Basics of Numerical Optimization: Iterative Methods

Ju Sun
Computer Science & Engineering
University of Minnesota, Twin Cities

September 27, 2023

Find global minimum

$$\min_{\boldsymbol{x}} \; f(\boldsymbol{x})$$

Grid search: incurs $O\left(\varepsilon^{-n}\right)$ cost

Smart search

> **1st-order necessary condition**: Assume f is 1st-order differentiable at $\boldsymbol{x}_0$. If $\boldsymbol{x}_0$ is a local minimizer, then $\nabla f(\boldsymbol{x}_0) = \mathbf{0}$.

$\boldsymbol{x}$ with $\nabla f(\boldsymbol{x}) = \mathbf{0}$: **1st-order stationary point** (1OSP)

> **2nd-order necessary condition**: Assume $f(\boldsymbol{x})$ is 2-order differentiable at $\boldsymbol{x}_0$. If $\boldsymbol{x}_0$ is a local min, $\nabla f(\boldsymbol{x}_0) = \mathbf{0}$ and $\nabla^2 f(\boldsymbol{x}_0) \succeq \mathbf{0}$.

$\boldsymbol{x}$ with $\nabla f(\boldsymbol{x}) = \mathbf{0}$ and $\nabla^2 f(\boldsymbol{x}) \succeq \mathbf{0}$: **2nd-order stationary point** (2OSP)

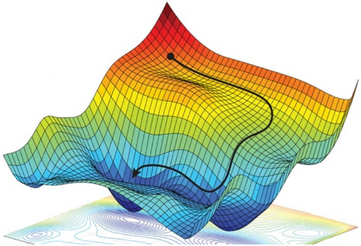
Smart search

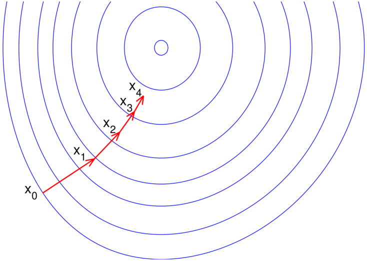
$\boldsymbol{x}$ with $\nabla f(\boldsymbol{x}) = \mathbf{0}$: **1st-order stationary point** (1OSP)
$\boldsymbol{x}$ with $\nabla f(\boldsymbol{x}) = \mathbf{0}$ and $\nabla^2 f(\boldsymbol{x}) \succeq \mathbf{0}$: **2nd-order stationary point** (2OSP)

- **Analytic method**: find 1OSP's using gradient first, then study them using Hessian — for simple functions! e.g., $f(\boldsymbol{x}) = \|\boldsymbol{y} - \boldsymbol{A}\boldsymbol{x}\|_2^2$, or $f(x, y) = x^2 y^2 - x^3 y + y^2 - 1$)
- **Iterative methods**: find 1OSP's/2OSP's by making consecutive small movements

This lecture: **iterative methods**

Iterative methods





Credit: aria42.com

Two questions: what direction to move, and how far to move

Two possibilities:

- **Line-search methods**: direction first, size second
- **Trust-region methods**: size first, direction second

Outline

Framework of line-search methods

A generic line search algorithm

Input: initialization $\boldsymbol{x}_0$, stopping criterion (SC), $k = 1$

```
while SC not satisfied do
    choose a direction d_k
    decide a step size t_k
    make a step: x_k = x_{k-1} + t_k d_k
    update counter: k = k + 1
end while
```

Four questions:

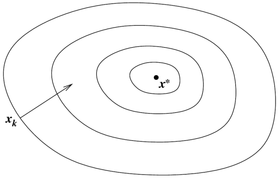
- How to choose direction $\boldsymbol{d}_k$?
- How to choose step size t_k?
- Where to initialize?
- When to stop?

How to choose a search direction?

We want to decrease the function value toward global minimum...
shortsighted answer: find a direction to decrease most rapidly

for any fixed $t > 0$, using 1st order Taylor expansion

$$f(\boldsymbol{x}_k + t\boldsymbol{v}) - f(\boldsymbol{x}_k) \approx t \langle \nabla f(\boldsymbol{x}_k), \boldsymbol{v} \rangle$$



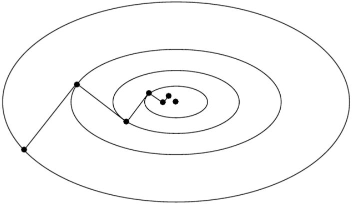
$$\min_{\|\boldsymbol{v}\|_2 = 1} \langle \nabla f(\boldsymbol{x}_k), \boldsymbol{v} \rangle \implies \boldsymbol{v} = -\frac{\nabla f(\boldsymbol{x}_k)}{\|\nabla f(\boldsymbol{x}_k)\|_2}$$

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

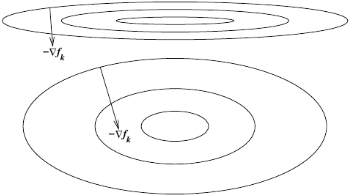
gradient/steepest descent: $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - t\nabla f(\boldsymbol{x}_k)$

$$\min_{\boldsymbol{x}} \; \boldsymbol{x}^{\intercal} \boldsymbol{A} \boldsymbol{x} + \boldsymbol{b}^{\intercal} \boldsymbol{x}$$

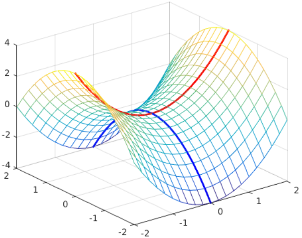
typical zig-zag path



conditioning affects the path length



$f(x,y) = x^2 - y^2$



- remember direction curvature?
  $\boldsymbol{v}^{\intercal} \nabla^2 f(\boldsymbol{x})\, \boldsymbol{v} = \frac{d^2}{dt^2} f(\boldsymbol{x} + t\boldsymbol{v}) \Big|_{t=0}$
- large curvature $\leftrightarrow$ narrow valley
- directional curvatures encoded in the Hessian

How to choose a search direction?

