

Unconstrained Optimization Algorithms

Ben Gravell

benjamin.gravell@utdallas.edu

The Erik Jonsson School of Engineering and Computer Science

The University of Texas at Dallas

800 W. Campbell Rd.

Richardson, TX 75080


My goal is to give you practical tools for your research.

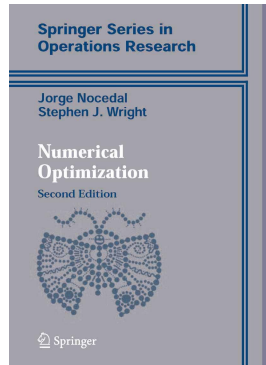


- 1 Unconstrained optimization preliminaries
- 2 Unconstrained optimization algorithms
- 3 Demo on LQR policy optimization

I will cover material from

- Nocedal and Wright's textbook "Numerical Optimization" [1]
- Assorted papers

It's going to be math-heavy 



How to use these slides

- Skim them quickly first
- Read the interesting sections in detail later
- Use them as a guide to chapters 1-4, 6-8, 10 of [1]
- [Contact me](#) if you find any mistakes!

Formatting

- Bold green words are **vocabulary**
- Bold orange words are **key thoughts**
- Bold black words are **just for emphasis**

- 1 Optimization preliminaries
 - Proof of FOSC
- 2 Slope conditions
 - Proof of convergence of gradient descent under RSI + RLG
- 3 Breezy tour of algorithms
- 4 LQR policy optimization
- 5 Demos

Unconstrained Optimization Preliminaries

Unconstrained Optimization Problem (hard version)

Find a point x^* that solves

$$x^* = \underset{x}{\operatorname{argmin}} f(x) \quad (1)$$

where $f : \mathbb{R}^n \rightarrow \mathbb{R}$ is called the **objective function**.

If such a solution x^* exists, it is called a **global minimizer** and satisfies

$$f(x^*) \leq f(x) \text{ for all } x \in \mathbb{R}^n$$

For some f the problem (1) is **ill-posed or too difficult**

- Settle for a solution to an easier problem
- **(Optional)** Impose mild restrictions on f so the easy and hard problem solutions coincide

Unconstrained Optimization Problem (easy version)

Assume f is **smooth** and has a **local minimizer** x^* . Find x^* .

Question: What do we mean by “**smooth**” and “**local minimizer**”?

Answer: Depends on the algorithm we are using so I phrased it vaguely.

First some definitions & preliminaries

“Smooth”

- First derivative exists
- First derivative is continuous
- Second derivative exists
- Second derivative is continuous

First derivative is the **gradient** vector

$$\nabla f(x) = \left[\frac{\partial f}{\partial x_i} \right]_{i=1, \dots, n} \in \mathbb{R}^n$$

Second derivative is the **Hessian** matrix

$$\nabla^2 f(x) = \left[\frac{\partial^2 f}{\partial x_i \partial x_j} \right]_{\substack{i=1, \dots, n \\ j=1, \dots, n}} \in \mathbb{R}^{n \times n}$$

A **neighborhood** \mathcal{N} of a point x^* is an open ball containing x^* ,
i.e. for some $\varepsilon > 0$

$$\mathcal{N} = \{x : \|x - x^*\| < \varepsilon\}$$

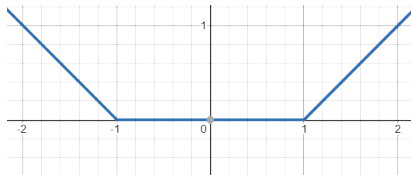
We say a point x is **(ε -)close** to x^* if $x \in \mathcal{N}$

Weak local minimizer

The point x^* is a **weak local minimizer** of f on \mathcal{N} if there exists a neighborhood \mathcal{N} of x^* for which

$$f(x^*) \leq f(x) \text{ for all } x \in \mathcal{N}.$$

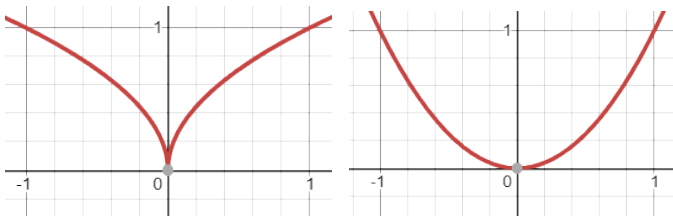
Mind picture: A lump of clay pressed on a table, a car tire on pavement



Isolated local minimizer

The point x^* is an **isolated local minimizer** of f on \mathcal{N} if there exists a neighborhood \mathcal{N} of x^* for which x is the only weak local minimizer in \mathcal{N} .

Mind picture: A knife dropped on its point on a table, a metal ball resting on flat ground

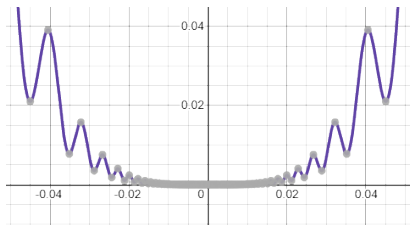


Strict local minimizer

The point x^* is a **strict local minimizer** of f on \mathcal{N} if there exists a neighborhood \mathcal{N} of x^* for which

$$f(x^*) < f(x) \text{ for all } x \in \mathcal{N} \text{ such that } x \neq x^*.$$

Mind picture: Like an isolated local minimizer, but allows ripples piling up near the strict minimizer



“Local minimizer”

- Usually we mean a weak local minimizer
- If f has a unique weak local minimizer x^* , then x^* is also isolated.
- Isolated local min \Rightarrow Strict local min \Rightarrow Weak local min

Question: How do we know if we have found a minimizer?

Answer: Check optimality conditions.

First-order necessary condition (FONC) (Theorem 2.2 of [1])

If f has a local minimizer x^* and f is continuously differentiable in a neighborhood of x^* , then

$$\nabla f(x^*) = 0.$$

Such a point is called a **stationary point**.

Proof: See [1]

Most optimization algorithms **search for a stationary point**, which are the only points which might be local minimizers by the FONC.¹

¹In the unconstrained setting, the KKT conditions reduce literally to the FONC.

The stationarity condition (FONC) can also be expressed as the finite-dimensional **variational inequality (VI)**²

$$\nabla f(x^*)^\top (x - x^*) \geq 0 \text{ for all } x \text{ close to } x^*$$

Note: Can only be true if $\nabla f(x^*) = 0$, so $\text{VI} \Leftrightarrow \text{FONC}$

The **global variational inequality (G-VI)**

$$\nabla f(x^*)^\top (x - x^*) \geq 0 \text{ for all } x$$

is necessary for x^* to be a global minimizer

²In the constrained setting, “for all x ” is replaced by “for all feasible x ”
- see eq. (4.21) in Chapter 4.2.3 of [2] for the convex setting

Alert! You will not find these powerful elementary results in standard textbooks or lecture notes!³

Math. Nachr. **144** (1989) 309–319

A First Order Sufficient Condition for Optimality in Nonsmooth Optimization

By RAFAËL CORREA and JEAN-BAPTISTE HIRIART-URRUTY

To take an example, suppose that f is differentiable in a neighborhood of a . Then, a *sufficient condition* for a to be a local minimum of f is that $\langle f'(x), a - x \rangle \leq 0$ for all x in a neighborhood of a . This very simple sufficient condition for minimality would deserve to be better known in the context of Differentiable Optimization.

...I agree!

³At least I did not come across them...please [contact me](#) if you have more sources!

First-order sufficient condition (FOSC) [3, 4]

Suppose f satisfies the conditions

- 1 f continuous on a neighborhood \mathcal{N} of x^*
- 2 f differentiable on $\mathcal{N} \setminus \{x^*\}$
- 3 $\nabla f(x)^\top (x - x^*) \geq 0$ for all $x \in \mathcal{N} \setminus \{x^*\}$

Then x^* is a weak local minimizer of f on \mathcal{N} .

Proof: [▶ Appendix](#)

Corollary (Global FOSC)

Suppose f satisfies the FOSC with $\mathcal{N} = \mathbb{R}^n$.
Then x^* is a global minimizer of f .

Proof: Follows from the FOSC.

Here are a couple trivial corollaries.

Corollary (Differentiable FOSC)

Suppose f satisfies the FOSC and f is continuously differentiable everywhere in \mathcal{N} (in particular $\nabla f(x^*)$ exists).

Then x^* is a stationary point and a weak local minimizer of f on \mathcal{N} .

Proof: Follows from the FOSC and FONC.

Corollary (Global differentiable FOSC (GD-FOSC))

Suppose f satisfies the differentiable FOSC with $\mathcal{N} = \mathbb{R}^n$.

Then x^* is a stationary point and a global minimizer of f .

Proof: Follows from the differentiable FOSC and global FOSC.

Here is the **strict** version of the FOSC.

Strict first-order sufficient condition (SFOSC) [3, 4]

Suppose f satisfies the conditions

- 1 f continuous on a neighborhood \mathcal{N} of x^*
- 2 f differentiable on $\mathcal{N} \setminus \{x^*\}$
- 3 $\nabla f(x)^\top (x - x^*) > 0$ for all $x \in \mathcal{N} \setminus \{x^*\}$

Then x^* is an isolated local minimizer of f on \mathcal{N} ,
and x^* is the only possible stationary point in \mathcal{N} .

Proof: [▶ Appendix](#)

Corollary (Global SFOSC)

Suppose f satisfies the SFOSC with $\mathcal{N} = \mathbb{R}^n$.
Then x^* is the unique global minimizer of f ,
and x^* is the only possible stationary point.

Proof: Follows from the SFOSC.

A couple more trivial corollaries:

Corollary (Differentiable SFOSC)

Suppose f satisfies the SFOSC and f is continuously differentiable everywhere in \mathcal{N} (in particular $\nabla f(x^*)$ exists).

Then x^* is a stationary point and an isolated local minimizer of f on \mathcal{N} .

Proof: Follows from the SFOSC and FONC.

Corollary (Global differentiable SFOSC)

Suppose f satisfies the differentiable SFOSC with $\mathcal{N} = \mathbb{R}^n$.

Then x^* is the unique stationary point and unique global minimizer of f .

Proof: Follows from the differentiable SFOSC and global SFOSC.

GD-SFOSC can be stated succinctly:⁴

Global differentiable SFOSC (simplified)

Suppose f is continuously differentiable and there exists x^* such that

$$\nabla f(x)^\top (x - x^*) \geq 0 \text{ for all } x \text{ with equality only when } x = x^*.$$

Then x^* is the unique stationary point and unique global minimizer of f .

