, such that, for any general FO method:

$$\frac{\frac{3}{32} L \left\|\boldsymbol{x}_0 - \boldsymbol{x}_\star\right\|_2^2}{(n+1)^2} \leq \Psi(\boldsymbol{x}_n) - \Psi(\boldsymbol{x}_\star).$$

Thus the $O(1/n^2)$ rate of FGM1 is "optimal" in a **big-O** sense.

Bound on convergence rate of "secondary" sequence $\{\boldsymbol{x}_n\}$ [7, eqn. (5.5)]:

$$\Psi(\boldsymbol{x}_n) - \Psi(\boldsymbol{x}_\star) \leq \frac{L \left\|\boldsymbol{x}_0 - \boldsymbol{x}_\star\right\|_2^2}{2t_n^2} \leq \frac{2L \left\|\boldsymbol{x}_0 - \boldsymbol{x}_\star\right\|_2^2}{(n+2)^2}. \tag{3.25}$$

The bounds (3.24) and (3.25) are **asymptotically** tight [6].

To reach a cost within ε of the minimum $\Psi(\boldsymbol{x}_\star)$, how many iterations are needed?

A: $O(1)$ B: $O(1/\sqrt{\epsilon})$ C: $O(1/\epsilon)$ D: $O(1/\epsilon^2)$ E: $O(1/\epsilon^4)$ $\boxed{??}$

The gap between 2 and $3/32$ suggests we can do better, and we can, thanks to recent work from UM.

Optimized gradient method (OGM)

Recall general family of **first-order** (FO) methods (3.23) with fixed step-size coefficients:

$$\boldsymbol{x}_{n+1} = \boldsymbol{x}_n - \frac{1}{L} \sum_{k=0}^{n} h_{n+1,k} \nabla \Psi(\boldsymbol{x}_k) \, .$$

Inspired by [5], recent work by former UM ECE PhD student Donghwan Kim [7]:

- Analyze (*i.e.*, bound) convergence rate as a function of
  - number of iterations N
  - Lipschitz constant L
  - step-size coefficients $H = \{h_{n+1,k}\}$
  - initial distance to a solution: $R = \|\boldsymbol{x}_0 - \boldsymbol{x}_\star\|$.
- Optimize H by minimizing the bound.
  "Optimizing the optimizer" (meta-optimization?)
- Seek an equivalent recursive form for efficient implementation.

(... many pages of derivations ...)

- Optimized step-size coefficients [7]:

$$H^* \quad : \quad h_{n+1,k} = \begin{cases} \frac{\theta_n - 1}{\theta_{n+1}} h_{n,k}, & k = 0, \ldots, n-2 \\ \frac{\theta_n - 1}{\theta_{n+1}} \left(h_{n,n-1} - 1 \right), & k = n-1 \\ 1 + \frac{2\theta_n - 1}{\theta_{n+1}}, & k = n. \end{cases}$$

$$\theta_n = \begin{cases} 1, & n = 0 \\ \frac{1}{2}\left(1 + \sqrt{1 + 4\theta_{n-1}^2}\right), & n = 1, \ldots, N-1 \\ \frac{1}{2}\left(1 + \sqrt{1 + 8\theta_{n-1}^2}\right), & n = N. \end{cases}$$

- Analytical convergence bound for FO method with these optimized step-size coefficients [7, eqn. (6.17)]:

$$\Psi(\boldsymbol{x}_N) - \Psi(\boldsymbol{x}_\star) \leq \frac{1 L \left\| \boldsymbol{x}_0 - \boldsymbol{x}_\star \right\|_2^2}{2\theta_N^2} \leq \frac{1 L \left\| \boldsymbol{x}_0 - \boldsymbol{x}_\star \right\|_2^2}{(N+1)(N+1+\sqrt{2})} \leq \frac{1 L \left\| \boldsymbol{x}_0 - \boldsymbol{x}_\star \right\|_2^2}{(N+1)^2}. \tag{3.26}$$

- Of course the bound is $O(1/N^2)$, but the constant is twice better than that of Nesterov's FGM in (3.24).