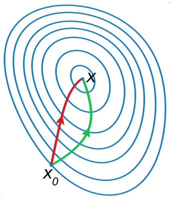
We want to decrease the function value toward global minimum...

shortsighted answer: find a direction to decrease most rapidly

farsighted answer: find a direction based on both gradient and Hessian

for any fixed $t > 0$, using 2nd-order Taylor expansion

$$f(\boldsymbol{x}_k + t\boldsymbol{v}) - f(\boldsymbol{x}_k) \approx t \langle \nabla f(\boldsymbol{x}_k), \boldsymbol{v} \rangle + \frac{1}{2} t^2 \langle \boldsymbol{v}, \nabla^2 f(\boldsymbol{x}_k) \boldsymbol{v} \rangle$$



grad desc: green; Newton: red

minimizing the right side

$$\Longrightarrow \boldsymbol{v} = -t^{-1} \left[\nabla^2 f(\boldsymbol{x}_k)\right]^{-1} \nabla f(\boldsymbol{x}_k)$$

Set $\boldsymbol{d}_{k+1} = -\left[\nabla^2 f(\boldsymbol{x}_k)\right]^{-1} \nabla f(\boldsymbol{x}_k)$

Newton's method: $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - t\left[\nabla^2 f(\boldsymbol{x}_k)\right]^{-1} \nabla f(\boldsymbol{x}_k)$,

t is often set to be 1.

Why called Newton's method?

Newton's method: $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - t\left[\nabla^2 f\left(\boldsymbol{x}_k\right)\right]^{-1} \nabla f\left(\boldsymbol{x}_k\right)$,

Recall Newton's method for root-finding: $f(x) = 0$

$$x_{k+1} = x_k - \frac{f\left(x_k\right)}{f'\left(x_k\right)}$$

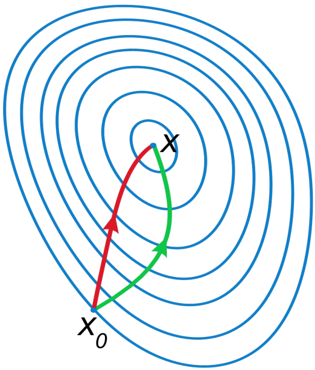
Newton's method for solving nonlinear system: $f\left(\boldsymbol{x}\right) = \mathbf{0}$

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \left[\boldsymbol{J}_f\left(\boldsymbol{x}_k\right)\right]^{\dagger} \boldsymbol{f}\left(\boldsymbol{x}_k\right)$$

Newton's method for solving $\nabla f\left(\boldsymbol{x}\right) = \mathbf{0}$

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \left[\nabla^2 f\left(\boldsymbol{x}_k\right)\right]^{-1} \nabla \boldsymbol{f}\left(\boldsymbol{x}_k\right)$$

How to choose a search direction?



grad desc: green; Newton: red

Newton's method take fewer steps

nearsighted choice: cost $O(n)$ per step

gradient/steepest descent:
$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - t\nabla f\left(\boldsymbol{x}_k\right)$

farsighted choice: cost $O(n^3)$ per step

Newton's method: $\boldsymbol{x}_{k+1} =$
$\boldsymbol{x}_k - t\left[\nabla^2 f\left(\boldsymbol{x}_k\right)\right]^{-1}\nabla f\left(\boldsymbol{x}_k\right),$

Implication: The plain Newton never used for large-scale problems. More on this later ...

Problems with Newton's method

Newton's method: $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - t \left[\nabla^2 f(\boldsymbol{x}_k)\right]^{-1} \nabla f(\boldsymbol{x}_k)$,

for any fixed $t > 0$, using 2nd-order Taylor expansion

$$f(\boldsymbol{x}_k + t\boldsymbol{v}) - f(\boldsymbol{x}_k) \approx t \langle \nabla f(\boldsymbol{x}_k), \boldsymbol{v} \rangle + \frac{1}{2} t^2 \langle \boldsymbol{v}, \nabla^2 f(\boldsymbol{x}_k) \boldsymbol{v} \rangle$$

minimizing the right side $\Longrightarrow \boldsymbol{v} = -t^{-1} \left[\nabla^2 f(\boldsymbol{x}_k)\right]^{-1} \nabla f(\boldsymbol{x}_k)$

- $\nabla^2 f(\boldsymbol{x}_k)$ may be non-invertible
- the minimum value is $-\frac{1}{2} \left\langle \nabla f(\boldsymbol{x}_k), \left[\nabla^2 f(\boldsymbol{x}_k)\right]^{-1} \nabla f(\boldsymbol{x}_k) \right\rangle$. If $\nabla^2 f(\boldsymbol{x}_k)$ not positive definite, may be positive

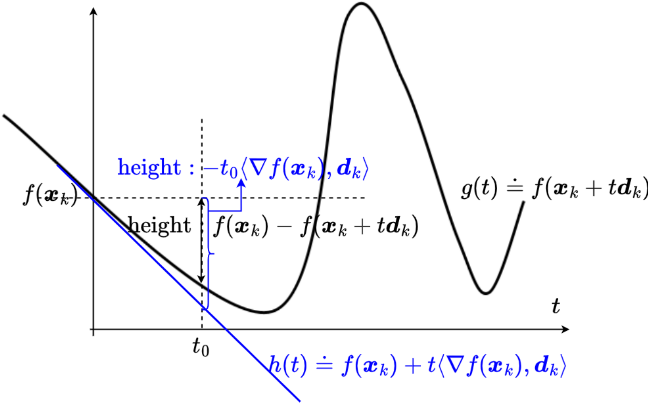
solution: e.g., modify the Hessian $\nabla^2 f(\boldsymbol{x}_k) + \tau \boldsymbol{I}$ with τ sufficiently large

How to choose step size?

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

- Naive choice: sufficiently small constant t for all k
- A robust and practical choice: **back-tracking line search**

Intuition for back-tracking line search:



Back-tracking line search

- $f(\boldsymbol{x}_k + t\boldsymbol{d}_k) = f(\boldsymbol{x}_k) + t\langle \nabla f(\boldsymbol{x}_k), \boldsymbol{d}_k\rangle + o\left(t\|\boldsymbol{d}_k\|_2\right)$ when t sufficiently small — $t\langle \nabla f(\boldsymbol{x}_k), \boldsymbol{d}_k\rangle$ dictates the value decrease
- But we also want t large as possible to make rapid progress
- **idea**: find a large possible t^* to make sure $f(\boldsymbol{x}_k + t^*\boldsymbol{d}_k) - f(\boldsymbol{x}_k) \le ct^*\langle \nabla f(\boldsymbol{x}_k), \boldsymbol{d}_k\rangle$ **(key condition)** for a chosen parameter $c \in (0,1)$, and no less
- **details**: start from $t = 1$. If the **key condition** not satisfied, $t = \rho t$ for a chosen parameter $\rho \in (0,1)$.