Note: The GD-SFOSC is **not necessary** (except in 1D) for f to have a unique stationary point and unique global minimizer

- **Counterexample:** Rosenbrock function

⁴In the past I called the GD-SFOSC “restricted quasiconvexity”

Notice the **similarity and subtle (important!) difference** between

G-VI

$$\nabla f(x^*)^\top (x - x^*) \geq 0 \text{ for all } x$$

“Gradient at x^* points uphill
towards any x ”

GD-FOSC

$$\nabla f(x)^\top (x - x^*) \geq 0 \text{ for all } x$$

“Negative gradient at any x
points downhill towards x^* ”

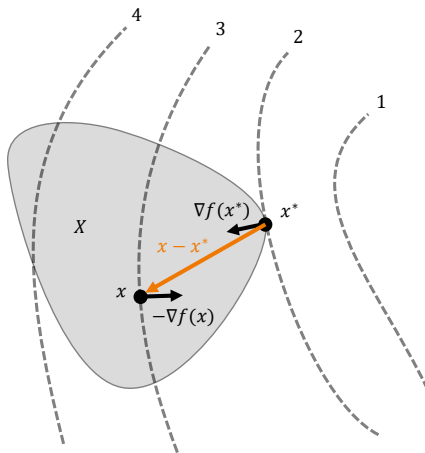


Figure 1: First order optimality conditions in the constrained setting. The point x^* is optimal and satisfies both the G-VI and GD-FOSC. The feasible set X is shaded. Level sets of f are dashed lines. Adapted from Figure 4.2 in [2].

Second-order necessary condition (SONC) (Theorem 2.3 of [1])

If f has a local minimizer x^* and $\nabla^2 f$ exists and is continuous in a neighborhood of x^* , then

$$\nabla f(x^*) = 0 \quad \text{and} \quad \nabla^2 f(x^*) \succeq 0.$$

Proof: See [1]

Second-order sufficient condition (SOSC) (Theorem 2.4 of [1])

Suppose that $\nabla^2 f$ is continuous in a neighborhood of a point x^* , and

$$\nabla f(x^*) = 0 \quad \text{and} \quad \nabla^2 f(x^*) \succ 0.$$

then x^* is a strict local minimizer of f .

Proof: See [1]

Note: **SOSC implies SFOSC [3].**

We want slope conditions on f to certify that
all stationary points are global minimizers.

Most slope conditions have a strict version which certifies that
a unique stationary point is the unique global minimizer.

We already saw a special case - the GD-SFOOSC

$$\nabla f(x)^\top (x - x^*) \geq 0 \text{ for all } x \text{ with equality only when } x = x^*.$$

implies **the stationary point x^* is the unique global minimizer.**

The most well-known and well-studied slope condition is **convexity**.

Definition of convexity (eq. (3.1) in [2])

The function $f : \mathbb{R}^n \rightarrow \mathbb{R}$ is **convex** if, for any $x, y \in \mathbb{R}^n$ and $\alpha \in [0, 1]$,

$$f(\alpha x + (1 - \alpha)y) \leq \alpha f(x) + (1 - \alpha)f(y)$$

First-order characterization of convexity (eq. (3.2) in [2])

Suppose the function $f : \mathbb{R}^n \rightarrow \mathbb{R}$ is differentiable. Then f is **convex** if, for any $x, y \in \mathbb{R}^n$,

$$\nabla f(x)^\top (x - y) \geq f(x) - f(y).$$

Second-order characterization of convexity (Chapter 3.1.4 in [2])

Suppose the function $f : \mathbb{R}^n \rightarrow \mathbb{R}$ is twice-differentiable. Then f is **convex** if, for any $x \in \mathbb{R}^n$,

$$\nabla^2 f(x) \succeq 0.$$

First-order sufficient conditions under convexity (Theorem 2.5 of [1])

(i) Suppose f is convex.

Then **all local minimizers are global minimizers.**

(ii) Suppose f is strictly convex.

Then **f has at most a single global minimizer.**

(iii) Suppose f is convex and differentiable.

Then **all stationary points are global minimizers.**

(iv) Suppose f is strictly convex, differentiable, and $\nabla f(x^*) = 0$.

Then **x^* is the unique stationary point and unique global minimizer.**

Convexity is nice, but the class of functions for which **all stationary points are global minimizers** is much more broad.

In fact, the **broadest class** of such functions are those which are **invex**.

Definition of invexity (eq. (1) in [5])

The differentiable function $f : \mathbb{R}^n \rightarrow \mathbb{R}$ is **invex** if there exists a function $\eta : \mathbb{R}^n \times \mathbb{R}^n \rightarrow \mathbb{R}^n$ such that, for any $x, y \in \mathbb{R}^n$,

$$\nabla f(x)^\top \eta(x, y) \geq f(x) - f(y).$$

Convexity implies invexity by taking $\eta(x, y) = x - y$.

Characterization of invexity (Theorem 1 in [5])

The differentiable function $f : \mathbb{R}^n \rightarrow \mathbb{R}$ is **invex** if and only if **all stationary points are global minimizers**.

There are **A LOT** of slope conditions enmeshed in a complex nesting of implication relationships.

For reference, here are all the slope conditions I know about.

Assume⁵

- f is continuously differentiable
- f attains a minimum value for a unique global minimizer x^*

Start off with the easy **quadratics**:

Quadratic $f(x) = x^T A x - b^T x + c$

Convex quadratic $f(x) = x^T A x - b^T x + c, A \succeq 0$

Strictly \Leftrightarrow strongly quadratic $f(x) = x^T A x - b^T x + c, A \succ 0$

⁵Many of these slope conditions hold under weaker assumptions e.g. multiple global minimizers, differentiable but not continuously so, etc. For consistency and comparative purposes we make the stated assumptions.

Weak conditions

Convex $\nabla f(x)^\top(x - y) \geq f(x) - f(y)$
for any $x, y \in \mathbb{R}^n$

Pseudoconvex (PC) $f(x) - f(y) \leq 0$,
for any $x, y \in \mathbb{R}^n$ such that $\nabla f(x)^\top(x - y) \leq 0$

Quasiconvex (QC) $\nabla f(x)^\top(x - y) \geq 0$,
for any $x, y \in \mathbb{R}^n$ such that $f(x) - f(y) \geq 0$

Star convex (★C) $\nabla f(x)^\top(x - x^*) \geq f(x) - f(x^*)$,
for any $x \in \mathbb{R}^n$

Quasar convex (QUC) $\nabla f(x)^\top(x - x^*) \geq \gamma[f(x) - f(x^*)]$,
 $\gamma > 0$, for any $x \in \mathbb{R}^n$

GD-FOSC $\nabla f(x)^\top(x - x^*) \geq 0$,
for any $x \in \mathbb{R}^n$

Invx $\nabla f(x)^\top \eta(x, y) \geq f(x) - f(y)$
for some function $\eta(\cdot, \cdot)$, for any $x, y \in \mathbb{R}^n$

Aliases

Weakly quasiconvex [6] \Leftrightarrow Quasar convex [7]

Strong conditions⁶

Strongly convex (SC) $\nabla f(x)^\top(x - y) \geq f(x) - f(y) + \mu\|x - y\|^2$,
 $\mu > 0$, for any $x, y \in \mathbb{R}^n$

Essentially strongly convex (ESC) $\nabla f(x)^\top(x - y) \geq f(x) - f(y) + \mu\|x - y\|^2$,
 $\mu > 0$, for any $x, y \in \mathbb{R}^n$ s.t. $\mathcal{P}(x) = \mathcal{P}(y)$ ⁷

Weakly strongly convex (WSC) $\nabla f(x)^\top(x - x^*) \geq f(x) - f(x^*) + \mu\|x - x^*\|^2$,
 $\mu > 0$, for any $x \in \mathbb{R}^n$

Strongly quasax convex (SQUC) $\nabla f(x)^\top(x - x^*) \geq \gamma[f(x) - f(x^*)] + \mu\|x - x^*\|^2$,
 $\gamma > 0$, $\mu > 0$, for any $x \in \mathbb{R}^n$

Restricted secant inequality (RSI) $\nabla f(x)^\top(x - x^*) \geq \mu\|x - x^*\|^2$,
 $\mu > 0$, for any $x \in \mathbb{R}^n$

Polyak-Łojasiewicz inequality (PL) $\|\nabla f(x)\|^2 \geq \mu(f(x) - f(x^*))$,
 $\mu > 0$, for any $x \in \mathbb{R}^n$

Error bound (EB) $\|\nabla f(x)\| \geq \mu\|x - x^*\|$,
 $\mu > 0$, for any $x \in \mathbb{R}^n$

Quadratic growth (QG) $f(x) - f(x^*) \geq \mu\|x - x^*\|^2$,
 $\mu > 0$, for any $x \in \mathbb{R}^n$

⁶Called “strong” because they force the slope to change quickly, and superior rates of convergence can be proved when they hold.

⁷Denote $\mathcal{P}(x)$ as the projection of x onto the solution set $\{x \mid f(x) = f(x^*)\}$. When x^* unique, $\mathcal{P}(x) = \mathcal{P}(y)$ ineffective so $\text{SC} \Leftrightarrow \text{ESC}$, else ESC is more general.

Note on terminology:

- “star”
- “quasar”
- “restricted”
- “radial”

suggests a slope condition holds with respect to only optimal points x^*

- Includes GD-FOSC and all stronger conditions

Contrast with e.g. (quasi)convexity which holds for any arbitrary point y

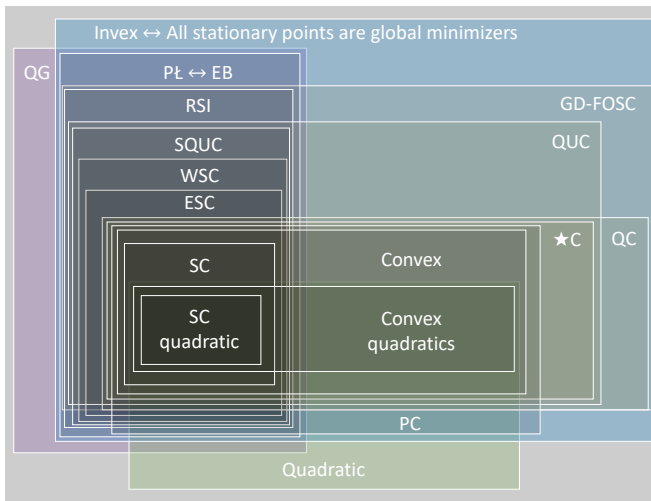


Figure 2: Slope condition nesting diagram when f is continuously differentiable and uniquely minimized. *Caveat emptor - check your assumptions and the literature before claiming an implication / non-implication.*

So far we have only looked at **lower bounds** on the gradient⁸

Often we also need **upper bounds** on the gradient to prove convergence

Almost all analysis assumes a **Lipschitz continuous gradient (LG)**

$$\|\nabla f(x) - \nabla f(y)\| \leq L\|x - y\|, \quad L > 0, \text{ for any } x, y \in \mathbb{R}^n$$

Restricted Lipschitz continuous gradient (RLG) also often suffices

$$\|\nabla f(x)\| \leq L\|x - x^*\|, \quad L > 0, \text{ for any } x \in \mathbb{R}^n$$

Putting RSI and RLG together yields linear convergence of iterates using gradient descent with constant step size

Proof: [▶ Appendix](#)

⁸Except quadratic growth and nonconvex quadratic

A closely-related line of work

- Lessard et al. 2014 [8]
- Fazlyab et al. 2018 [9]
- Lessard slides

Idea: Use tools from robust control theory for nonlinear systems to certify convergence

- **Integral quadratic constraints (IQCs)** restrict the gradient to lie in certain bounded regions
- **Sector IQC** (“radial quasiconvexity”) implies RSI + RLG
 - **Proof:** [▶▶ Appendix](#)
- Solve a small **semidefinite program (SDP)** to find valid step sizes
- Focus is on accelerated gradient methods
 - Gradient descent
 - Polyak momentum
 - Nesterov acceleration

