Basics of Numerical Optimization: Iterative Methods

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

October 12, 2020
Logistics

- Project grouping
  * Proposal due: Oct 23
  * Proposal template: https://nips.cc/Conferences/2020/PaperInformation/StyleFiles

- Colab purchase
Find global minimum

$$\min_x f(x)$$

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

**Smart search**

1st-order necessary condition: Assume $f$ is 1st-order differentiable at $x_0$. If $x_0$ is a local minimizer, then $\nabla f(x_0) = 0$.

$x$ with $\nabla f(x) = 0$: 1st-order stationary point (1OSP)

2nd-order necessary condition: Assume $f(x)$ is 2-order differentiable at $x_0$. If $x_0$ is a local min, $\nabla f(x_0) = 0$ and $\nabla^2 f(x_0) \succeq 0$.

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

x with \( \nabla f(x) = 0 \) and \( \nabla^2 f(x) \succeq 0 \): 2nd-order stationary point (2OSP)

- **Analytic method**: find 1OSP’s using gradient first, then study them using Hessian — for simple functions! e.g.,

  \[
  f(x) = \|y - Ax\|_2^2, \text{ or } f(x, y) = x^2y^2 - x^3y + y^2 - 1
  \]

- **Iterative methods**: find 1OSP’s/2OSP’s by making consecutive small movements

This lecture: **iterative methods**
Iterative methods

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**: size first, direction second
Outline

Classic line-search methods

Advanced line-search methods
  Momentum methods
  Quasi-Newton methods
  Coordinate descent
  Conjugate gradient methods

Trust-region methods
Framework of line-search methods

A generic line search algorithm

**Input:** initialization \(x_0\), stopping criterion (SC), \(k = 1\)

1: while SC not satisfied do
2: choose a direction \(d_k\)
3: decide a step size \(t_k\)
4: make a step: \(x_k = x_{k-1} + t_k d_k\)
5: update counter: \(k = k + 1\)
6: end while

Four questions:

- How to choose direction \(d_k\)?
- 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

$$f(x_k + td_{k+1}) - f(x_k) \approx t \langle \nabla f(x_k), d_{k+1} \rangle$$

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

Set $d_k = -\nabla f(x_k)$

**gradient/steepest descent**: $x_{k+1} = x_k - t \nabla f(x_k)$
Gradient descent

\[ \min_x \ x^T A x + b^T x \]

**typical zig-zag path**

**conditioning affects the path length**

- remember direction curvature?
  \[ \nu^T \nabla^2 f(x) \nu = \left. \frac{d^2}{dt^2} f(x + t\nu) \right|_{t=0} \]
- 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

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

minimizing the right side

$$\implies v = -t^{-1} \left[ \nabla^2 f(x_k) \right]^{-1} \nabla f(x_k)$$

Set $d_k = \left[ \nabla^2 f(x_k) \right]^{-1} \nabla f(x_k)$

**Newton’s method:** $x_{k+1} = x_k - t \left[ \nabla^2 f(x_k) \right]^{-1} \nabla f(x_k)$, $t$ can set to be 1.
Why called Newton’s method?

**Newton’s method:** \( x_{k+1} = x_k - t \left[ \nabla^2 f (x_k) \right]^{-1} \nabla f (x_k) \),

Recall Newton’s method for root-finding

\[
x_{k+1} = x_k - f' (x_n) f (x_n)
\]

Newton’s method for solving nonlinear system \( f (x) = 0 \)

\[
x_{k+1} = x_k - [J_f (x_n)]^\dagger f (x_n)
\]

Newton’s method for solving \( \nabla f (x) = 0 \)

\[
x_{k+1} = x_k - \left[ \nabla^2 f (x_n) \right]^{-1} \nabla f (x_n)
\]
How to choose a search direction?

- **Nearsighted choice:** cost $O(n)$ per step

  **Gradient/steepest descent:**
  \[ x_{k+1} = x_k - t \nabla f(x_k) \]

- **Farsighted choice:** cost $O(n^3)$ per step

  **Newton’s method:**
  \[ x_{k+1} = x_k - t \left[ \nabla^2 f(x_k) \right]^{-1} \nabla f(x_k), \]

**Implication:** The plain Newton never used for large-scale problems. More on this later ...
Problems with Newton’s method

Newton’s method: $x_{k+1} = x_k - t \left[ \nabla^2 f(x_k) \right]^{-1} \nabla f(x_k)$,