A widely implemented strategy in numerical optimization packages

---

Back-tracking line search

Input: initial $t > 0$, $\rho \in (0,1)$, $c \in (0,1)$

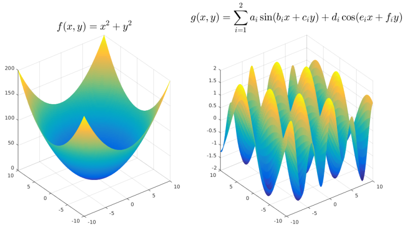
1: **while** $f(\boldsymbol{x}_k + t\boldsymbol{d}_k) - f(\boldsymbol{x}_k) \ge ct\langle \nabla f(\boldsymbol{x}_k), \boldsymbol{d}_k\rangle$ **do**

2: $\quad t = \rho t$

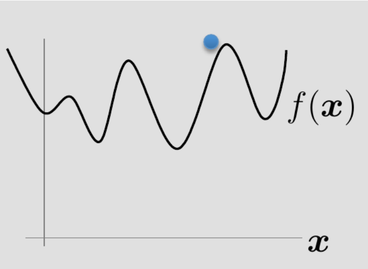
3: **end while**

Output: $t_k = t$.

Where to initialize?



convex vs. nonconvex functions



- **Convex**: most iterative methods converge to the global min no matter the initialization
- **Nonconvex**: initialization matters a lot. Common heuristics: random initialization, multiple independent runs
- **Nonconvex**: clever initialization is possible with certain assumptions on the data:

  https://sunju.org/research/nonconvex/

  and sometimes random initialization works!

When to stop?

1st-order necessary condition: Assume f is 1st-order differentiable at $\boldsymbol{x}_0$. If $\boldsymbol{x}_0$ is a local minimizer, then $\nabla f(\boldsymbol{x}_0) = \mathbf{0}$.
2nd-order necessary condition: Assume $f(\boldsymbol{x})$ is 2-order differentiable at $\boldsymbol{x}_0$. If $\boldsymbol{x}_0$ is a local min, $\nabla f(\boldsymbol{x}_0) = \mathbf{0}$ and $\nabla^2 f(\boldsymbol{x}_0) \succeq \mathbf{0}$.

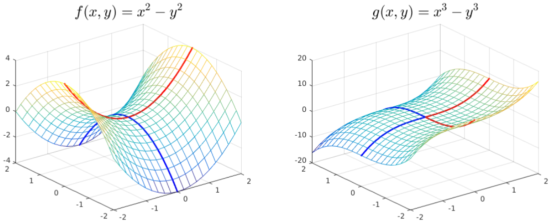
Fix some positive tolerance values ε_g, ε_H, ε_f, ε_v. Possibilities:

- $\|\nabla f(\boldsymbol{x}_k)\|_2 \le \varepsilon_g$, i.e., check 1st order cond
- $\|\nabla f(\boldsymbol{x}_k)\|_2 \le \varepsilon_g$ and $\lambda_{\min}\left(\nabla^2 f(\boldsymbol{x}_k)\right) \ge -\varepsilon_H$, i.e., check 2nd order cond
- $|f(\boldsymbol{x}_k) - f(\boldsymbol{x}_{k-1})| \le \varepsilon_f$
- $\|\boldsymbol{x}_k - \boldsymbol{x}_{k-1}\|_2 \le \varepsilon_v$

Nonconvex optimization is hard

Nonconvex: Even computing (verifying!) a local minimizer is NP-hard! (see, e.g., [Murty and Kabadi, 1987])

2nd order sufficient: $\nabla f(\boldsymbol{x}_0) = \mathbf{0}$ and $\nabla^2 f(\boldsymbol{x}_0) \succ \mathbf{0}$
2nd order necessary: $\nabla f(\boldsymbol{x}_0) = \mathbf{0}$ and $\nabla^2 f(\boldsymbol{x}_0) \succeq \mathbf{0}$



Cases in between: local shapes around SOSP determined by **spectral properties of higher-order derivative tensors**, calculating which is hard [Hillar and Lim, 2013]!

Outline

Classic line-search methods

Advanced line-search methods

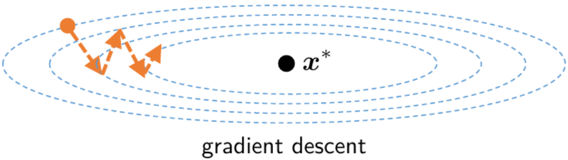
- Momentum methods
- Quasi-Newton methods
- Coordinate descent
- Conjugate gradient methods

Trust-region methods

A word on constrained problems

Outline

Why momentum?



Credit: Princeton ELE522

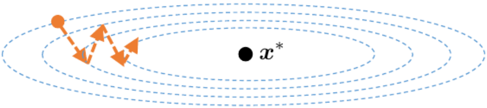
- GD is cheap ($O(n)$ per step) but overall convergence sensitive to conditioning
- Newton's convergence is not sensitive to conditioning but expensive ($O(n^3)$ per step)

A cheap way to achieve faster convergence? Answer: using historic information

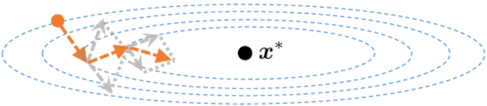
Heavy ball method

In physics, a heavy object has a large inertia/momentum — resistance to change of velocity.

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \alpha_k \nabla f\left(\boldsymbol{x}_k\right) + \beta_k \underbrace{\left(\boldsymbol{x}_k - \boldsymbol{x}_{k-1}\right)}_{\textbf{momentum}} \quad \text{due to Polyak}$$



gradient descent



heavy-ball method

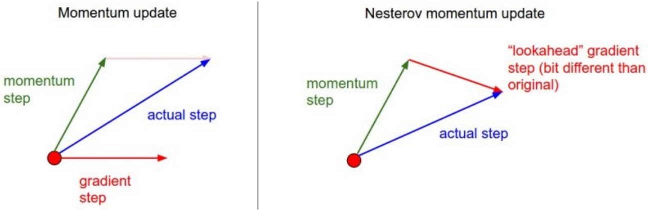
Credit: Princeton ELE522

History helps to smooth out the zig-zag path!