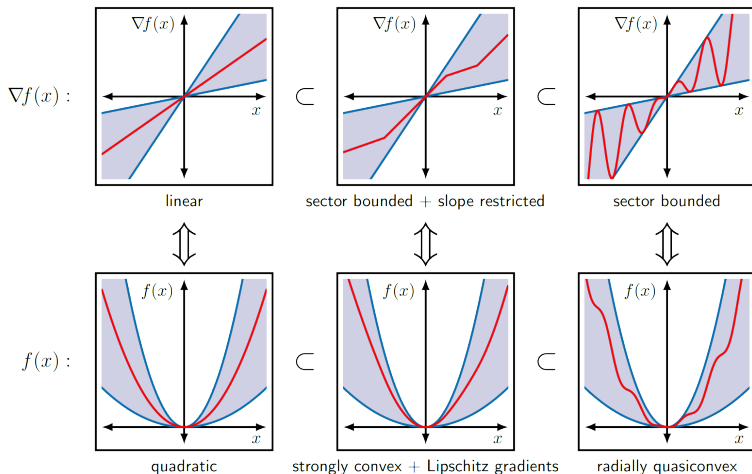


Figure 3: IQC-based slope conditions from Lessard's [slides](#)

Examples of functions in **control theory** that satisfy slope conditions:

- SC quadratic** Finite-horizon LQR cost as function of the state-input sequence
- Convex** Lyapunov-based control design objectives from Dr. Summers' convex optimization course [10]
- Invex** Rosenbrock function
- Polyak-Łojasiewicz inequality** Infinite-horizon LQR cost as function of gain matrix [11] (w/ multiplicative noise [12])
- Weakly quasiconvex** “Idealized risk” in learning linear systems [6]

Key takeaways up to this point:

- Seek a stationary point x^* where $\nabla f(x^*) = 0$.
- If we find x^* , we solved the “easy” problem.⁹
- If f continuously differentiable and invex, then x^* is a global minimizer of f , and we solved the “hard” problem.

⁹To be precise, we seek stationary points by means of *descent*, which avoids finding local maxima. Converging to saddle points is unlikely to happen in practice using random initialization [13].

Phew, OK, take a breath...



...now let's actually solve the
“easy” problem!

Unconstrained Optimization Algorithms

Nesterov's thought experiment (Chapter 1.1.2 of [14])

- 1 Suppose our goal was just to **solve a particular problem** \mathcal{P}_0
- 2 Suppose somehow we managed to solve it and find x^*
- 3 At this point, we already have the best “algorithm” to solve \mathcal{P}_0 , which is to **simply report** x^*
 - i.e. if we had to solve \mathcal{P}_0 again, we would just mindlessly blab x^*
- 4 But what happens if we wanted to **solve a different problem** $\mathcal{P}_1 \neq \mathcal{P}_0$?
- 5 Unless we get exceptionally lucky, our old “algorithm” will give a **wrong answer**
- 6 We don't really care about the solution to a **particular problem**
- 7 We really want an algorithm to solve an **entire class of problems**
 - And the faster it solves, the better!

1 Accelerated gradient methods

- Gradient descent w/ constant step size
- Polyak momentum
- Nesterov acceleration
- RMSprop / AdaGrad / Adam / AdaMax

2 Line-search methods

- Gradient (steepest) descent
- Newton method
- Quasi-Newton methods
- Conjugate gradient methods

3 Trust-region methods

- Dogleg method
- 2D subspace minimization
- Nearly-exact subproblem solution

Line search and trust-region are **iterative descent methods**, meaning that each iteration is designed to decrease the function value.

Distinctions between line-search and trust-region are blurry!

Algorithms can be modified for common applications:

1 Large-scale problems

- Limited-memory quasi-Newton methods
 - L-BFGS
- Inexact Newton methods
 - Conjugate gradient solution of Newton equation
 - Lanczos solution of Newton equation

2 Least-squares problems

- Gauss-Newton (Newton line search using approximate Hessian)
- Levenberg-Marquadt (trust-region using approximate Hessian)

Accelerated Gradient Methods

Motivation: Large-scale ML objectives & gradients are expensive

- Only store a few iterates in memory / hard drive
- Only evaluate gradient once per iteration
- Use cheap noisy gradient estimates in place of exact gradients
- Use heuristics & tricks to get speedups for free

Accelerated gradient methods use updates of the general form

$$x_{k+1} = f(x_k, z_k, \nabla f(y_k); \theta_k)$$

$$z_{k+1} = g(x_k, z_k, \nabla f(y_k); \theta_k)$$

$$y_k = h(x_k, z_k; \theta_k)$$

where

- $x_k \in \mathbb{R}^n$ is the **iterate**
- z_k is the **auxiliary iterate**
- y_k is the **intermediate quantity**
- θ_k are the algorithm **hyperparameters**

This is a **nonlinear dynamical system** where

- f and g act as dynamics functions
- h acts as a measurement function
- $\nabla f(y_k)$ acts as an output-feedback control policy
- Perspective taken in [8], elsewhere

Too many accelerated gradient methods to cover right now

Here are some famous methods & references

- Polyak momentum [15] ([Beautiful interactive explainer](#))
- Nesterov acceleration¹⁰
 - [Original paper \(Russian\)](#) [16]
 - [Nesterov's lecture notes](#) [14]
 - Chapter 3.7 of [Bubeck's book](#) [17]
 - [Sutskever paper](#) [18]
- [Conformal symplectic / relativistic acceleration](#) [19]
 - [Zhang et al.](#) [20] question the connection between symplectic integration of the descent ODE and acceleration
- [RMSprop](#) (never published, only in Hinton's lecture notes)
- [AdaGrad](#) [21]
- [Adam & AdaMax](#) [22]

¹⁰“Nesterov acceleration” actually refers to a collection of closely-related updates, some of which are actually equivalent - there's no universal standard.

Line Search Methods

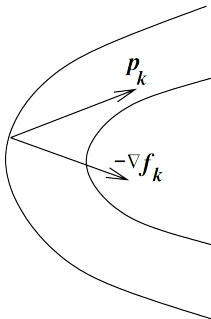
The **line-search method** uses updates of the form

$$x_{k+1} = x_k + \alpha_k p_k$$

where

- $x_k \in \mathbb{R}^n$ is the **current iterate**
- $\alpha_k \in \mathbb{R}$ is the **step size**, also called the **step length** in [1]
- $p_k \in \mathbb{R}^n$ is the **search direction**
- $x_{k+1} \in \mathbb{R}^n$ is the **next iterate**

Idea: Choose α_k and p_k so that $f(x_{k+1}) < f(x_k)$.



A **descent direction** of f at x is any vector p that satisfies

$$\nabla f(x)^\top p < 0$$

i.e. the angle between $\nabla f(x)$ and p is less than 90 degrees.

Question: Why does $\nabla f(x)^\top p < 0$ ensure descent is possible?

From Taylor's theorem:

$$f(x + \alpha p) = f(x) + \alpha \nabla f(x)^\top p + \mathcal{O}(\alpha^2)$$

so $\nabla f(x)^\top p < 0$ implies

$$f(x + \alpha p) < f(x)$$

for all $\alpha > 0$ sufficiently small.

(the linear term dominates the quadratic term for small α)

Use search directions of the form:

$$p_k = -B_k^{-1} \nabla f_k$$

Gradient method $B_k = I$

Newton method $B_k = \nabla^2 f_k$

Quasi-Newton method $B_k \approx \nabla^2 f_k$

Each of these p_k is a descent direction under mild assumptions.

Gradient method ($p_k = -\nabla f_k$)

Assume x_k is not a stationary point so $\nabla f_k \neq 0$
(i.e. we have not solved the problem yet).

Then

$$\nabla f_k^\top p_k = -\nabla f_k^\top \nabla f_k < 0$$

Negative gradient is special - it is always a descent direction away from stationary points.

Newton method ($p_k = -\nabla^2 f_k^{-1} \nabla f_k$)

Assume $\nabla f_k \neq 0$. Also assume either

- 1 f is strictly convex
- 2 x_k is close enough to a local minimizer where the SOSC holds

so that $\nabla^2 f_k \succ 0$. Then

$$\nabla f_k^\top p_k = -\nabla f_k^\top \nabla^2 f_k^{-1} \nabla f_k < 0$$

Quasi-Newton method ($p_k = -B_k^{-1} \nabla f_k$)

Assume $\nabla f_k \neq 0$ and $B_k \succ 0$. Then

$$\nabla f_k^\top p_k = -\nabla f_k^\top B_k^{-1} \nabla f_k < 0$$

Question: What if the Hessian is (nearly) not positive definite?

- Only happens if f is not strictly convex

Concept: Sometimes the negative gradient and the Hessian will not agree on a “good” search direction

- Negative gradient takes precedence since it is always a descent direction
- Need to have at least some contribution of the negative gradient to achieve descent

Modify eigenvalues so

$$B_k = \nabla^2 f_k + E_k \succ \varepsilon I \text{ for some } 0 < \varepsilon \ll 1$$

Easily done with **symmetric eigendecomposition**:

$$\nabla^2 f_k = \sum_{i=1}^n \lambda_i v_i v_i^\top$$

Just replace all $\lambda_i < \varepsilon$ with $\tilde{\lambda}_i = \varepsilon$ and $\lambda \geq \varepsilon$ with $\tilde{\lambda}_i = \lambda_i$ then

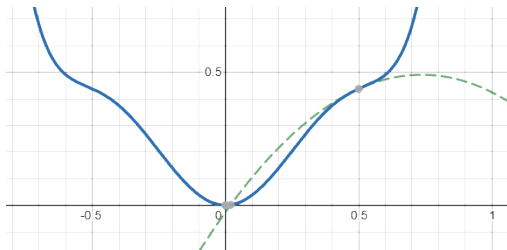
$$B_k = \sum_{i=1}^n \tilde{\lambda}_i v_i v_i^\top \succ \varepsilon I$$

Note: The smaller ε is, the longer the step $p_k = -B_k^{-1} \nabla f_k$ will be along the modified directions

Bad idea: Set components of search direction along directions of negative curvature to zero

- Equivalent to making $\varepsilon \rightarrow \infty$ only for $\lambda_i < 0$
- Prevents progress along negative curvature directions, even if the negative gradient is pointing those directions!
- Not effective

Consider this
interactive
1D example



You would get stuck anytime the Hessian went negative!