Optimized gradient method (OGM) recursion

Donghwan Kim also found an efficient recursive algorithm [7]. Initialize: $\theta_0 = 1$, $\boldsymbol{z}_0 = \boldsymbol{x}_0$

$$
\begin{aligned}
\boldsymbol{z}_{n+1} &= \boldsymbol{x}_n - \frac{1}{L} \nabla \Psi(\boldsymbol{x}_n) \\
\theta_{n+1} &= \begin{cases} \frac{1}{2}\left(1 + \sqrt{1 + 4\theta_n^2}\right), & n \leq N - 2 \\ \frac{1}{2}\left(1 + \sqrt{1 + 8\theta_n^2}\right), & n = N - 1 \end{cases} \\
\boldsymbol{x}_{n+1} &= \boldsymbol{z}_{n+1} + \frac{\theta_n - 1}{\theta_{n+1}} (\boldsymbol{z}_{n+1} - \boldsymbol{z}_n) + \underbrace{\frac{\theta_n}{\theta_{n+1}} (\boldsymbol{z}_{n+1} - \boldsymbol{x}_n)}_{\text{new momentum}}.
\end{aligned}
$$

Reverts to Nesterov's FGM if one removes the new term.

- Very simple modification of existing Nesterov code.
- Factor of 2 better bound than Nesterov's "optimal" FGM.
- Similar momentum to Güler's 1992 proximal point algorithm [14].
- Inconvenience: must pick N in advance to use bound (3.26) on $\Psi(\boldsymbol{x}_N)$.
- Convergence bound for *every iteration* of the "primary" sequence [15, eqn. (20)]:

$$
\Psi(\boldsymbol{z}_n) - \Psi(\boldsymbol{x}_\star) \leq \frac{1L \left\|\boldsymbol{x}_0 - \boldsymbol{x}_\star\right\|_2^2}{4t_{n-1}^2} \leq \frac{1L \left\|\boldsymbol{x}_0 - \boldsymbol{x}_\star\right\|_2^2}{(n+1)^2}.
$$

This bound is **asymptotically tight** [15, p. 198].

Recent refinement of OGM

Newer version OGM' [15, p. 199]:

$$
\begin{aligned}
\boldsymbol{z}_{n+1} &= \boldsymbol{x}_n - \frac{1}{L} \nabla \Psi(\boldsymbol{x}_n) \\
t_{n+1} &= \frac{1}{2}\left(1 + \sqrt{1 + 4t_n^2}\right) \qquad \text{(momentum factors)} \\
\boldsymbol{x}_{n+1} &= \underbrace{\boldsymbol{x}_n - \frac{1 + t_n/t_{n+1}}{L} \nabla \Psi(\boldsymbol{x}_n)}_{\text{over-relaxed GD}} + \underbrace{\frac{t_n - 1}{t_{n+1}} (\boldsymbol{z}_{n+1} - \boldsymbol{z}_n)}_{\text{FGM momentum}}.
\end{aligned}
$$

- Convergence bound for *every iteration* on the "primary" sequence [15, eqn. (25)]:

$$
\Psi(\boldsymbol{z}_n) - \Psi(\boldsymbol{x}_\star) \leq \frac{1L \, \|\boldsymbol{x}_0 - \boldsymbol{x}_\star\|_2^2}{4t_{n-1}^2} \leq \frac{1L \, \|\boldsymbol{x}_0 - \boldsymbol{x}_\star\|_2^2}{(n+1)^2}.
$$