Nesterov's accelerated gradient methods

Another version, due to Y. Nesterov

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \beta_k \left(\boldsymbol{x}_k - \boldsymbol{x}_{k-1}\right) - \alpha_k \nabla f \left(\boldsymbol{x}_k + \beta_k \left(\boldsymbol{x}_k - \boldsymbol{x}_{k-1}\right)\right)$$



Credit: Stanford CS231N

$$\text{HB} \begin{cases} x_{\text{ahead}} = x + \beta(x - x_{\text{old}}), \\ x_{\text{new}} = x_{\text{ahead}} - \alpha \nabla f(x). \end{cases} \quad \text{Nesterov} \begin{cases} x_{\text{ahead}} = x + \beta(x - x_{\text{old}}), \\ x_{\text{new}} = x_{\text{ahead}} - \alpha \nabla f(x_{\text{ahead}}). \end{cases}$$

For more info, see Chap 10 of [Beck, 2017] and Chap 2 of [Nesterov, 2018].

Outline

Quasi-Newton methods

quasi-: seemingly; apparently but not really.

Newton's method: cost $O(n^2)$ storage and $O(n^3)$ computation per step

$$\boxed{\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - t\left[\nabla^2 f(\boldsymbol{x}_k)\right]^{-1} \nabla f(\boldsymbol{x}_k)}$$

Idea: approximate $\nabla^2 f(\boldsymbol{x}_k)$ or $\left[\nabla^2 f(\boldsymbol{x}_k)\right]^{-1}$ to allow efficient storage and computation — **Quasi-Newton Methods**

Choose $\boldsymbol{H}_k$ to approximate $\nabla^2 f(\boldsymbol{x}_k)$ so that

- avoid calculation of second derivatives
- simplify matrix inversion, i.e., computing the search direction

Quasi-Newton methods

given: starting point $x_0 \in \operatorname{dom} f$, $H_0 \succ 0$

for $k = 0, 1, \ldots$

1. compute quasi-Newton direction $\Delta x_k = -H_k^{-1} \nabla f(x_k)$
2. determine step size t_k (*e.g.*, by backtracking line search)
3. compute $x_{k+1} = x_k + t_k \Delta x_k$
4. compute H_{k+1}

– Different variants differ on how to compute $\boldsymbol{H}_{k+1}$

– Normally $\boldsymbol{H}_k^{-1}$ or its factorized version stored to simplify calculation of $\Delta \boldsymbol{x}_k$

Credit: UCLA ECE236C

BFGS method

Broyden–Fletcher–Goldfarb–Shanno (BFGS) method

BFGS update

$$H_{k+1} = H_k + \frac{yy^T}{y^Ts} - \frac{H_k ss^T H_k}{s^T H_k s}$$

where

$$s = x_{k+1} - x_k, \qquad y = \nabla f(x_{k+1}) - \nabla f(x_k)$$

Inverse update

$$H_{k+1}^{-1} = \left(I - \frac{sy^T}{y^Ts}\right) H_k^{-1} \left(I - \frac{ys^T}{y^Ts}\right) + \frac{ss^T}{y^Ts}$$

Cost of update: $O(n^2)$ (vs. $O(n^3)$ in Newton's method), storage: $O(n^2)$ To derive the update equations, three conditions are imposed:

- secant condition: $\boldsymbol{H}_{k+1}\boldsymbol{s} = \boldsymbol{y}$ (think of 1st Taylor expansion to ∇f)
- curvature condition: $\boldsymbol{s}_k^\mathsf{T}\boldsymbol{y}_k > 0$ to ensure that $\boldsymbol{H}_{k+1} \succ \boldsymbol{0}$ if $\boldsymbol{H}_k \succ \boldsymbol{0}$
- $\boldsymbol{H}_{k+1}$ and $\boldsymbol{H}_k$ are close in an appropriate sense

See Chap 6 of [Nocedal and Wright, 2006] Credit: UCLA ECE236C

Limited-memory BFGS (L-BFGS)

Limited-memory BFGS (L-BFGS): do not store H_k^{-1} explicitly

- instead we store up to m (*e.g.*, $m = 30$) values of

$$s_j = x_{j+1} - x_j, \qquad y_j = \nabla f(x_{j+1}) - \nabla f(x_j)$$

- we evaluate $\Delta x_k = H_k^{-1} \nabla f(x_k)$ recursively, using

$$H_{j+1}^{-1} = \left(I - \frac{s_j y_j^T}{y_j^T s_j}\right) H_j^{-1} \left(I - \frac{y_j s_j^T}{y_j^T s_j}\right) + \frac{s_j s_j^T}{y_j^T s_j}$$

for $j = k-1, \ldots, k-m$, assuming, for example, $H_{k-m} = I$

- an alternative is to restart after m iterations

Cost of update: $O(mn)$ (vs. $O(n^2)$ in BFGS), storage: $O(mn)$ (vs. $O(n^2)$ in BFGS) — linear in dimension n! recall the cost of GD?