Idea: We are already factorizing to solve the Newton equation

$$\nabla^2 f_k p_k = -\nabla f_k$$

Modify the factors to make a positive definite matrix

- Cholesky factorization
- Symmetric indefinite factorization
- See Chapter 3.4 of [1]

Idea: Use a **constant step size**

$$\alpha_k = \alpha$$

- Advantage: Simple
- Advantage: Simplifies theoretical analysis
- Disadvantage: Cannot adapt to changing f_k , ∇f_k , $\nabla^2 f_k$ over iterations
 - If α too big, might get oscillations, divergence
 - If α too small, convergence will take forever
- Disadvantage: Choice of proper α depends on f and x_0
 - Might be difficult or impossible to verify in practice

Idea: Use a **varying history-dependent step size**

$$\alpha_k = \alpha\left(\{x_j, f_j, \nabla f_j, \nabla^2 f_j\}_{j=0}^k\right)$$

- Advantage: Adaptive, can accelerate convergence
- Advantage: State-of-the-art in training huge neural networks
- Disadvantage: Hyperparameter settings require careful tuning
- Disadvantage: Most are heuristics w/o convergence guarantees
- Examples: Accelerated gradient methods mentioned earlier
- Not covered in [1], we'll skip these for now
- See the [survey](#)

Idea: Use a varying step size that solves the 1D **line search** problem

$$\alpha_k = \underset{\alpha}{\operatorname{argmin}} f(x_k + \alpha p_k)$$

- Advantage: Adaptive, maximizes greedy one-step benefit
- Advantage: Guarantees descent if p_k is a descent direction
- Advantage: Formal convergence guarantees
- Disadvantage: Requires extra function evaluations, usually have to settle for an approximate solution

Usually we cannot solve the line search problem exactly

- Need conditions to **certify quality** of inexact line search solutions

Wolfe conditions

The **Wolfe conditions** require

$$f(x_k + \alpha_k p_k) \leq f(x_k) + c_1 \alpha_k \nabla f_k^T p_k \quad (\text{Sufficient decrease})$$

$$\nabla f(x_k + \alpha_k p_k)^T p_k \geq c_2 \nabla f_k^T p_k \quad (\text{Curvature condition})$$

for some constants $0 < c_1 < c_2 < 1$.

Practically, Wolfe conditions are used to select step sizes that ensure

- The function value decreases enough in a relative sense
- The iterates make reasonable progress

Theoretically, Wolfe conditions are used to prove convergence and rates

Maybe you are worried that the Wolfe conditions cannot be satisfied - don't be!

Feasibility of Wolfe conditions (Lemma 3.1 in [1])

Suppose that f is continuously differentiable.

Let p_k be a descent direction at x_k , and assume that f is bounded below along the ray $\{x_k + \alpha p_k \mid \alpha > 0\}$.

Then there exist intervals of step sizes satisfying the Wolfe conditions.

Special case: If f achieves a global minimum at x^* , then f is bounded below, so f is bounded below along any ray and Lemma 3.1 holds.

Question: Now that we know such step sizes exist, how do we find them?

Backtracking Line Search (Algorithm 3.1 in [1])

Choose $\bar{\alpha} > 0$, $\rho \in (0, 1)$, $c \in (0, 1)$; Set $\alpha \leftarrow \bar{\alpha}$;
repeat until $f(x_k + \alpha p_k) \leq f(x_k) + c\alpha \nabla f_k^T p_k$
 $\alpha \leftarrow \rho \alpha$;
end (repeat)
Terminate with $\alpha_k = \alpha$.

Backtracking ensures α_k is never smaller than it needs to be

Maybe you are worried that the backtracking line search will loop forever
- don't be!

- Terminates after a finite number of trials
- Eventually α shrinks enough that the sufficient decrease holds (since p_k is a descent direction).

Note: Backtracking line search satisfies sufficient decrease, but not the curvature Wolfe condition!

- Rules out some theoretical results in [1]

More sophisticated line search algorithms exist:

- Solve the line search problem more exactly
- Guarantee satisfaction of the Wolfe conditions
 - In such a case, call it **Wolfe line search**
- Use interpolation polynomials, bracketing and selection phases
- Tedious to code from scratch
- Implemented in [scipy.optimize](#)
- See Chapter 3.5 of [1]

The Wolfe conditions quantified the permissible step sizes.

The **Zoutendijk condition** quantifies the permissible search directions in terms of the angle between the negative gradient and search direction:

$$\cos \theta_k = \frac{-\nabla f_k^T p_k}{\|\nabla f_k\| \|p_k\|}$$

Zoutendijk Theorem (Theorem 3.2 in [1])

Consider any iteration of the form (3.1), where p_k is a descent direction and α_k satisfies the Wolfe conditions (3.6). Suppose that f is bounded below in \mathbb{R}^n and that f is continuously differentiable in an open set \mathcal{N} containing the level set $\mathcal{L} \stackrel{\text{def}}{=} \{x : f(x) \leq f(x_0)\}$, where x_0 is the starting point of the iteration. Assume also that the gradient ∇f is Lipschitz continuous on \mathcal{N} , that is, there exists a constant $L > 0$ such that

$$\|\nabla f(x) - \nabla f(\tilde{x})\| \leq L\|x - \tilde{x}\|, \quad \text{for all } x, \tilde{x} \in \mathcal{N}. \quad (3.13)$$

Then

$$\sum_{k \geq 0} \cos^2 \theta_k \|\nabla f_k\|^2 < \infty. \quad (3.14)$$

Inequality (3.14) is called the **Zoutendijk condition**.

If this seems technical and obscure, it is...but we can get some practical conditions out of it.

The Zoutendijk condition implies

$$\lim_{k \rightarrow \infty} \cos^2(\theta_k) \|\nabla f_k\|^2 = 0. \quad (3.15)$$

If p_k are chosen so angles θ_k are bounded away from 90 degrees, i.e.

$$\cos(\theta_k) \geq \delta > 0 \text{ for all } k$$

for some constant δ , then (3.15) implies that

$$\lim_{k \rightarrow \infty} \|\nabla f_k\| = 0,$$

i.e. x_k approach a stationary point.

Special case: If $p_k = -\nabla f_k$ then $\cos(\theta_k) = 1$, so **the gradient method with Wolfe line search solves the optimization problem.**

Special case: If $p_k = -B_k^{-1}\nabla f_k$ and the B_k are positive definite with **condition numbers** uniformly bounded by a positive constant M i.e.

$$\|B_k\| \|B_k^{-1}\| \leq M \text{ for all } k$$

then

$$\cos(\theta_k) \geq \frac{1}{M} > 0,$$

Thus **the Newton and quasi-Newton methods with Wolfe line search solve the optimization problem.**

Q-order convergence

The sequence $\{x_k\}$ is

- **Q-linearly convergent** if there exists a constant $r \in (0, 1)$ such that

$$\frac{\|x_{k+1} - x^*\|}{\|x_k - x^*\|} \leq r \text{ for all } k \text{ sufficiently large.}$$

- **Q-superlinearly convergent** if

$$\lim_{k \rightarrow \infty} \frac{\|x_{k+1} - x^*\|}{\|x_k - x^*\|} = 0.$$

- **Q-quadratically convergent** if there exists a constant $M > 0$ such that

$$\frac{\|x_{k+1} - x^*\|}{\|x_k - x^*\|^2} \leq M \text{ for all } k \text{ sufficiently large.}$$

Note: Q-quadratic \Rightarrow Q-superlinear \Rightarrow Q-linear

We established that the **gradient method** converges...but how quickly?

Linear convergence of gradient method (Theorem 3.4 in [1])

Suppose that $f : \mathbb{R}^n \rightarrow \mathbb{R}$ is twice continuously differentiable, and that the iterates generated by the steepest-descent method with exact line searches converge to a point x^ at which the Hessian matrix $\nabla^2 f(x^*)$ is positive definite. Let r be any scalar satisfying*

$$r \in \left(\frac{\lambda_n - \lambda_1}{\lambda_n + \lambda_1}, 1 \right),$$

where $\lambda_1 \leq \lambda_2 \leq \dots \leq \lambda_n$ are the eigenvalues of $\nabla^2 f(x^)$. Then for all k sufficiently large, we have*

$$f(x_{k+1}) - f(x^*) \leq r^2[f(x_k) - f(x^*)].$$

We established that the **Newton method** converges...but how quickly?

Quadratic convergence of Newton method (Theorem 3.5 in [1])

Suppose that f is twice differentiable and that the Hessian $\nabla^2 f(x)$ is Lipschitz continuous (see (A.42)) in a neighborhood of a solution x^ at which the sufficient conditions (Theorem 2.4) are satisfied. Consider the iteration $x_{k+1} = x_k + p_k$, where p_k is given by (3.30). Then*

- (i) if the starting point x_0 is sufficiently close to x^* , the sequence of iterates converges to x^* ;*
- (ii) the rate of convergence of $\{x_k\}$ is quadratic; and*
- (iii) the sequence of gradient norms $\{\|\nabla f_k\|\}$ converges quadratically to zero.*

We established **quasi-Newton methods** converge...but how quickly?

Superlinear convergence of quasi-Newton method (Theorem 3.6 in [1])

Suppose that $f : \mathbb{R}^n \rightarrow \mathbb{R}$ is twice continuously differentiable. Consider the iteration $x_{k+1} = x_k + \alpha_k p_k$, where p_k is a descent direction and α_k satisfies the Wolfe conditions (3.6) with $c_1 \leq 1/2$. If the sequence $\{x_k\}$ converges to a point x^ such that $\nabla f(x^*) = 0$ and $\nabla^2 f(x^*)$ is positive definite, and if the search direction satisfies*

$$\lim_{k \rightarrow \infty} \frac{\|\nabla f_k + \nabla^2 f_k p_k\|}{\|p_k\|} = 0, \quad (3.35)$$

then

- (i) *the step length $\alpha_k = 1$ is admissible for all k greater than a certain index k_0 ; and*
- (ii) *if $\alpha_k = 1$ for all $k > k_0$, $\{x_k\}$ converges to x^* superlinearly.*

Note: Eq. (3.35) holds if the Hessian approximations $B_k \rightarrow \nabla^2 f_k$

Quasi-Newton Methods

Motivation: Newton method converges fast... but Hessians are expensive!

Idea: Approximate Hessian with gradient information at multi-points

- In **finite-differencing**¹¹, the multi-points are chosen close to x_k i.e. $\{x_k + \varepsilon e_i\}$ where e_i are unit vectors, $0 < \varepsilon \ll 1$
- In **model-based derivative-free methods**¹² the multi-points are spread out over a relatively large trust-region centered on x_k
- In **quasi-Newton methods**, the multi-points are chosen as the iterates themselves $\{x_0, x_1, \dots, x_k\}$

¹¹Discussed later. Also see Chapter 8 of [1]

¹²See Chapter 9.2 of [1]

Idea: Approximate the true objective f with a quadratic model m_k

$$f(x_k + p) \approx m_k(p) := f_k + \nabla f_k^\top p + \frac{1}{2} p^\top B_k p$$

Note: The model m matches f in zero- and first-derivative at $p = 0$:

$$\begin{aligned} f_k &= m_k(0) \\ \nabla f_k &= \nabla m_k(0) \end{aligned}$$

Idea: Update quadratic model from m_k to m_{k+1} at each iteration:

$$m_{k+1}(p) = f_{k+1} + \nabla f_{k+1}^\top p + \frac{1}{2} p^\top B_{k+1} p$$

Question: How to choose B_{k+1} given all prior information?