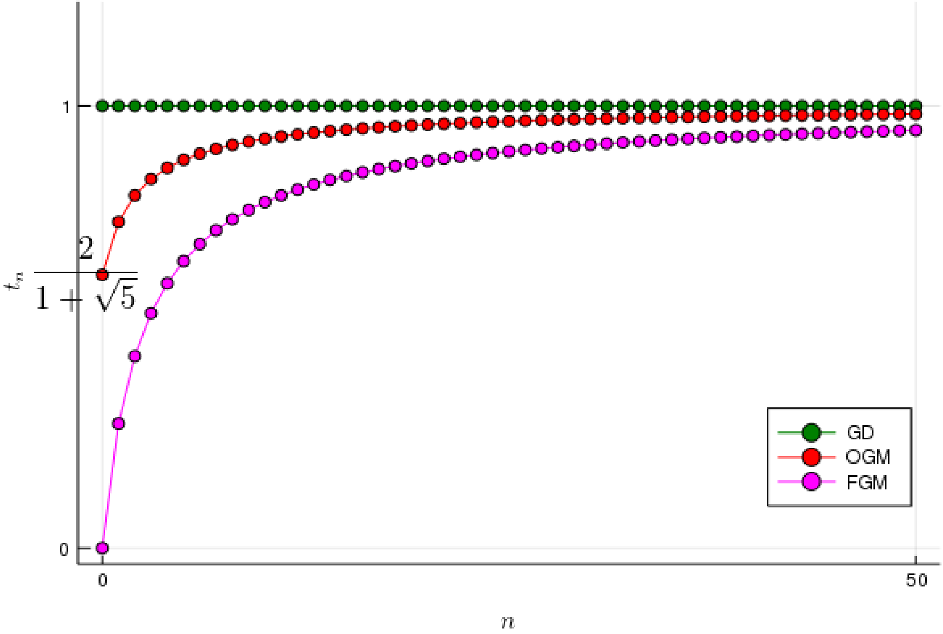
- Simpler and more practical implementation.
- Need not pick N in advance.

One can show $t_n^2 \geq (n+1)^2/4$ for $n > 1$ [15, p. 197].

OGM’ momentum factors illustrated

$$\boldsymbol{x}_{n+1} = \boldsymbol{x}_n - \frac{1 + t_n/t_{n+1}}{L} \nabla \Psi(\boldsymbol{x}_n) + \frac{t_n - 1}{t_{n+1}} (\boldsymbol{z}_{n+1} - \boldsymbol{z}_n)$$

Intuition:
$1 + t_n/t_{n+1} \to 2$ as $n \to \infty$



Optimized gradient method (OGM) is an optimal GFO method (!)

Recall that within the class of first-order (FO) methods with fixed step sizes:

$$\boldsymbol{x}_{n+1} = \boldsymbol{x}_n - \frac{1}{L} \sum_{k=0}^{n} h_{n+1,k} \nabla \Psi(\boldsymbol{x}_k),$$

OGM is based on optimized $\{h_{n,k}\}$ step sizes and provides the convergence rate upper bound:

$$\Psi(\boldsymbol{x}_N) - \Psi(\boldsymbol{x}_\star) \leq \frac{L \|\boldsymbol{x}_0 - \boldsymbol{x}_\star\|_2^2}{2\theta_N^2} \leq \frac{L \|\boldsymbol{x}_0 - \boldsymbol{x}_\star\|_2^2}{N^2}.$$

Recently Y. Drori [16, Thm. 3] considered the class of general FO (GFO) methods:

$$\boldsymbol{x}_{n+1} = F(\boldsymbol{x}_0, \Psi(\boldsymbol{x}_0), \nabla \Psi(\boldsymbol{x}_0), \ldots, \Psi(\boldsymbol{x}_n), \nabla \Psi(\boldsymbol{x}_n)),$$

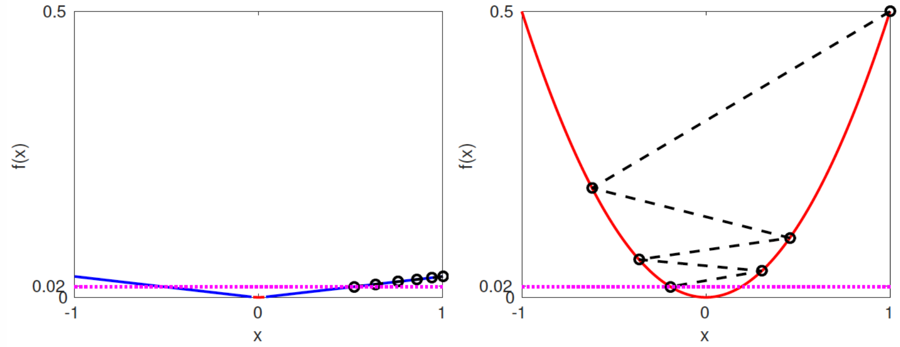
and showed for $d > N$ (large-scale problems), *any* algorithm in this GFO class has a function Ψ such that

$$\frac{L \|\boldsymbol{x}_0 - \boldsymbol{x}_\star\|_2^2}{2\theta_N^2} \leq \Psi(\boldsymbol{x}_N) - \Psi(\boldsymbol{x}_\star),$$

Thus OGM has optimal (worst-case) complexity among all GFO methods, not just fixed-step FO methods.