See Chap 7 of [Nocedal and Wright, 2006] Credit: UCLA ECE236C

Outline

Block coordinate descent

Consider a function $f(\boldsymbol{x}_1, \dots, \boldsymbol{x}_p)$ with $\boldsymbol{x}_1 \in \mathbb{R}^{n_1}$, . . . , $\boldsymbol{x}_p \in \mathbb{R}^{n_p}$

A generic block coordinate descent algorithm

Input: initialization $(\boldsymbol{x}_{1,0}, \dots, \boldsymbol{x}_{p,0})$ (the 2nd subscript indexes iteration number)

1: **for** $k = 1, 2, \dots$ **do**

2: Pick a block index $i \in \{1, \dots, p\}$

3: Minimize wrt the chosen block:

$\boldsymbol{x}_{i,k} = \arg\min_{\boldsymbol{\xi} \in \mathbb{R}^{n_i}} \; f(\boldsymbol{x}_{1,k-1}, \dots, \boldsymbol{x}_{i-1,k-1}, \boldsymbol{\xi}, \boldsymbol{x}_{i+1,k-1}, \dots, \boldsymbol{x}_{p,k-1})$

4: Leave other blocks unchanged: $\boldsymbol{x}_{j,k} = \boldsymbol{x}_{j,k-1} \; \forall \, j \neq i$

5: **end for**

- Also called **alternating direction/minimization methods**
- When $n_1 = n_2 = \cdots = n_p = 1$, called **coordinate descent**
- Minimization in Line 3 can be inexact: e.g.,
  $\boldsymbol{x}_{i,k} = \boldsymbol{x}_{i,k-1} - t_k \frac{\partial f}{\partial \boldsymbol{\xi}}(\boldsymbol{x}_{1,k-1}, \dots, \boldsymbol{x}_{i-1,k-1}, \boldsymbol{x}_{i,k-1}, \boldsymbol{x}_{i+1,k-1}, \dots, \boldsymbol{x}_{p,k-1})$
- In Line 2, many different ways of picking an index, e.g., cyclic, randomized, weighted sampling, etc

Block coordinate descent: examples

Least-squares $\min_{\boldsymbol{x}}\ f(\boldsymbol{x}) = \|\boldsymbol{y} - \boldsymbol{A}\boldsymbol{x}\|_2^2$

- $\|\boldsymbol{y} - \boldsymbol{A}\boldsymbol{x}\|_2^2 = \|\boldsymbol{y} - \boldsymbol{A}_{-i}\boldsymbol{x}_{-i} - \boldsymbol{a}_i x_i\|^2$
- coordinate descent: $\min_{\xi \in \mathbb{R}}\ \|\boldsymbol{y} - \boldsymbol{A}_{-i}\boldsymbol{x}_{-i} - \boldsymbol{a}_i \xi\|^2$

$$\Longrightarrow x_{i,+} = \frac{\langle \boldsymbol{y} - \boldsymbol{A}_{-i}\boldsymbol{x}_{-i}, \boldsymbol{a}_i \rangle}{\|\boldsymbol{a}_i\|_2^2}$$

($\boldsymbol{A}_{-i}$ is $\boldsymbol{A}$ with the i-th column removed; $\boldsymbol{x}_{-i}$ is $\boldsymbol{x}$ with the i-th coordinate removed)

Matrix factorization $\min_{\boldsymbol{A},\boldsymbol{B}}\ \|\boldsymbol{Y} - \boldsymbol{A}\boldsymbol{B}\|_F^2$

- Two groups of variables, consider block coordinate descent
- Updates:

$$\boldsymbol{A}_+ = \boldsymbol{Y}\boldsymbol{B}^\dagger,$$
$$\boldsymbol{B}_+ = \boldsymbol{A}^\dagger\boldsymbol{Y}.$$

$(\cdot)^\dagger$ denotes the matrix pseudoinverse.

Why block coordinate descent?

- may work with constrained problems and non-differentiable problems (e.g., $\min_{\boldsymbol{A},\boldsymbol{B}} \ \|\boldsymbol{Y}-\boldsymbol{A}\boldsymbol{B}\|_F^2, \ \mathrm{s.t.}\ \boldsymbol{A}$ orthogonal, Lasso: $\min_{\boldsymbol{x}} \ \|\boldsymbol{y}-\boldsymbol{A}\boldsymbol{x}\|_2^2+\lambda\|\boldsymbol{x}\|_1$)
- may be faster than gradient descent or Newton (next)
- may be simple and cheap!

Some references:

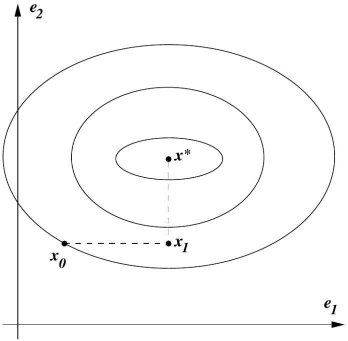
- [Wright, 2015]
- Lecture notes by Prof. Ruoyu Sun

Outline

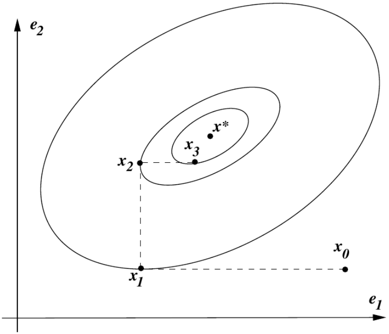
Conjugate direction methods

Solve linear equation $\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x} \Longleftrightarrow \min_{\boldsymbol{x}} \ \frac{1}{2}\boldsymbol{x}^\top \boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}^\top \boldsymbol{x}$ with $\boldsymbol{A} \succ \boldsymbol{0}$

apply coordinate descent...



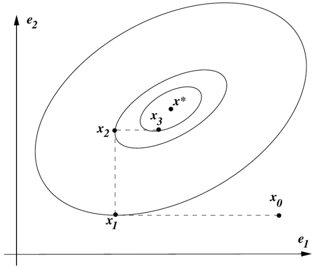
diagonal $\boldsymbol{A}$: solve the problem in n steps



non-diagonal $\boldsymbol{A}$: does not solve the problem in n steps

Conjugate direction methods

Solve linear equation $\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x} \Longleftrightarrow \min_{\boldsymbol{x}} \ \frac{1}{2}\boldsymbol{x}^\intercal \boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}^\intercal \boldsymbol{x}$ with $\boldsymbol{A} \succ \boldsymbol{0}$



non-diagonal $\boldsymbol{A}$: does not solve the problem in n steps

Idea: define n "conjugate directions" $\{\boldsymbol{p}_1, \dots, \boldsymbol{p}_n\}$ so that $\boldsymbol{p}_i^\intercal \boldsymbol{A}\boldsymbol{p}_j = 0$ for all $i \neq j$—conjugate as generalization of orthogonal