Idea: Make the gradient of m_{k+1} match the gradient of f at
(i) x_{k+1} and (ii) x_k , and (iii) make B_{k+1} symmetric positive definite

Condition (i) holds automatically by construction

$$\nabla f_{k+1} = \nabla m_{k+1}(0)$$

Condition (ii) can be enforced by rearranging the update

$$\begin{aligned}x_{k+1} &= x_k + \alpha_k p_k \\ \Leftrightarrow \quad x_k &= x_{k+1} - \alpha_k p_k\end{aligned}$$

and putting it into m_{k+1} as

$$\nabla f_k = \nabla m_{k+1}(-\alpha_k p_k) = \nabla f_{k+1} - \alpha_k B_{k+1} p_k$$

Rearranging we obtain the **secant equation**

$$B_{k+1} s_k = y_k$$

where

$$\begin{aligned}s_k &= x_{k+1} - x_k = \alpha_k p_k \\ y_k &= \nabla f_{k+1} - \nabla f_k\end{aligned}$$

Question: Is the secant equation solvable for some $B_{k+1} \succ 0$?

Answer: Yes, so long as the **curvature condition**

$$y_k^\top s_k > 0$$

is satisfied.

- If f is strictly convex, then $y^\top s > 0$ holds for any $s = z - x$ and $y = \nabla f(z) - \nabla f(x)$
- If α_k satisfies the Wolfe conditions, then $y_k^\top s_k > 0$ holds

Matrix B_{k+1} starts off with n^2 unknowns

- Enforcing the secant equation $B_{k+1}s_k = y_k$ removes n unknowns
- Enforcing symmetry $B_{k+1} = B_{k+1}^\top$ removes $n(n-1)/2$ unknowns
- Enforcing positive definiteness $B_{k+1} \succ 0$ removes n unknowns (requires all n eigenvalues are positive)

Still leaves

$$n^2 - \frac{n(n-1)}{2} - n - n = \frac{n(n-3)}{2}$$

additional degrees-of-freedom, which is positive for any $n \geq 4$

To determine B_{k+1} uniquely, impose the additional condition

“Among all symmetric positive definite matrices satisfying the secant equation, B_{k+1} is closest to the current matrix B_k ”

$$\begin{aligned} B_{k+1} &= \arg \min_B \|B - B_k\| \\ \text{s.t.} \quad &Bs_k = y_k \\ &B = B^\top \succ 0 \end{aligned}$$

If we choose the **weighted Frobenius matrix norm**

$$\|A\|_W = \|W^{1/2} A W^{1/2}\|_F$$

with weight matrix equal to the **inverse average Hessian** of f between x_k and x_{k+1}

$$W = \overline{G}^{-1} = \left[\int_0^1 \nabla^2 f(\tau x_k + \tau(x_{k+1} - x_k)) d\tau \right]^{-1}$$

then the unique solution is the **Davidon-Fletcher-Powell (DFP) Hessian update**

$$B_{k+1} = \left(I - \frac{s_k y_k^\top}{y_k^\top s_k} \right)^\top B_k \left(I - \frac{s_k y_k^\top}{y_k^\top s_k} \right) + \frac{y_k y_k^\top}{y_k^\top s_k}$$

Search direction is computed as

$$p_k = -B_k^{-1} \nabla f_k$$

Useful to get the inverse Hessian $H_k = B_k^{-1}$ directly

- Avoid solving linear equations $B_k p_k = -\nabla f_k$
- Becomes a simple matrix-vector product $p_k = -H_k \nabla f_k$

Apply Sherman–Morrison–Woodbury matrix inversion identity to the DFP Hessian update to get the **DFP inverse Hessian update**

$$H_{k+1} = H_k - \frac{H_k y_k y_k^\top H_k}{y_k^\top H_k y_k} + \frac{s_k s_k^\top}{y_k^\top s_k}$$

Idea: Instead of minimizing the Hessian deviations $\|B_{k+1} - B_k\|$, we could minimize the inverse Hessian deviations $\|H_{k+1} - H_k\|$

$$\begin{aligned} H_{k+1} = \arg \min_H \quad & \|H - H_k\| \\ \text{s.t.} \quad & Hy_k = s_k \\ & H = H^\top \succ 0 \end{aligned}$$

If we choose the weighted Frobenius matrix norm $\|A\|_W$ with weight matrix equal to the **average Hessian** $W = \overline{G}$, then the unique solution is the **Broyden-Fletcher-Goldfarb-Shanno (BFGS) inverse Hessian update**

$$H_{k+1} = \left(I - \frac{y_k s_k^\top}{y_k^\top s_k} \right)^\top H_k \left(I - \frac{y_k s_k^\top}{y_k^\top s_k} \right) + \frac{s_k s_k^\top}{y_k^\top s_k}$$

Applying the Sherman–Morrison–Woodbury matrix inversion identity gives the **BFGS Hessian update**

$$B_{k+1} = B_k - \frac{B_k s_k s_k^\top B_k}{s_k^\top B_k s_k} + \frac{y_k y_k^\top}{y_k^\top s_k}$$

Concepts:

- B_{k+1} (or H_{k+1}) is a **rank-2 update** from B_k (or H_k) based on most recent iterate x_k and gradient ∇f_k information
- B_k (or H_k) encodes information about curvature from **all previous updates**
- Similar flavor to recursive least-squares / Kalman filter *a posteriori* state covariance updates e.g.

$$P_{k+1} = (I - K_k H_k) P_k (I - K_k H_k)^\top + K_k R_k K_k^\top$$

Question: Is BFGS or DFP better?

- Historically, DFP was developed first
- Empirically, BFGS usually outperforms DFP
- Theoretically, BFGS has better self-correcting Hessian approximation properties
- Many optimization libraries do not even bother implementing DFP
e.g. `scipy.optimize`

Idea: Instead of a rank-2 update like BFGS or DFP, find a **symmetric rank-1 (SR1)** update

$$B_{k+1} = B_k + \sigma v v^\top$$

that satisfies the secant equation $B_{k+1} s_k = y_k$, where $\sigma \in \{1, -1\}$ and $v \in \mathbb{R}^n$

Since σ can be negative, B_{k+1} **may not be positive definite**, even if B_k is

Substitute the SR1 update into the secant equation:

$$\begin{aligned}B_{k+1} s_k &= y_k \\(B_k + \sigma v v^\top) s_k &= y_k \\ \sigma [\sigma v^\top s_k] v &= y_k - B_k s_k\end{aligned}$$

Since $\sigma v^\top s_k$ is a scalar, deduce that for some other scalar δ

$$v = \delta (y_k - B_k s_k)$$

Substitute v to get

$$(y_k - B_k s_k) = \sigma \delta^2 [s_k^\top (y_k - B_k s_k)] (y_k - B_k s_k)$$

which is solved only by

$$\begin{aligned}\sigma &= \text{sign} [s_k^\top (y_k - B_k s_k)] \\ \delta &= |s_k^\top (y_k - B_k s_k)|^{-1/2}\end{aligned}$$

Putting the feasible σ and δ into $B_{k+1} = B_k + \sigma vv^\top$ gives the **SR1 Hessian update**

$$B_{k+1} = B_k + \frac{(y_k - B_k s_k)(y_k - B_k s_k)^\top}{(y_k - B_k s_k)^\top s_k}$$

Applying the Sherman–Morrison–Woodbury matrix inversion identity gives the **SR1 inverse Hessian update**

$$H_{k+1} = H_k + \frac{(s_k - H_k y_k)(s_k - H_k y_k)^\top}{(s_k - H_k y_k)^\top y_k}$$

Issue: Unlike BFGS/DFP, the SR1 matrices often become **indefinite**

- Actually an **advantage** since true Hessian may be indefinite
- **Fix 1:** Make eigenvalues positive for **line-search** (not recommended by [1])
- **Fix 2:** Use indefinite B_k as-is in a **modified trust-region method** (Algorithm 6.2 in [1])

Issue: As denominator $s_k^T(y_k - B_k s_k) \rightarrow 0$, Hessian B_{k+1} blows up

- **Fix:** Use a **skipping rule**: set constant $0 < r \ll 1$. If

$$|s_k^T(y_k - B_k s_k)| < r \|s_k\| \|y_k - B_k s_k\|$$

set $B_{k+1} = B_k$, else apply the SR1 update

Question: How should we initialize the inverse Hessian estimate H_0 ?

- **Init 1:** Set it to a multiple of identity $H_0 = \beta_0 I$
 - Set $\beta_0 = \delta_0 / \|\nabla f_0\|$ where δ_0 is a desired initial step size
- **Init 2:** Estimate $\nabla^2 f_0$ using finite differences on $\nabla f(x)$

Post-process: After computing x_1 and ∇f_1 , but before computing H_1 , correct H_0 by setting $H_0 = \gamma_0 I$ where

$$\gamma_0 = \frac{y_0^\top s_0}{y_0^\top y_0}$$

- Scalar γ_0 approximates an eigenvalue of $\nabla^2 f_0^{-1}$
- Effective heuristic

Convergence of BFGS (Theorem 6.5 in [1])

Assumption 6.1.

- (i) *The objective function f is twice continuously differentiable.*
- (ii) *The level set $\mathcal{L} = \{x \in \mathbb{R}^n \mid f(x) \leq f(x_0)\}$ is convex, and there exist positive constants m and M such that*

$$m\|z\|^2 \leq z^T G(x)z \leq M\|z\|^2 \quad (6.39)$$

for all $z \in \mathbb{R}^n$ and $x \in \mathcal{L}$.

Theorem 6.5.

Let B_0 be any symmetric positive definite initial matrix, and let x_0 be a starting point for which Assumption 6.1 is satisfied. Then the sequence $\{x_k\}$ generated by Algorithm 6.1 (with $\epsilon = 0$) converges to the minimizer x^ of f .*

Note: Assumption 6.1 requires that f be **strongly convex with a Lipschitz gradient** on the sublevel set below the initial function value

Superlinear convergence of BFGS (Theorem 6.6 in [1])

Assumption 6.2.

The Hessian matrix G is Lipschitz continuous at x^ , that is,*

$$\|G(x) - G(x^*)\| \leq L\|x - x^*\|,$$

for all x near x^ , where L is a positive constant.*

Theorem 6.6.