Worst-case functions for OGM

From [7, eqn. (8.1)] and [15, Thm. 5.1], the worst-case behavior of OGM is for a **Huber** function and a **quadratic** function.



(c) $N = 5$: $f_{1,\mathrm{OGM}}\ (\boldsymbol{x};5)$ (d) $N = 5$: $f_2(\boldsymbol{x})$

For $R \triangleq \|\boldsymbol{x}_0 - \boldsymbol{x}_\star\|$, the worst-case behavior is:

$$\Psi(\boldsymbol{x}_N) - \Psi(\boldsymbol{x}_\star) = \frac{LR^2}{2\theta_N^2} \leq \frac{LR^2}{(N+1)(N+1+\sqrt{2})} \leq \frac{LR^2}{(N+1)^2}.$$

Monotonicity

In these examples, the cost function $\Psi(\boldsymbol{x}_n)$ happens to decrease monotonically. In general, neither FGM nor OGM guarantee non-increasing cost functions, despite the *bound* $1/N^2$ being strictly decreasing. Nesterov [1, p. 71] states that optimal methods in general do not ensure that $\Psi(\boldsymbol{x}_{k+1}) \leq \Psi(\boldsymbol{x}_k)$.

3.8 Machine learning via logistic regression for binary classification

To learn weights $\boldsymbol{x} \in \mathbb{R}^N$ of a binary classifier given feature vectors $\{\boldsymbol{v}_m\} \in \mathbb{R}^N$ (training data) and labels $\{y_m = \pm 1 \ : \ m = 1, \ldots, M\}$, we can minimize a cost function with a regularization parameter $\beta > 0$:

$$\hat{\boldsymbol{x}} = \arg\min_{\boldsymbol{x}} \Psi(\boldsymbol{x}), \qquad \Psi(\boldsymbol{x}) = \sum_{m=1}^{M} \psi(y_m \langle \boldsymbol{x},\, \boldsymbol{v}_m \rangle) + \beta \frac{1}{2} \|\boldsymbol{x}\|_2^2 . \tag{3.27}$$

Want:

- $\langle \boldsymbol{x},\, \boldsymbol{v}_m \rangle > 0$ if $y_m = +1$ and
- $\langle \boldsymbol{x},\, \boldsymbol{v}_m \rangle < 0$ if $y_m = -1$,
- *i.e.*, $\langle \boldsymbol{x},\, y_m \boldsymbol{v}_m \rangle > 0$,

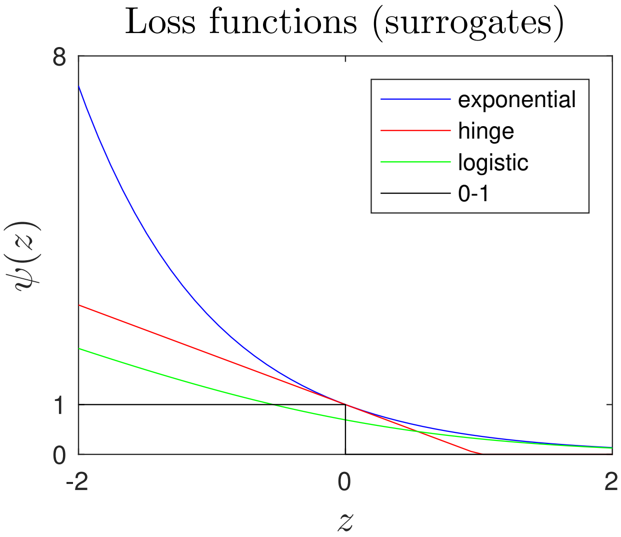
so that $\operatorname{sign}(\langle \boldsymbol{x},\, \boldsymbol{v}_m \rangle)$ is a reasonable classifier.

Logistic loss function has a Lipschitz derivative:

$$\psi(z) = \log\left(1 + \mathrm{e}^{-z}\right)$$

$$\dot{\psi}(z) = \frac{-1}{\mathrm{e}^z + 1}$$

$$\ddot{\psi}(z) = \frac{\mathrm{e}^z}{(\mathrm{e}^z + 1)^2} \in \left(0, \frac{1}{4}\right] .$$



The logistic regression cost function (3.27) is a special case of the "general inverse problem" (3.13). (?)