- Write $\boldsymbol{P} = [\boldsymbol{p}_1, \dots, \boldsymbol{p}_n]$. Can verify that $\boldsymbol{P}^\intercal \boldsymbol{A}\boldsymbol{P}$ is diagonal and positive
- Write $\boldsymbol{x} = \boldsymbol{P}\boldsymbol{s}$. Then $\frac{1}{2}\boldsymbol{x}^\intercal \boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}^\intercal \boldsymbol{x} = \frac{1}{2}\boldsymbol{s}^\intercal \left(\boldsymbol{P}^\intercal \boldsymbol{A}\boldsymbol{P}\right) \boldsymbol{s} - \left(\boldsymbol{P}^\intercal \boldsymbol{b}\right)^\intercal \boldsymbol{s}$ — quadratic with diagonal $\boldsymbol{P}^\intercal \boldsymbol{A}\boldsymbol{P}$
- Perform updates in the $\boldsymbol{s}$ space, but write the equivalent form in $\boldsymbol{x}$ space
- The i-the coordinate direction in the $\boldsymbol{s}$ space is $\boldsymbol{p}_i$ in the $\boldsymbol{x}$ space

In short, change of variable trick!

Conjugate gradient methods

Solve linear equation $\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x} \Longleftrightarrow \min_{\boldsymbol{x}} \ \frac{1}{2}\boldsymbol{x}^\intercal \boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}^\intercal \boldsymbol{x}$ with $\boldsymbol{A} \succ \boldsymbol{0}$
Idea: define n "conjugate directions" $\{\boldsymbol{p}_1, \ldots, \boldsymbol{p}_n\}$ so that $\boldsymbol{p}_i^\intercal \boldsymbol{A}\boldsymbol{p}_j = 0$ for all $i \neq j$—conjugate as generalization of orthogonal

Generally, many choices for $\{\boldsymbol{p}_1, \ldots, \boldsymbol{p}_n\}$.
Conjugate gradient methods: choice based on ideas from steepest descent

Algorithm 5.2 (CG).
Given x_0;
Set $r_0 \leftarrow Ax_0 - b$, $p_0 \leftarrow -r_0$, $k \leftarrow 0$;
while $r_k \neq 0$

$$\alpha_k \leftarrow \frac{r_k^T r_k}{p_k^T A p_k}; \tag{5.24a}$$

$$x_{k+1} \leftarrow x_k + \alpha_k p_k; \tag{5.24b}$$

$$r_{k+1} \leftarrow r_k + \alpha_k A p_k; \tag{5.24c}$$

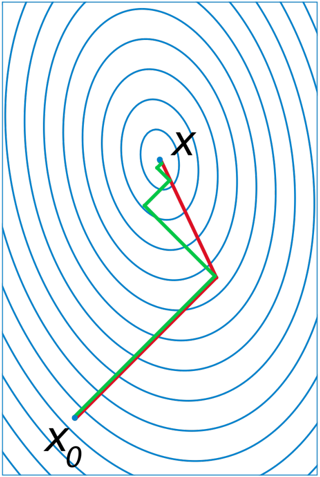
$$\beta_{k+1} \leftarrow \frac{r_{k+1}^T r_{k+1}}{r_k^T r_k}; \tag{5.24d}$$

$$p_{k+1} \leftarrow -r_{k+1} + \beta_{k+1} p_k; \tag{5.24e}$$

$$k \leftarrow k + 1; \tag{5.24f}$$

end (while)

Conjugate gradient methods



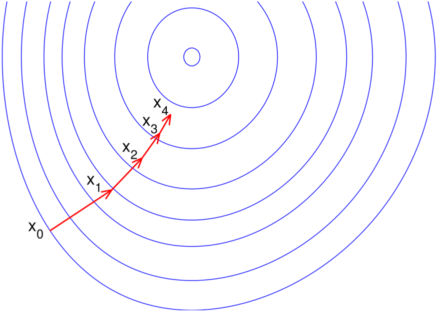
CG vs. GD (Green: GD, Red: CG)

- Can be extended to general non-quadratic functions
- Often used to solve subproblems of other iterative methods, e.g., truncated Newton method, the trust-region subproblem (later)

See Chap 5 of [Nocedal and Wright, 2006]

Outline

Iterative methods



Credit: aria42.com

Illustration of iterative methods on the contour/levelset plot (i.e., the function assumes the same value on each curve)

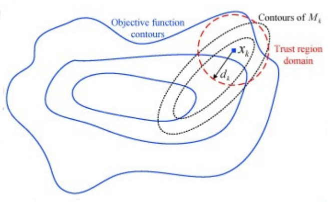
Two questions: what direction to move, and how far to move

Two possibilities:

- **Line-search methods**: direction first, size second
- **Trust-region methods** (TRM): size first, direction second

Ideas behind TRM

Recall Taylor expansion $f(\boldsymbol{x}+\boldsymbol{d}) \approx f(\boldsymbol{x}) + \langle \nabla f(\boldsymbol{x}_k), \boldsymbol{d}\rangle + \frac{1}{2}\langle \boldsymbol{d}, \nabla^2 f(\boldsymbol{x}_k)\boldsymbol{d}\rangle$



Credit: [Arezki et al., 2018]

Start with $\boldsymbol{x}_0$. Repeat the following:

- At $\boldsymbol{x}_k$, approximate f by the quadratic function (called model function dotted black in the left plot)

$$m_k(\boldsymbol{d}) = f(\boldsymbol{x}_k) + \langle \nabla f(\boldsymbol{x}_k), \boldsymbol{d}\rangle + \frac{1}{2}\langle \boldsymbol{d}, \boldsymbol{B}_k\boldsymbol{d}\rangle$$

  i.e., $m_k(\boldsymbol{d}) \approx f(\boldsymbol{x}_k + \boldsymbol{d})$, and $\boldsymbol{B}_k$ to approximate $\nabla^2 f(\boldsymbol{x}_k)$

- Minimize $m_k(\boldsymbol{d})$ within a **trust region** $\{\boldsymbol{d} : \|\boldsymbol{d}\| \le \Delta\}$, i.e., a norm ball (in red), to obtain $\boldsymbol{d}_k$
- If the approximation is inaccurate, decrease the region size; if the approximation is sufficiently accurate, increase the region size.
- If the approximation is reasonably accurate, update the iterate $\boldsymbol{x}_{k+1} = x_k + \boldsymbol{d}_k$.