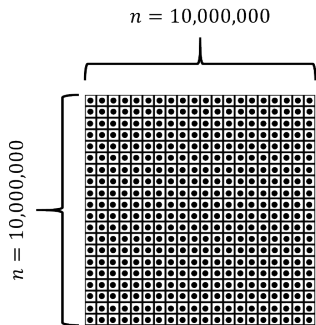
Suppose that f is twice continuously differentiable and that the iterates generated by the BFGS algorithm converge to a minimizer x^ at which Assumption 6.2 holds. Suppose also that*

$$\sum_{k=1}^{\infty} \|x_k - x^*\| < \infty. \quad (6.52)$$

Then x_k converges to x^ at a superlinear rate.*

Note: Iterates of BFGS converge under Theorem 6.5, which satisfies one of the assumptions of Theorem 6.6

Motivation: Storing full $n \times n$ approximate Hessians can be intractable!



$$(1e7)^2 \text{ entries} \times 8 \text{ bytes / entry} \\ = 800 \text{ TB}$$



Recall: Full Hessian approximation captures gradient information from **all previous iterations**

Idea: Instead, just store $m \ll n$ pairs of length- n vectors associated with the **most recent few iterations**

Doing this uses **limited memory**

Special structure of H_k and updates in **BFGS** allows efficient reconstruction of $p_k = -H_k \nabla f_k$ **without explicitly constructing H_k**

Thus we will develop **limited-memory BFGS (L-BFGS)**

Recall: BFGS updates

$$\begin{aligned}x_{k+1} &= x_k - \alpha_k H_k \nabla f_k \\ H_{k+1} &= V_k^\top H_k V_k + \rho_k s_k s_k^\top\end{aligned}$$

where

$$\rho_k = \frac{1}{y_k^\top s_k}, \quad V_k = I - y_k s_k^\top$$

Unroll the recursion for H_{k+1} for m steps as

$$\begin{aligned}H_k &= (V_{k-1}^\top \cdots V_{k-m}^\top) H_k^0 (V_{k-m} \cdots V_{k-1}) \\ &\quad + \rho_{k-m} (V_{k-1}^\top \cdots V_{k-m+1}^\top) s_{k-m} s_{k-m}^\top (V_{k-m+1} \cdots V_{k-1}) \\ &\quad + \rho_{k-m+1} (V_{k-1}^\top \cdots V_{k-m+2}^\top) s_{k-m+1} s_{k-m+1}^\top (V_{k-m+2} \cdots V_{k-1}) \\ &\quad + \cdots \\ &\quad + \rho_{k-1} s_{k-1} s_{k-1}^\top\end{aligned}$$

The unrolled expression can be computed recursively:

L-BFGS two-loop Hessian recursion (Algorithm 7.4 in [1])

```

$$\begin{aligned} & q \leftarrow \nabla f_k; \\ & \text{for } i = k - 1, k - 2, \dots, k - m \\ & \quad \alpha_i \leftarrow \rho_i s_i^T q; \\ & \quad q \leftarrow q - \alpha_i y_i; \\ & \text{end (for)} \\ & r \leftarrow H_k^0 q; \\ & \text{for } i = k - m, k - m + 1, \dots, k - 1 \\ & \quad \beta \leftarrow \rho_i y_i^T r; \\ & \quad r \leftarrow r + s_i (\alpha_i - \beta) \\ & \text{end (for)} \\ & \text{stop with result } H_k \nabla f_k = r. \end{aligned}$$

```

L-BFGS (Algorithm 7.5 in [1])

Choose starting point x_0 , integer $m > 0$;

$k \leftarrow 0$;

repeat

 Choose H_k^0 (for example, by using (7.20));

 Compute $p_k \leftarrow -H_k \nabla f_k$ from Algorithm 7.4;

 Compute $x_{k+1} \leftarrow x_k + \alpha_k p_k$, where α_k is chosen to
 satisfy the Wolfe conditions;

if $k > m$

 Discard the vector pair $\{s_{k-m}, y_{k-m}\}$ from storage;

 Compute and save $s_k \leftarrow x_{k+1} - x_k, y_k = \nabla f_{k+1} - \nabla f_k$;

$k \leftarrow k + 1$;

until convergence.

Question: How should we initialize the inverse Hessian estimate H_k^0 ?

- Use the same scaled identity as in the post-process step from BFGS:
 $H_k^0 = \gamma_k I$ where

$$\gamma_k = \frac{y_{k-1}^\top s_{k-1}}{y_{k-1}^\top y_{k-1}}$$

- Estimates the size of $\nabla^2 f_k$ along the most recent search direction

Question: How should we choose the number m of stored vector pairs?

- Problem dependent
- No clear answer
- 10-20 seems to be a good rule-of-thumb empirically

Performance of L-BFGS (Table 7.1 in [1])

Problem	n	L-BFGS $m = 3$		L-BFGS $m = 5$		L-BFGS $m = 17$		L-BFGS $m = 29$	
		nfg	time	nfg	time	nfg	time	nfg	time
DIXMAANL	1500	146	16.5	134	17.4	120	28.2	125	44.4
EIGENALS	110	821	21.5	569	15.7	363	16.2	168	12.5
FREUROTH	1000	>999	—	>999	—	69	8.1	38	6.3
TRIDIA	1000	876	46.6	611	41.4	531	84.6	462	127.1

Trust-Region Methods

Idea: Approximate the true objective f with a quadratic model m :

$$f(x + p) \approx m(p) := f(x) + \nabla f(x)^\top p + \frac{1}{2} p^\top B p$$

Idea: Only trust the model on a small region around the expansion point x i.e. only consider

$$\|p\| \leq \Delta$$

Together this yields the **trust-region subproblem**

$$\begin{array}{ll} \underset{p}{\text{minimize}} & m(p) = f(x) + \nabla f(x)^\top p + \frac{1}{2} p^\top \nabla^2 f(x) p \\ \text{subject to} & \|p\| \leq \Delta \end{array}$$

Trust-region methods (Figure 2.4 in [1])

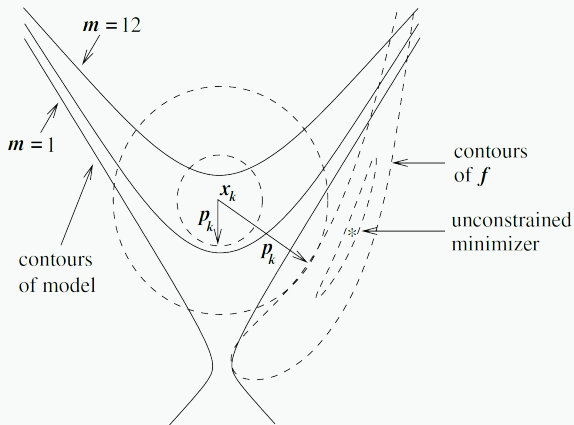


Figure 4: Two possible trust regions (circles) and their corresponding steps p_k . The solid lines are contours of the model function m_k

How do we know if the trust-region is too big or small?

Define the **reduction ratio**

$$\rho_k = \frac{f(x_k) - f(x_k + p_k)}{m_k(0) - m_k(p_k)} = \frac{\text{actual reduction}}{\text{modeled reduction}}$$

- $\rho_k < 0$ is really bad
 - Step would increase the function value
 - Need to shrink trust-region & re-solve subproblem
- $0 < \rho_k \ll 1$ is bad
 - Step would not make much progress
 - Need to shrink trust-region & re-solve subproblem
- $\rho_k \approx 1$ is best
 - Model agrees with the true function
 - Take the step & enlarge the trust region
- $\rho_k \gg 1$ is good
 - Model is being too conservative
 - Take the step & enlarge the trust region

Trust-region algorithm (Algorithm 4.1 in [1])

```
Given  $\hat{\Delta} > 0$ ,  $\Delta_0 \in (0, \hat{\Delta})$ , and  $\eta \in [0, \frac{1}{4}]$ :  
for  $k = 0, 1, 2, \dots$   
    Obtain  $p_k$  by (approximately) solving (4.3);  
    Evaluate  $\rho_k$  from (4.4);  
    if  $\rho_k < \frac{1}{4}$   
         $\Delta_{k+1} = \frac{1}{4} \Delta_k$   
    else  
        if  $\rho_k > \frac{3}{4}$  and  $\|p_k\| = \Delta_k$   
             $\Delta_{k+1} = \min(2\Delta_k, \hat{\Delta})$   
        else  
             $\Delta_{k+1} = \Delta_k$ ;  
    if  $\rho_k > \eta$   
         $x_{k+1} = x_k + p_k$   
    else  
         $x_{k+1} = x_k$ ;  
end (for).
```

Solves the trust-region subproblem and encodes trust-region shrinkage/expansion logic.

Since $\nabla^2 f(x)$ may not be positive semidefinite, the trust-region subproblem is generally a **nonconvex** problem.

However, we can still show that **strong duality** holds

- Best bound from the **Lagrange dual** is tight (zero **duality gap**)

Strong duality of the trust-region subproblem (Theorem 4.1 in [1])

The vector p^ is a global solution of the trust-region problem*

$$\min_{p \in \mathbb{R}^n} m(p) = f + g^T p + \frac{1}{2} p^T B p, \quad \text{s.t. } \|p\| \leq \Delta, \quad (4.7)$$

if and only if p^ is feasible and there is a scalar $\lambda \geq 0$ such that*

$$(B + \lambda I)p^* = -g, \quad (4.8a)$$

$$\lambda(\Delta - \|p^*\|) = 0, \quad (4.8b)$$

$$(B + \lambda I) \text{ is positive semidefinite.} \quad (4.8c)$$

- Direct proof - see Chapter 4.3 of [1]
- Proof using S-procedure - see Appendix B of [2]

How to solve the trust-region subproblem?

- Form the Lagrange dual problem & solve in CVX (expensive)
- Use a specialized approximate solution (cheap)

Remember: We have to solve a new subproblem at each iteration, so we want to get “good enough” solutions quickly

Approximations effectively try to solve the KKT conditions in Thm. 4.1

Idea: Do something really dumb and use a **linear** model:

$$f(x + p) \approx f(x) + \nabla f(x)^\top p$$

Then the trust-region subproblem is convex:

$$\begin{aligned} p^S &= \arg \min_p f(x) + \nabla f(x)^\top p \\ \text{s.t.} \quad &\|p\| \leq \Delta \end{aligned}$$

Closed-form solution is

$$p^S = -\frac{\Delta}{\|\nabla f(x)\|} \nabla f(x)$$

If $p_k = p^S$ we get **gradient descent** with step sizes $\alpha_k = \frac{\Delta_k}{\|\nabla f_k\|}$.

Works OK, but do not expect better-than-linear convergence.

Idea: Improve on p^S by finding the point p^C on the ray from 0 to p^S that minimizes the quadratic model m over the trust region:

$$\begin{aligned} \tau^C &= \arg \min_{\tau} m(\tau p^S) \\ \text{s.t.} \quad &\|\tau p^S\| \leq \Delta, \quad \tau > 0 \end{aligned}$$

Closed-form solution is

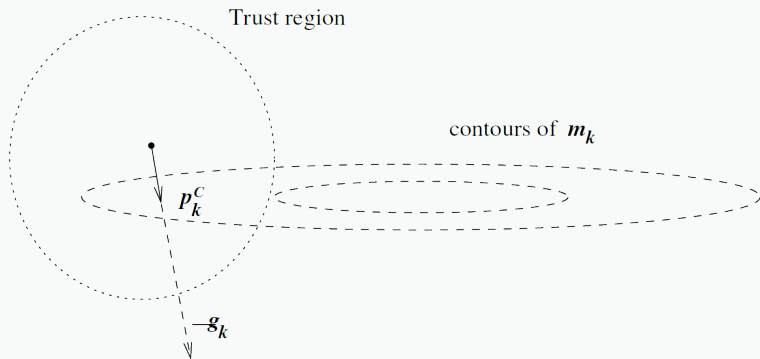
$$\tau^C = \begin{cases} 1 & \text{if } a(x) \leq 0 \\ \min \left(\frac{\|\nabla f(x)\|^3}{\Delta \cdot a(x)}, 1 \right) & \text{otherwise.} \end{cases}$$

where $a(x) = \nabla f(x)^\top \nabla^2 f(x) \nabla f(x)$

The point $p^C = \tau^C p^S$ is called the **Cauchy point**.

Still $p_k = p^C$ yields **gradient descent** with funky step sizes.

Cauchy point (Figure 4.3 in [1])



Idea: Delete the trust-region constraint.