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

$$\boldsymbol{A} =$$

A regularization term like $\frac{\beta}{2}\|\boldsymbol{x}\|_2^2$ is especially important in the typical case where the feature vector dimension N is large relative to the sample size M.

For gradient-based optimization, we need the cost function gradient and a Lipschitz constant:

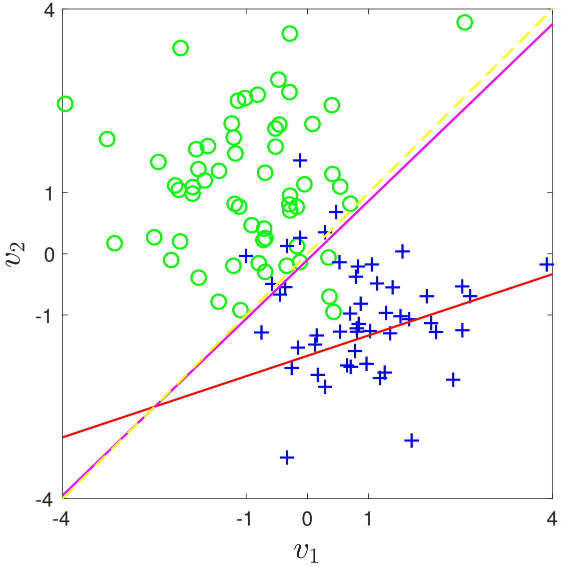
$$\nabla \Psi(\boldsymbol{x}) = \boldsymbol{A}\,\dot{\psi}\,.(\boldsymbol{A}'\boldsymbol{x}) + \beta\boldsymbol{x} \Longrightarrow L_{\nabla \Psi} \leq \|\boldsymbol{A}\|_2^2 L_{\dot{\psi}} + \beta = \frac{1}{4}\|\boldsymbol{A}\|_2^2 + \beta.$$

Practical implementation:

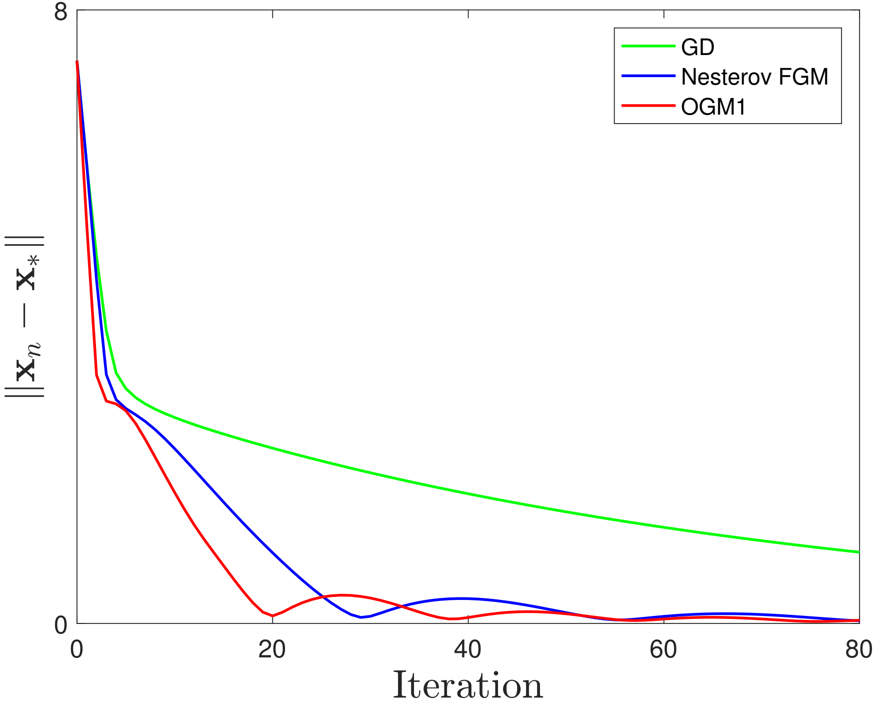
- Normalizing each column of $\boldsymbol{A}$ to unit norm can help keep e^z from overflowing.
- Tuning β should use cross validation or other such tools from machine learning.
- The cost function is convex with Lipschitz gradient, so it is well-suited for FGM and OGM.
- When feature dimension N is very 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 [17]) for optimization.