Framework of trust-region methods

To measure approximation quality: $\rho_k \doteq \frac{f(\boldsymbol{x}_k) - f(\boldsymbol{x}_k + \boldsymbol{d}_k)}{m_k(\mathbf{0}) - m_k(\boldsymbol{d}_k)} = \frac{\text{actual decrease}}{\text{model decrease}}$

A generic trust-region algorithm

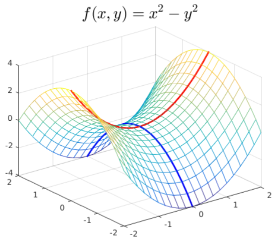
Input: $\boldsymbol{x}_0$, radius cap $\widehat{\Delta} > 0$, initial radius Δ_0, acceptance ratio $\eta \in [0, 1/4)$

for $k = 0, 1, \ldots$ **do**
    $\boldsymbol{d}_k = \arg\min_{\boldsymbol{d}} m_k(\boldsymbol{d})\,, \ \text{s.t.} \ \|\boldsymbol{d}\| \le \Delta_k$ **(TR Subproblem)**
    if $\rho_k < 1/4$ **then**
        $\Delta_{k+1} = \Delta_k / 4$
    else
        if $\rho_k > 3/4$ and $\|\boldsymbol{d}_k\| = \Delta_k$ **then**
            $\Delta_{k+1} = \min\left(2\Delta_k, \widehat{\Delta}\right)$
        else
            $\Delta_{k+1} = \Delta_k$
        end if
    end if
    if $\rho_k > \eta$ **then**
        $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \boldsymbol{d}_k$
    else
        $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k$
    end if
end for

Why TRM?

Recall the model function $m_k(\boldsymbol{d}) \doteq f(\boldsymbol{x}_k) + \langle \nabla f(\boldsymbol{x}_k), \boldsymbol{d} \rangle + \frac{1}{2} \langle \boldsymbol{d}, \boldsymbol{B}_k \boldsymbol{d} \rangle$

- Take $\boldsymbol{B}_k = \nabla^2 f(\boldsymbol{x}_k)$
- Gradient descent: stop at $\nabla f(\boldsymbol{x}_k) = \boldsymbol{0}$
- Newton's method: $\left[\nabla^2 f(\boldsymbol{x}_k)\right]^{-1} \nabla f(\boldsymbol{x}_k)$ may just stop at $\nabla f(\boldsymbol{x}_k) = \boldsymbol{0}$ or be ill-defined
- Trust-region method: $\min_{\boldsymbol{d}} \; m_k(\boldsymbol{d}) \quad \text{s.t.} \; \|\boldsymbol{d}\| \le \Delta_k$



When $\nabla f(\boldsymbol{x}_k) = \boldsymbol{0}$,

$$m_k(\boldsymbol{d}) - f(\boldsymbol{x}_k) = \frac{1}{2} \left\langle \boldsymbol{d}, \nabla^2 f(\boldsymbol{x}_k)\, \boldsymbol{d} \right\rangle .$$

If $\nabla^2 f(\boldsymbol{x}_k)$ has negative eigenvalues, i.e., there are negative directional curvatures, $\frac{1}{2} \left\langle \boldsymbol{d}, \nabla^2 f(\boldsymbol{x}_k)\, \boldsymbol{d} \right\rangle < 0$ for certain choices of $\boldsymbol{d}$ (e.g., eigenvectors corresponding to the negative eigenvalues)

TRM can help to move away from "nice" saddle points!

To learn more about TRM

- A comprehensive reference [Conn et al., 2000]
- A closely-related alternative: cubic regularized second-order (CRSOM) method [Nesterov and Polyak, 2006, Agarwal et al., 2018]
- Example implementation of both TRM and CRSOM: Manopt (in Matlab, Python, and Julia) https://www.manopt.org/ (choosing the Euclidean manifold)
- Computational complexity of numerical optimization methods [Cartis et al., 2022]

Outline

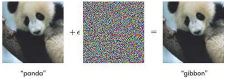
Why constrained optimization in deep learning?

General constrained optimization (nonlinear programming, NLP):

$$\min_{\boldsymbol{x}} \; f(\boldsymbol{x}) \quad \text{s.t.} \; \underbrace{g_i(\boldsymbol{x}) \le 0 \; \forall\, i \in \mathcal{I},}_{\text{inequality constraints}} \quad \underbrace{h_e(\boldsymbol{x}) = 0 \; \forall\, e \in \mathcal{E}}_{\text{equality constraints}}$$

$f(\boldsymbol{x})$, $g_i(\boldsymbol{x})$'s, and $h_e(\boldsymbol{x})$ can be nonconvex, and non-differentiable

Example I: Robustness of DL



$\boldsymbol{x}$ $\boldsymbol{\delta}$ $\boldsymbol{x}'$

$f_{\boldsymbol{\theta}}$: deep learning model

$$\max_{\boldsymbol{x}'} \; \ell(\boldsymbol{y}, f_{\boldsymbol{\theta}}(\boldsymbol{x}')) \Longleftarrow \text{Try to change the label}$$
$$\text{s.t.} \; d(\boldsymbol{x}, \boldsymbol{x}') \le \epsilon \Longleftarrow \text{Small perturbation}$$
$$\boldsymbol{x}' \in [0,1]^n \Longleftarrow \text{Still a valid image}$$

Example II: Physics-informed neural networks (PINNs)