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

$$f(x_k + tv) - f(v) \approx t \left< \nabla f(x_k), v \right> + \frac{1}{2} t^2 \left< v, \nabla^2 f(x_k) v \right>$$

minimizing the right side $\implies v = -t^{-1} \left[ \nabla^2 f(x_k) \right]^{-1} \nabla f(x_k)$

- $\nabla^2 f(x_k)$ may be non-invertible
- the minimum value is $-\frac{1}{2} \left< \nabla f(x_k), \left[ \nabla^2 f(x_k) \right]^{-1} \nabla f(x_k) \right>$. If $\nabla^2 f(x_k)$ not positive definite, may be positive

**solution**: e.g., modify the Hessian $\nabla^2 f(x_k) + \tau I$ with $\tau$ sufficiently large
How to choose step size?

\[ x_k = x_{k-1} + t_k d_k \]

- Naive choice: sufficiently small constant \( t \) for all \( k \)
- Robust and practical choice: back-tracking line search

Intuition for back-tracking line search:

- By Taylor’s theorem,
  \[ f(x_k + td_k) = f(x_k) + t \langle \nabla f(x_k), d_k \rangle + o(t \|d_k\|^2) \] when \( t \) sufficiently small — \( t \langle \nabla f(x_k), d_k \rangle \) 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
  \[ f(x_k + t^* d_k) - f(x_k) \leq ct^* \langle \nabla f(x_k), d_k \rangle \] (**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 $f(x_k + td_k) - f(x_k) \geq ct \langle \nabla f(x_k), d_k \rangle$ do
2. $t = \rho t$
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 $x_0$. If $x_0$ is a local minimizer, then $\nabla f(x_0) = 0$.

2nd-order necessary condition: Assume $f(x)$ is 2-order differentiable at $x_0$. If $x_0$ is a local min, $\nabla f(x_0) = 0$ and $\nabla^2 f(x_0) \succeq 0$.

Fix some positive tolerance values $\varepsilon_g, \varepsilon_H, \varepsilon_f, \varepsilon_v$. Possibilities:

- $\|\nabla f(x_k)\|_2 \leq \varepsilon_g$, i.e., check 1st order cond
- $\|\nabla f(x_k)\|_2 \leq \varepsilon_g$ and $\lambda_{\min} (\nabla^2 f(x_k)) \geq -\varepsilon_H$, i.e., check 2nd order cond
- $|f(x_k) - f(x_{k-1})| \leq \varepsilon_f$
- $\|x_k - x_{k-1}\|_2 \leq \varepsilon_v$
Nonconvex optimization is hard

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

2nd order sufficient: \( \nabla f(x_0) = 0 \) and \( \nabla^2 f(x_0) \succ 0 \)
2nd order necessary: \( \nabla f(x_0) = 0 \) and \( \nabla^2 f(x_0) \succeq 0 \)

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
Outline

Classic line-search methods

Advanced line-search methods

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

Trust-region methods
Why momentum?

- 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.

\[ x_{k+1} = x_k - \alpha_k \nabla f(x_k) + \beta_k (x_k - x_{k-1}) \]

due to Polyak

Credit: Princeton ELE522

History helps to smooth out the zig-zag path!
Nesterov’s accelerated gradient methods

Another version, due to Y. Nesterov

\[ x_{k+1} = x_k + \beta_k (x_k - x_{k-1}) - \alpha_k \nabla f (x_k + \beta_k (x_k - x_{k-1})) \]

Credit: Stanford CS231N

For more info, see Chap 10 of [Beck, 2017] and Chap 2 of [Nesterov, 2018].
Classic line-search methods

Advanced line-search methods

Momentum methods

Quasi-Newton methods

Coordinate descent

Conjugate gradient methods

Trust-region methods
Quasi-Newton methods

*quasi-*: seemingly; apparently but not really.

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

\[
x_{k+1} = x_k - t \left[ \nabla^2 f(x_k) \right]^{-1} \nabla f(x_k)
\]

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

Choose $H_k$ to approximate $\nabla^2 f(x_k)$ so that

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

given: starting point $x_0 \in \text{dom } f$, $H_0 > 0$

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 $H_{k+1}$

- Normally $H_k^{-1}$ or its factorized version stored to simplify calculation of $\Delta x_k$

Credit: UCLA ECE236C
BFGS method

Broyden–Fletcher–Goldfarb–Shanno (BFGS) method

BFGS update

\[ H_{k+1} = H_k + \frac{yy^T}{y^Ts} - \frac{H_kss^T H_k}{s^TH_k s} \]

where

\[ s = x_{k+1} - x_k, \quad y = \nabla f(x_{k+1}) - \nabla f(x_k) \]