Numerical Results: logistic regression

Labeled training data (green and blue points);
initial decision boundary (red);
final decision boundary (magenta);
ideal boundary (yellow).
$M = 100$, $N = 7$ (cf "large scale" ?)

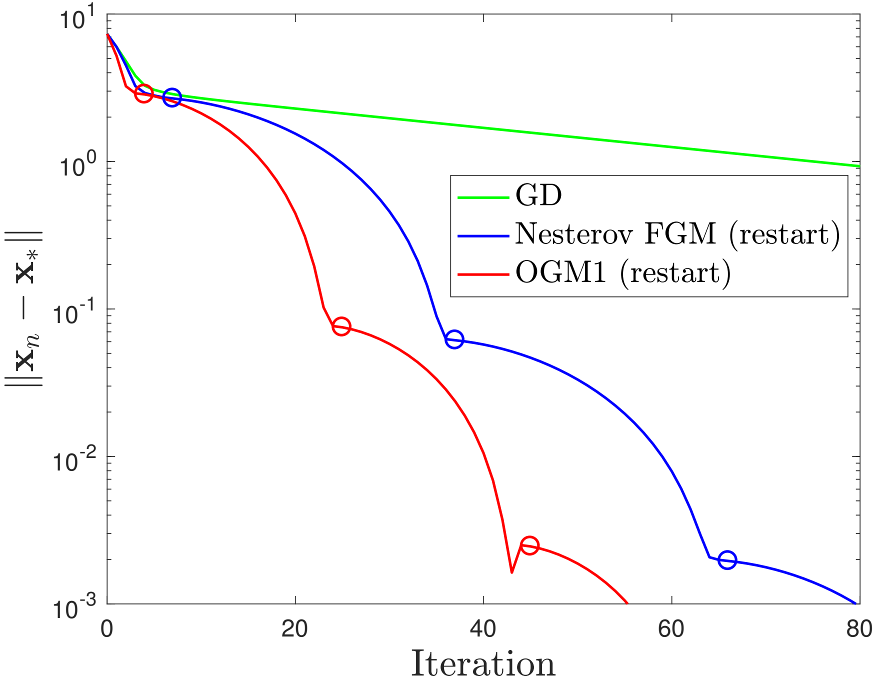


Numerical Results: convergence rates



OGM faster than FGM in early iterations...

Adaptive restart of accelerated GD



FGM restart, O'Donoghue & Candès, 2015 [19]
OGM restart [20]

Adaptive restart of OGM

Recall:

$$\boldsymbol{x}_{n+1} = \boldsymbol{x}_n - \frac{1 + t_n/t_{n+1}}{L} \nabla \Psi(\boldsymbol{x}_n) + \frac{t_n - 1}{t_{n+1}} (\boldsymbol{z}_{n+1} - \boldsymbol{z}_n)$$

Heuristic: restart momentum (set $t_n = 1$) if

$$\langle -\nabla \Psi(\boldsymbol{x}_n),\, \boldsymbol{z}_{n+1} - \boldsymbol{z}_n \rangle < 0.$$

This modified method, OGM-restart, has a better worst-case convergence bound than OGM for the class of convex cost functions with L-Lipschitz smooth gradients. (?)

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

Define. A function $f : \mathbb{R}^N \mapsto \mathbb{R}$ is **strongly convex** with parameter $\mu > 0$ iff it is **convex** and

$$f(\boldsymbol{x}) \geq f(\boldsymbol{z}) + \langle \nabla f(\boldsymbol{z}),\, \boldsymbol{x} - \boldsymbol{z} \rangle + \mu \frac{1}{2} \|\boldsymbol{x} - \boldsymbol{z}\|_2^2, \quad \forall \boldsymbol{x}, \boldsymbol{z} \in \mathbb{R}^N.$$

Smooth cost functions are often *locally* **strongly convex**, but rarely are the cost functions of interest in modern signal processing (globally) **strongly convex**.