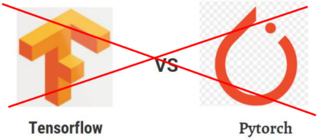
$$\underbrace{f\left(\mathbf{x}; \frac{\partial u}{\partial x_1}, \ldots, \frac{\partial u}{\partial x_d}; \frac{\partial^2 u}{\partial x_1 \partial x_1}, \ldots, \frac{\partial^2 u}{\partial x_1 \partial x_d}; \ldots; \boldsymbol{\lambda}\right) = 0, \quad \mathbf{x} \in \Omega,}_{\textbf{PDE}} \qquad \underbrace{\mathcal{B}(u, \mathbf{x}) = 0 \quad \text{on} \quad \partial\Omega}_{\textbf{Boundary Condition}}$$

$$\mathcal{L}(\boldsymbol{\theta}; \mathcal{T}) = w_f \overbrace{\mathcal{L}_f(\boldsymbol{\theta}; \mathcal{T}_f)}^{\text{objective}} + w_b \overbrace{\mathcal{L}_b(\boldsymbol{\theta}; \mathcal{T}_b)}^{\text{constraint}}$$

penalty parameters

$$\mathcal{L}_f(\boldsymbol{\theta}; \mathcal{T}_f) = \frac{1}{|\mathcal{T}_f|} \sum_{\mathbf{x} \in \mathcal{T}_f} \left\| f\left(\mathbf{x}; \frac{\partial \hat{u}}{\partial x_1}, \ldots, \frac{\partial \hat{u}}{\partial x_d}; \frac{\partial^2 \hat{u}}{\partial x_1 \partial x_1}, \ldots, \frac{\partial^2 \hat{u}}{\partial x_1 \partial x_d}; \ldots; \boldsymbol{\lambda}\right) \right\|_2^2$$
$$\mathcal{L}_b(\boldsymbol{\theta}; \mathcal{T}_b) = \frac{1}{|\mathcal{T}_b|} \sum_{\mathbf{x} \in \mathcal{T}_b} \|\mathcal{B}(\hat{u}, \mathbf{x})\|_2^2,$$

Solvers for constrained optimization



current Tensorflow and Pytorch only target uncontrained problems

Solvers	Nonconvex	Nonsmooth	Differentiable manifold constraints	General smooth constraint	Specific constrained ML problem
SDPT3, Gurobi, Cplex, TFOCS, CVX(PY), AMPL, YALMIP	✗	✓	✗	✗	✗
PyTorch, Tensorflow	✓	✓	✗	✗	✗
(Py)manopt, Geomstats, McTorch, Geoopt, GeoTorch	✓	✓	✓	✗	✗
KNITRO, IPOPT, GENO, ensmallen	✓	✓	✓	✓	✗
Scikit-learn, MLib, Weka	✓	✓	✗	✗	✓

NCVX PyGRANSO

- more info in https://ncvx.org/tutorials/_files/SDM23_Deep_Learning_with_Nontrivial_Constraints.pdf and https://arxiv.org/abs/2210.00973
- complete code examples for machine/deep learning applications https://ncvx.org/examples/index.html

Quick examples

$$\min_{\boldsymbol{w},b,\zeta} \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{i=1}^{n}\zeta_i$$

$$\text{s.t.} \quad y_i\left(\boldsymbol{w}^\intercal \boldsymbol{x}_i + b\right) \geq 1 - \zeta_i, \ \zeta_i \geq 0 \ \ \forall i = 1, ..., n$$

```
def comb_fn(X_struct):
    # obtain optimization variables
    w = X_struct.w
    b = X_struct.b
    zeta = X_struct.zeta
    # objective function
    f = 0.5*w.T@w + C*torch.sum(zeta)
    # inequality constraints
    ci = pygransoStruct()
    ci.c1 = 1 - zeta - y*(x@w+b)
    ci.c2 = -zeta
    # equality constraint
    ce = None
    return [f,ci,ce]
# specify optimization variables
var_in = {"w": [d,1], "b": [1,1], "zeta": [n,1]}
# pygranso main algorithm
soln = pygranso(var_in,comb_fn)
```

$$\begin{aligned} \max_{\boldsymbol{x}'} \quad & \ell(\boldsymbol{y}, f_{\boldsymbol{\theta}}(\boldsymbol{x}')) \\ \text{s.t.} \quad & d(\boldsymbol{x}, \boldsymbol{x}') \leq \epsilon \\ & \boldsymbol{x}' \in [0,1]^n \end{aligned}$$

```
def comb_fn(X_struct):
    # obtain optimization variables
    x_prime = X_struct.x_prime
    # objective function
    f = loss_func(y, f_theta(x_prime))
    # inequality constraints
    ci = pygransoStruct()
    ci.c1 = d(x,x_prime) - epsilon
    ci.c2 = -x_prime
    ci.c3 = x_prime-1
    # equality constraint
    ce = None
    return [f,ci,ce]
# specify optimization variable (tensor)
var_in = {"x_prime": list(x.shape)}
# pygranso main algorithm
soln = pygranso(var_in,comb_fn)
```

[Agarwal et al., 2018] Agarwal, N., Boumal, N., Bullins, B., and Cartis, C. (2018). **Adaptive regularization with cubics on manifolds.** *arXiv:1806.00065*.

[Arezki et al., 2018] Arezki, Y., Nouira, H., Anwer, N., and Mehdi-Souzani, C. (2018). **A novel hybrid trust region minimax fitting algorithm for accurate dimensional metrology of aspherical shapes.** *Measurement*, 127:134–140.

[Beck, 2017] Beck, A. (2017). **First-Order Methods in Optimization.** Society for Industrial and Applied Mathematics.

[Cartis et al., 2022] Cartis, C., Gould, N., and Toint, P. (2022). **Evaluation Complexity of Algorithms for Nonconvex Optimization: Theory, Computation and Perspectives.** SIAM.

[Conn et al., 2000] Conn, A. R., Gould, N. I. M., and Toint, P. L. (2000). **Trust Region Methods.** Society for Industrial and Applied Mathematics.

[Hillar and Lim, 2013] Hillar, C. J. and Lim, L.-H. (2013). **Most tensor problems are NP-hard.** *Journal of the ACM*, 60(6):1–39.

References ii

[Murty and Kabadi, 1987] Murty, K. G. and Kabadi, S. N. (1987). **Some NP-complete problems in quadratic and nonlinear programming.** *Mathematical Programming*, 39(2):117–129.

[Nesterov, 2018] Nesterov, Y. (2018). **Lectures on Convex Optimization.** Springer International Publishing.

[Nesterov and Polyak, 2006] Nesterov, Y. and Polyak, B. (2006). **Cubic regularization of newton method and its global performance.** *Mathematical Programming*, 108(1):177–205.

[Nocedal and Wright, 2006] Nocedal, J. and Wright, S. J. (2006). **Numerical Optimization.** Springer New York.

[Wright, 2015] Wright, S. J. (2015). **Coordinate descent algorithms.** *Mathematical Programming*, 151(1):3–34.