Then the trust-region subproblem is

$$\underset{p}{\text{minimize}} \quad m(p) = f(x) + \nabla f(x)^\top p + \frac{1}{2} p^\top \nabla^2 f(x) p$$

When $\nabla^2 f(x) \succ 0$ the solution is

$$p^B = -\nabla^2 f(x)^{-1} \nabla f(x)$$

which is precisely the **Newton step**, also called the **full step**.

Idea: What happens to p as the trust radius Δ varies from 0 to ∞ ?

Case 1: $\Delta \rightarrow 0$

Since $\|p\| \leq \Delta$ also $\|p\| \rightarrow 0$, so the term $p^\top \nabla^2 f(x) p$ becomes negligible.

Solution is the **gradient step** $p = p^S = -\frac{\Delta}{\|\nabla f(x)\|} \nabla f(x)$

Case 2: $\Delta \rightarrow \infty$

The problem becomes unconstrained.

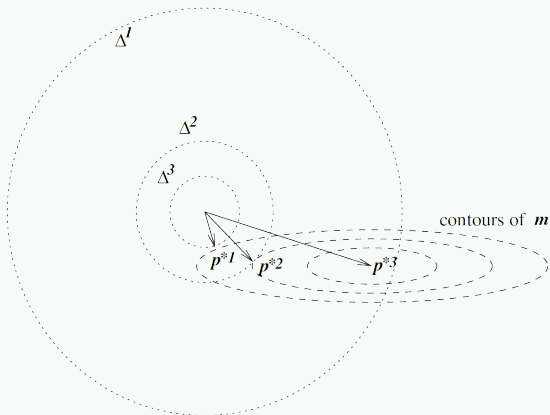
Solution is the **Newton step** $p = p^B = -\nabla^2 f(x)^{-1} \nabla f(x)$

Case 3: $0 < \Delta < \infty$

Solution follows a curved path.

Trust-region interpolates between gradient and Newton method in a principled way.

Trust-region subproblem solutions (Figure 4.2 in [1])



Trust-region subproblem solutions for different radii $\Delta^1, \Delta^2, \Delta^3$.

Idea: Replace the curved path with a pair of line segments running from the origin to p^U to p^B :

$$\tilde{p}(\tau) = \begin{cases} \tau p^U, & 0 \leq \tau \leq 1 \\ p^U + (\tau - 1)(p^B - p^U), & 1 \leq \tau \leq 2 \end{cases}$$

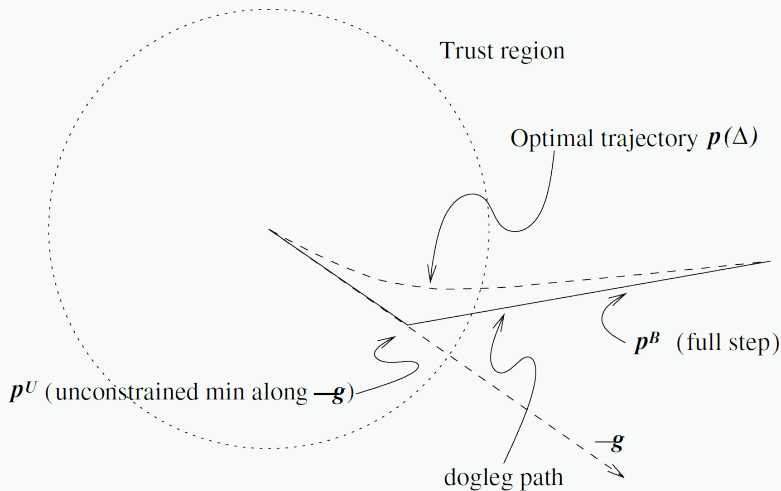
where p^U is the minimizer of m along the negative gradient direction

$$p^U = -\frac{\nabla f(x)^\top \nabla f(x)}{\nabla f(x)^\top \nabla^2 f(x) \nabla f(x)}$$

Then choose p as the point on the path that minimizes m

This is called the **dogleg path**

Dogleg path (Figure 4.4 in [1])



Case 1: $0 < \Delta \leq \|p^U\|$

The dogleg path intersects the trust-region boundary

The point on the dogleg path is the Cauchy point p^C

Find $0 < \tau < 1$ as

$$\tau = \frac{\|\Delta\|}{\|p^U\|}$$

and the solution is $p = \tilde{p}(\tau)$.

Case 2: $\|p^U\| \leq \Delta \leq \|p^B\|$

The dogleg path intersects the trust-region boundary

The point on the dogleg path is a convex combination of the point p^U and the full step p^B

Find $\tau > 1$ as the root of the quadratic equation

$$\|p^U + (\tau - 1)(p^B - p^U)\|^2 = \Delta^2$$

and the solution is $p = \tilde{p}(\tau)$.

Case 3: $\Delta \geq \|p^B\|$

The dogleg path does not intersect the trust-region boundary

The solution is the full step $p = p^B$.

Idea: Generalize the dogleg path by searching over the entire **2D-subspace** spanned by p^U and p^B

Equivalent to the 2D-subspace spanned by

- The **gradient direction** $\nabla f(x)$
- The **Newton direction** $\nabla^2 f(x)^{-1} \nabla f(x)$

2D-subspace **includes the entire dogleg path**

- 2D-subspace solutions are **at least as good as dogleg solutions**

The trust-region subproblem is modified by adding the subspace constraint:

$$\begin{aligned} \underset{p}{\text{minimize}} \quad & m(p) = f(x) + \nabla f(x)^\top p + \frac{1}{2} p^\top \nabla^2 f(x) p \\ \text{subject to} \quad & \|p\| \leq \Delta \\ & p \in \text{span}\left(-\nabla f(x), -\nabla^2 f(x)^{-1} \nabla f(x)\right) \end{aligned}$$

- Only 2 decision variables (the projection of p onto the 2D subspace)
- Reduces to finding the roots of a degree-4 polynomial¹³
- Efficiently solved in closed-form using quartic formula¹⁴
- Handle indefinite $\nabla^2 f(x)$ by adding a multiple of identity

¹³For details see the blog post at <https://nmayorov.wordpress.com/2015/07/01/2d-subspace-trust-region-method/>

¹⁴We got pretty lucky - Abel-Ruffini theorem tells us that degree-5 polynomials and

above do not have closed-form solutions!

Conjugate gradient method can approximately solve the trust-region subproblem by improving on the Cauchy point

- Very similar to dogleg, 2D-subspace minimization
- Suitable for large-scale problems
- Algorithm is due to Steihaug [23]
- See Chapter 5 of [1] for details on conjugate gradient
- See Chapter 7.1 of [1] for details on Steihaug-CG algorithm

Our final method to solve the trust-region subproblem is called the **iterative solution**

- Idea is to solve the dual of the subproblem directly by tweaking the Lagrange multiplier λ
- At each iteration
 - Cholesky factorize $\nabla^2 f(x) + \lambda I = R^\top R$
 - Update λ based on $\nabla f(x)$ and R
- Development is rather technical - see Chapter 4.3 of [1]
- More expensive than dogleg, 2D-subspace, or CG-Steihaug, but more accurate - gives **nearly exact solutions**

Convergence to stationary points is based on the following estimate of decrease in the model function:

$$m_k(0) - m_k(p_k) \geq c_1 \|\nabla f_k\| \min \left(\Delta_k, \frac{\|\nabla f_k\|}{\|\nabla^2 f_k\|} \right),$$

for some constant $c_1 \in (0, 1]$.

The Cauchy point p^C satisfies the bound with $c_1 = 1/2$.

Since the dogleg, 2D-subspace, and CG-Steihaug solutions are at least as good as the Cauchy point, they also satisfy this bound.

Convergence of trust-region methods (Theorem 4.6 in [1])

Let $\eta \in (0, \frac{1}{4})$ in Algorithm 4.1. Suppose that $\|B_k\| \leq \beta$ for some constant β , that f is bounded below on the level set S (4.24) and Lipschitz continuously differentiable in $S(R_0)$ for some $R_0 > 0$, and that all approximate solutions p_k of (4.3) satisfy the inequalities (4.20) and (4.25) for some positive constants c_1 and γ . We then have

$$\lim_{k \rightarrow \infty} g_k = 0. \quad (4.33)$$

Dogleg, 2D-subspace, and CG-Steihaug trust-region methods all converge to stationary points and solve the optimization problem.

Different analysis shows **the iterative solution trust-region method converges to stationary points and solves the optimization problem.**

We established that **trust-region methods** converge...but how quickly?

Key is to realize that, for all k sufficiently large, the steps p_k either

- 1 Become **asymptotically similar** to the Newton step:

$$\|p_K - p_k^N\| = o(\|p_k^N\|)$$

- 2 Become **identical** to the Newton step:

$$\|p_K - p_k^N\| = 0$$

This leads to

- 1 Superlinear convergence (Theorem 4.9)
- 2 Quadratic convergence (Theorem 3.5)

Superlinear convergence of trust-region (Theorem 4.9 in [1])

Let f be twice Lipschitz continuously differentiable in a neighborhood of a point x^ at which second-order sufficient conditions (Theorem 2.4) are satisfied. Suppose the sequence $\{x_k\}$ converges to x^* and that for all k sufficiently large, the trust-region algorithm based on (4.3) with $B_k = \nabla^2 f(x_k)$ chooses steps p_k that satisfy the Cauchy-point-based model reduction criterion (4.20) and are asymptotically similar to Newton steps p_k^N whenever $\|p_k^N\| \leq \frac{1}{2} \Delta_k$, that is,*

$$\|p_k - p_k^N\| = o(\|p_k^N\|). \quad (4.53)$$

Then the trust-region bound Δ_k becomes inactive for all k sufficiently large and the sequence $\{x_k\}$ converges superlinearly to x^ .*

Dogleg, 2D-subspace, and iterative solution also eventually use $p_k = p_k^N$, so they converge quadratically.

Least-squares Methods

In **nonlinear least-squares** problems, the objective f takes the form

$$f(x) = \frac{1}{2} \|r(x)\|^2$$

where

- $r(x) : \mathbb{R}^n \rightarrow \mathbb{R}^m$ is a nonlinear **residual** function
- n is the number of **parameters** (e.g. in a prediction model)
- m is the number of **observations** (e.g. collected from experiments)

The **Jacobian** of the residuals is

$$J(x) = \left[\frac{\partial r_j}{\partial x_i} \right]_{j=1,2,\dots,m} = \begin{bmatrix} \nabla r_1(x)^T \\ \nabla r_2(x)^T \\ \vdots \\ \nabla r_m(x)^T \end{bmatrix}$$

The gradient and Hessian are then

$$\begin{aligned}\nabla f(x) &= J(x)^T r(x) \\ \nabla^2 f(x) &= J(x)^T J(x) + \sum_{j=1}^m r_j(x) \nabla^2 r_j(x)\end{aligned}$$

Idea: Approximate the Hessian as

$$\nabla^2 f(x) \approx J(x)^T J(x)$$

Need to compute $J(x)$ for $\nabla f(x)$, so we get this for free

Two cases when the Hessian approximation is good:

- If $r(x)$ is small (model fits data well)
- If $\nabla^2 r_j(x)$ is small (residuals nearly affine in parameters)

Gauss-Newton

- Line-search quasi-Newton using the Hessian approximation

$$B_k = J_k^T J_k$$

Can be solved very efficiently

- We only have to compute J_k
- The Newton equation is a **normal equation**

$$J_k^T J_k p_k = -J_k r_k$$

which can be solved with QR or SVD techniques - see Ch. 10.2 of [1]

Levenberg-Marquadt

- Trust-region using the Hessian approximation $B_k = J_k^T J_k$

Can be solved very efficiently because

- We only have to compute J_k
- The KKT conditions for the trust-region subproblem specialize and can be solved using linear algebra techniques - see Ch. 10.3 of [1]

Optimization of Nonsmooth Functions

So far we only considered objective functions that are smooth

- We demanded the gradient $\nabla f(x)$ exist everywhere

But sometimes objective functions are not differentiable at certain points

What can we do?