Formal analysis of OGM for strongly convex quadratic functions is in [20].

Code: `https://gitlab.eecs.umich.edu/michigan-fast-optimization/ogm-adaptive-restart`

3.9 Summary

This chapter summarizes some of the most important gradient-based algorithms for solving unconstrained optimization problems with differentiable cost functions.

All of the methods discussed here require computing the gradient $\nabla \Psi(\boldsymbol{x}_k)$ each iteration, and often that is the most expensive operation.

Some of the algorithms (PSD, PCG) also require a **line search** step. A line search is itself a 1D optimization problem that requires evaluating the cost function or its gradient multiple times, and those evaluations can add considerable expense for general cost functions.

For cost functions of the form (3.13), where each component function f_j and its gradient are easy to evaluate, one can perform a line search quite efficiently, as described on p. 3.33.

The set of cost functions of the form (3.13), where each f_j has a Lipschitz continuous gradient, is a strict subset of the set of cost functions Ψ having Lipschitz continuous gradients. (?)

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

Recent work made a version of OGM with a line search [21].

Bibliography

[1] Y. Nesterov. *Introductory lectures on convex optimization: A basic course*. Springer, 2004 (cit. on pp. 3.8, 3.9, 3.34, 3.41, 3.44, 3.51).

[2] R. C. Fair. “On the robust estimation of econometric models”. In: *Ann. Econ. Social Measurement* 2 (Oct. 1974), 667–77 (cit. on p. 3.12).

[3] 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. 3.12).

[4] B. T. Polyak. *Introduction to optimization*. New York: Optimization Software Inc, 1987 (cit. on pp. 3.14, 3.35).

[5] 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. 3.14, 3.16, 3.39, 3.45).

[6] 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 pp. 3.14, 3.44).

[7] 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. 3.16, 3.18, 3.43, 3.44, 3.45, 3.46, 3.47, 3.51).

[8] A. Cauchy. “Methode générale pour la résolution des systems d’équations simultanées”. In: *Comp. Rend. Sci. Paris* 25 (1847), 536–8 (cit. on p. 3.18).

[9] A. Florescu, E. Chouzenoux, J-C. Pesquet, P. Ciuciu, and S. Ciochina. “A majorize-minimize memory gradient method for complex-valued inverse problems”. In: *Signal Processing* 103 (Oct. 2014), 285–95 (cit. on pp. 3.25, 3.26).

[10] J. A. Fessler. *Image reconstruction: Algorithms and analysis*. Book in preparation. ., 2006 (cit. on pp. 3.31, 3.38, 3.39).

[11] J. Barzilai and J. Borwein. “Two-point step size gradient methods”. In: *IMA J. Numerical Analysis* 8.1 (1988), 141–8 (cit. on p. 3.42).

[12] Y. Nesterov. “A method for unconstrained convex minimization problem with the rate of convergence $O(1/k^2)$”. In: *Dokl. Akad. Nauk. USSR* 269.3 (1983), 543–7 (cit. on p. 3.43).

[13] Y. Nesterov. “Smooth minimization of non-smooth functions”. In: *Mathematical Programming* 103.1 (May 2005), 127–52 (cit. on p. 3.43).

[14] O. Güler. “New proximal point algorithms for convex minimization”. In: *SIAM J. Optim.* 2.4 (1992), 649–64 (cit. on p. 3.47).

[15] D. Kim and J. A. Fessler. “On the convergence analysis of the optimized gradient methods”. In: *J. Optim. Theory Appl.* 172.1 (Jan. 2017), 187–205 (cit. on pp. 3.47, 3.48, 3.51).

[16] Y. Drori. “The exact information-based complexity of smooth convex minimization”. In: *J. Complexity* 39 (Apr. 2017), 1–16 (cit. on p. 3.50).

[17] 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. 3.53).

[18] D. Böhning and B. G. Lindsay. “Monotonicity of quadratic approximation algorithms”. In: *Ann. Inst. Stat. Math.* 40.4 (Dec. 1988), 641–63.

[19] 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. 3.56).

[20] 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 pp. 3.56, 3.57).

[21] Y. Drori and A. B. Taylor. *Efficient first-order methods for convex minimization: a constructive approach*. 2018 (cit. on p. 3.58).