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: \( H_{k+1} s = y \) (think of 1st Taylor expansion to \( \nabla f \))
– Curvature condition: \( s_k^T y_k > 0 \) to ensure that \( H_{k+1} \succ 0 \) if \( H_k \succ 0 \)
– \( H_{k+1} \) and \( H_k \) 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

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

- 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

- Classic line-search methods
- Advanced line-search methods
  - Momentum methods
  - Quasi-Newton methods
  - Coordinate descent
  - Conjugate gradient methods
- Trust-region methods
Block coordinate descent

Consider a function $f(x_1, \ldots, x_p)$ with $x_1 \in \mathbb{R}^{n_1}, \ldots, x_p \in \mathbb{R}^{n_p}$

A generic block coordinate descent algorithm

**Input:** initialization $(x_{1,0}, \ldots, x_{p,0})$ (the 2nd subscript indexes iteration number)

1: for $k = 1, 2, \ldots$ do
2: Pick a block index $i \in \{1, \ldots, p\}$
3: Minimize wrt the chosen block:
   $$x_{i,k} = \arg \min_{\xi \in \mathbb{R}^{n_i}} f(x_{1,k-1}, \ldots, x_{i-1,k-1}, \xi, x_{i+1,k-1}, \ldots, x_{p,k-1})$$
4: Leave other blocks unchanged: $x_{j,k} = x_{j,k-1} \forall j \neq i$
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.,
  $$x_{i,k} = x_{i,k-1} - t_k \frac{\partial f}{\partial \xi} (x_{1,k-1}, \ldots, x_{i-1,k-1}, x_{i,k-1}, x_{i+1,k-1}, \ldots, x_{p,k-1})$$
- In Line 2, many different ways of picking an index, e.g., cyclic, randomized, weighted sampling, etc
Block coordinate descent: examples

Least-squares \( \min_{x} f(x) = \| y - Ax \|^2 \)

- \( \| y - Ax \|^2 = \| y - A_{-i}x_{-i} - a_i x_i \|^2 \)
- coordinate descent: \( \min_{\xi \in \mathbb{R}} \| y - A_{-i}x_{-i} - a_i \xi \|^2 \)

\[ \iff x_{i,+} = \frac{\langle y - A_{-i}x_{-i}, a_i \rangle}{\| a_i \|^2} \]

\((A_{-i} \) is \( A \) with the \( i \)-th column removed; \( x_{-i} \) is \( x \) with the \( i \)-th coordinate removed\)

Matrix factorization \( \min_{A,B} \| Y - AB \|^2_F \)

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

\[ A_+ = Y B^\dagger, \]
\[ B_+ = A^\dagger Y. \]

\((\cdot)^\dagger \) denotes the matrix pseudoinverse.\)
Why block coordinate descent?

– may work with constrained problems and non-differentiable problems (e.g., $\min_{A,B} \|Y - AB\|_F^2$, s.t. $A$ orthogonal, Lasso: $\min_x \|y - Ax\|_2^2 + \lambda \|x\|_1$)

– 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

Classic line-search methods

Advanced line-search methods

Momentum methods
Quasi-Newton methods
Coordinate descent

Conjugate gradient methods

Trust-region methods
Conjugate direction methods

Solve linear equation \( y = Ax \iff \min_x \frac{1}{2} x^T Ax - b^T x \) with \( A \succeq 0 \)

apply coordinate descent...

diagonal \( A \): solve the problem in \( n \) steps

non-diagonal \( A \): does not solve the problem in \( n \) steps
Conjugate direction methods

Solve linear equation \( y = Ax \iff \min_x \frac{1}{2} x^T Ax - b^T x \) with \( A \succ 0 \)

**Idea:** define \( n \) “conjugate directions” \( \{p_1, \ldots, p_n\} \) so that \( p_i^T A p_j = 0 \) for all \( i \neq j \)—conjugate as generalization of orthogonal

- Write \( P = [p_1, \ldots, p_n] \). Can verify that \( P^T A P \) is diagonal and positive
- Write \( x = Ps \). Then \( \frac{1}{2} x^T Ax - b^T x = \frac{1}{2} s^T (P^T A P) s - (P^T b)^T s \) — quadratic with diagonal \( P^T A P \)
- Perform updates in the \( s \) space, but write the equivalent form in \( x \) space
- The \( i \)-the coordinate direction in the \( s \) space is \( p_i \) in the \( x \) space

In short, change of variable trick!
Conjugate gradient methods

Solve linear equation $y = Ax \iff \min x \frac{1}{2} x^T Ax - b^T x$ with $A \succ 0$

**Idea:** define $n$ “conjugate directions” $\{p_1, \ldots, p_n\}$ so that $p_i^T A p_j = 0$ for all $i \neq j$—conjugate as generalization of orthogonal

Generally, many choices for $\{p_1, \ldots, p_n\}$.

**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 $r_k \neq 0$

\[
\begin{align*}
\alpha_k &\leftarrow \frac{r_k^T r_k}{p_k^T A p_k} \tag{5.24a} \\
x_{k+1} &\leftarrow x_k + \alpha_k p_k \tag{5.24b} \\
r_{k+1} &\leftarrow r_k + \alpha_k A p_k \tag{5.24c} \\
\beta_{k+1} &\leftarrow \frac{r_{k+1}^T r_{k+1}}{r_k^T r_k} \tag{5.24d} \\
p_{k+1} &\leftarrow -r_{k+1} + \beta_{k+1} p_k \tag{5.24e} \\
k &\leftarrow k + 1 \tag{5.24f}
\end{align*}
\]

end (while)
Conjugate gradient methods

- 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

Classic line-search methods

Advanced line-search methods
  Momentum methods
  Quasi-Newton methods
  Coordinate descent
  Conjugate gradient methods

Trust-region methods
Iterative methods

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 \( f(x + d) \approx f(x) + \langle \nabla f(x_k), d \rangle + \frac{1}{2} \langle d, \nabla^2 f(x_k) d \rangle \)

Start with \( x_0 \). Repeat the following:

- At \( x_k \), approximate \( f \) by the quadratic function (called model function dotted black in the left plot)
  \[
  m_k(d) = f(x_k) + \langle \nabla f(x_k), d \rangle + \frac{1}{2} \langle d, B_k d \rangle
  \]
  i.e., \( m_k(d) \approx f(x_k + d) \), and \( B_k \) to approximate \( \nabla^2 f(x_k) \)

- Minimize \( m_k(d) \) within a trust region \( \{d : \|d\| \leq \Delta\} \), i.e., a norm ball (in red), to obtain \( d_k \)

- 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 \( x_{k+1} = x_k + d_k \).
Framework of trust-region methods

To measure approximation quality: \( \rho_k = \frac{f(x_k) - f(x_k + d_k)}{m_k(0) - m_k(d_k)} = \frac{\text{actual decrease}}{\text{model decrease}} \)

A generic trust-region algorithm

Input: \( x_0 \), radius cap \( \hat{\Delta} > 0 \), initial radius \( \Delta_0 \), acceptance ratio \( \eta \in [0, 1/4) \)

1: for \( k = 0, 1, \ldots \) do
2: \( d_k = \arg \min_d m_k(d) \), s.t. \( \|d\| \leq \Delta_k \) (TR Subproblem)
3: if \( \rho_k < 1/4 \) then
4: \( \Delta_{k+1} = \Delta_k / 4 \)
5: else
6: if \( \rho_k > 3/4 \) and \( \|d_k\| = \Delta_k \) then
7: \( \Delta_{k+1} = \min(2\Delta_k, \hat{\Delta}) \)
8: else
9: \( \Delta_{k+1} = \Delta_k \)
10: end if
11: end if
12: if \( \rho_k > \eta \) then
13: \( x_{k+1} = x_k + d_k \)
14: else
15: \( x_{k+1} = x_k \)
16: end if
17: end for
Recall the model function \( m_k(d) = f(x_k) + \langle \nabla f(x_k), d \rangle + \frac{1}{2} \langle d, B_k d \rangle \)

- Take \( B_k = \nabla^2 f(x_k) \)
- Gradient descent: stop at \( \nabla f(x_k) = 0 \)
- Newton’s method: \( \left[ \nabla^2 f(x_k) \right]^{-1} \nabla f(x_k) \) may just stop at \( \nabla f(x_k) = 0 \) or be ill-defined
- Trust-region method: \( \min_d m_k(d) \quad \text{s.t.} \quad \|d\| \leq \Delta_k \)

When \( \nabla f(x_k) = 0 \),

\[
m_k(d) - f(x_k) = \frac{1}{2} \langle d, \nabla^2 f(x_k) d \rangle.
\]

If \( \nabla^2 f(x_k) \) has negative eigenvalues, i.e., there are negative directional curvatures, \( \frac{1}{2} \langle d, \nabla^2 f(x_k) d \rangle < 0 \) for certain choices of \( d \) (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) https://www.manopt.org/ (choosing the Euclidean manifold)


177–205.


tion. Springer New York.