- 1 Smoothing [[Vandenberghe's notes](#)]
- 2 Subgradient method [[Boyd's notes](#)]
- 3 Proximal method [[Parikh & Boyd tutorial](#)]

Old idea from Huber 1964 [24]

Approximate $f(x) = |x|$ with a quadratic tip

$$\hat{f}(x) = \begin{cases} |x| - \frac{\mu}{2} & \text{if } |x| \geq \mu \\ \frac{x^2}{2\mu} & \text{if } |x| < \mu \end{cases}$$

As $\mu \rightarrow 0$, get $\hat{f} \rightarrow f$

In practice: gradually decrease μ over iterations

A **subgradient** g of a convex function at x is any vector that satisfies

$$f(y) \geq f(x) + g^T(y - x) \text{ for all } y$$

i.e. a supporting hyperplane of the epigraph of f

The **subdifferential** $\partial f(x)$ is the set of all subgradients g at x

- $g \in \partial f(x)$ means g is a subgradient
- If f differentiable at x , then $\partial f(x) = \{\nabla f(x)\}$

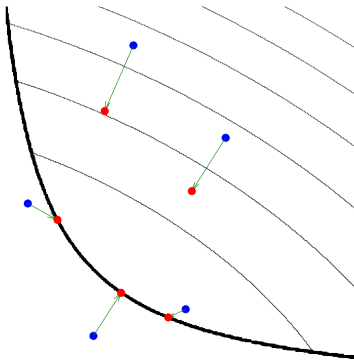
Note: Unlike gradients, subgradients may **fail to be descent directions**

Subgradient method

$$x_{k+1} = x_k - \alpha_k g_k \text{ where } g_k \in \partial f(x_k)$$

$$f_{k+1}^{\text{best}} = \min\{f(x_{k+1}), f_k^{\text{best}}\}$$

$$\text{prox}_{\lambda f}(x) = \underset{y}{\operatorname{argmin}} \{f(y) + (1/2\lambda)\|y - x\|_2^2\}$$



Interpretations (Chapter 3 of [25])

- 1 Gradient descent on the Moreau envelope of f
- 2 Resolvent of subdifferential operator
- 3 Trust-region subproblem

Evaluating $\text{prox}_{\lambda f}(x)$ (Chapter 6 of [25])

- For some simple f , evaluate in closed form
- For complicated f , use specialized algorithms

Proximal algorithm (Chapter 4.1 of [25])

$$x_{k+1} = \mathbf{prox}_{\alpha_k f}(x_k)$$

Interpretations

- 1 Disappearing quadratic regularization

$$\text{minimize} \quad f(x) + (1/2\lambda) \|x - x^k\|_2^2$$

- Damping term that encourages x_{k+1} to be close to x_k

- 2 Backward Euler numerical integration of gradient flow

$$\frac{d}{dt}x(t) = -\nabla f(x(t))$$

- Contrast w/ gradient descent, which is forward Euler

Special decomposition of objective

$$\text{minimize} \quad \underbrace{f(x)}_{\text{smooth}} + \underbrace{g(x)}_{\text{nonsmooth, convex}}$$

Application: sparsity-promoting regularization via ℓ_1 -norm

$$\text{minimize} \quad f(x) + \|x\|_1$$

(See Chapter 6 of Boyd CVX textbook)

Proximal gradient method (Chapter 4.2 of [25])

$$y_{k+1} = x_k - \alpha_k \nabla f(x_k)$$

$$x_{k+1} = \mathbf{prox}_{\alpha_k g}(y_{k+1})$$

Interpretations

- 1 Forward-backward splitting numerical integration of gradient flow

$$\frac{d}{dt}x(t) = -\nabla f(x(t)) - \nabla g(x(t))$$

- Forward Euler for the differentiable part f
- Backward Euler for the nondifferentiable part g

- 2 Others (see Chapter 4.2 of [25])

Calculating Derivatives

Let's face it, calculating derivatives by hand is hard.

Wouldn't it be great to get derivatives without the hassle?

- Finite differences
- Automatic differentiation
- Symbolic differentiation

Idea: Take inspiration from the limit definition of the derivative:

$$\frac{\partial f}{\partial x_i}(x) = \lim_{\epsilon \rightarrow 0} \frac{f(x + \epsilon e_i) - f(x)}{\epsilon}$$

Forward difference

$$\frac{\partial f}{\partial x_i}(x) \approx \widehat{\frac{\partial f}{\partial x_i}}(x) = \frac{f(x + \epsilon e_i) - f(x)}{\epsilon}$$

for some small $\epsilon > 0$

Easy to show from Taylor's theorem that

$$\frac{\partial f}{\partial x_i}(x) = \frac{f(x + \epsilon e_i) - f(x)}{\epsilon} + \mathcal{O}(\epsilon)$$

As $\epsilon \rightarrow 0$, the error also $\rightarrow 0$ **linearly** in ϵ

Central difference

$$\frac{\partial f}{\partial x_i}(x) \approx \widehat{\frac{\partial f}{\partial x_i}}(x) = \frac{f(x + \epsilon e_i) - f(x - \epsilon e_i)}{2\epsilon}$$

for some small $\epsilon > 0$

Easy to show from Taylor's theorem that

$$\frac{\partial f}{\partial x_i}(x) = \frac{f(x + \epsilon e_i) - f(x - \epsilon e_i)}{2\epsilon} + \mathcal{O}(\epsilon^2)$$

As $\epsilon \rightarrow 0$, the error also $\rightarrow 0$ **quadratically** in ϵ

Central difference incurs 2X computations of forward difference

Finite machine precision limits max precision/benefit

Build **gradient** by FD approximation of all first partial derivatives of f :

$$\widehat{\nabla} f = \left[\frac{\widehat{\partial} f}{\partial x_1}(x) \cdots \frac{\widehat{\partial} f}{\partial x_n}(x) \right]^T$$

Build **Hessian** by FD approximation of all second partial derivatives of f :

- **If gradients are available,**

$$\widehat{\nabla^2 f} = \left[\frac{\widehat{\partial} \widehat{\nabla} f}{\partial x_1}(x) \cdots \frac{\widehat{\partial} \widehat{\nabla} f}{\partial x_n}(x) \right]$$

where the i^{th} column of $\widehat{\nabla^2 f}$ is

$$\frac{\widehat{\partial} \widehat{\nabla} f}{\partial x_i}(x) = \frac{\nabla f(x + \epsilon e_i) - \nabla f(x)}{\epsilon}$$

- **If gradients are not available,**

$$\frac{\widehat{\partial^2 f}}{\partial x_i \partial x_j}(x) = \frac{f(x + \epsilon e_i + \epsilon e_j) - f(x + \epsilon e_i) - f(x + \epsilon e_j) + f(x)}{\epsilon^2}$$

Idea: Break down code for evaluating f into a **composition of elementary arithmetic operations** over a **computation graph**, then apply the **chain rule**

Forward-mode

- Works forwards from inputs to outputs
- Fundamentally the forward mode always computes directional derivatives¹⁵
- Faster than reverse-mode when there are few inputs and lots of outputs

¹⁵See <https://autodiff.github.io/faq/>

Reverse-mode

- Works backwards from outputs to inputs
- Accumulates partial derivatives (memoization) as it goes
- Very similar flavor to dynamic programming
- Faster than forward-mode when there are lots of inputs and few outputs
 - Example: high-dimensional feature space + scalar loss
 - Common situation in optimization & machine learning
- Called **backpropagation** in the context of training neural networks¹⁶

¹⁶ML people argue about definitions, but this is generally what you will hear

Modern packages make AD **fast and user-friendly**

Well-suited for **large-scale computation**

- Efficient implementations
- Many run on CUDA graphics cards

You can **differentiate through an amazing diversity of functions!**

- Neural networks + basically anything in NumPy
 - TensorFlow
 - PyTorch
 - autograd / JAX
- Physics simulators (great for learning-based control)
 - DiffSim (Rigid body)
 - ChainQueen / DiffTaichi (Soft body)
 - PhiFlow (Fluids)
 - JAX-MD (Molecules)
 - MATch (Materials / light transport)
- Convex optimization solvers themselves (mind blown!)
 - cvxpylayers

Idea: Manipulate algebraic specification for the function f to produce new algebraic expressions for each component of the gradient

Nice for theoretical work where you really need math expressions for derivatives¹⁷

- [SymPy](#) package for Python
- [Symbolic Math Toolbox](#) for MATLAB
- [Mathematica](#)
- [Maple](#)

Not super practical when you have huge expressions and just need the numbers

¹⁷At least that's what people say...don't ask me because I rarely use symdiff...

LQR Policy Optimization

Motivation: Why use LQR policy optimization as a test case in optimization?

- Attractive theoretical properties
 - Nontrivial slope characteristics
 - Nonconvex
 - Extremely steep near stability boundary
 - Extremely flat near minimum
 - Motivates varying step size and 2nd-order methods
 - Baseline solution from control theoretic techniques
 - Validate optimization algorithms
- Practical utility
 - Fundamental problem in control theory
 - Design useful controllers automatically
- Springboard for harder problem classes, algorithms, & analysis
 - Nonquadratic costs
 - Nonlinear systems
 - Stochastic optimal control
 - Model-free reinforcement learning

Consider the **linear time-invariant (LTI) dynamical system**

$$x_{t+1} = Ax_t + Bu_t$$

- $x_t \in \mathbb{R}^n$ is the **state**
- $u_t \in \mathbb{R}^m$ is the **input**
- $A \in \mathbb{R}^{n \times n}$ is the **state-to-state** matrix
- $B \in \mathbb{R}^{n \times m}$ is the **input-to-state** matrix

The **initial state** x_0 is a random vector distributed as

$$x_0 \sim \mathcal{D}(0, X_0)$$

where $\mathcal{D}(0, X_0)$ is any **initial state distribution** with

- Mean zero
- Covariance $X_0 \succ 0$

A sensible quadratic objective

$$\underset{\{u_t\}_{t=0}^{\infty}}{\text{minimize}} \quad \mathbb{E}_{x_0} \sum_{t=0}^{\infty} \begin{bmatrix} x_t \\ u_t \end{bmatrix}^{\top} \begin{bmatrix} Q_{xx} & Q_{xu} \\ Q_{ux} & Q_{uu} \end{bmatrix} \begin{bmatrix} x_t \\ u_t \end{bmatrix}$$

where $Q \in \mathbb{S}^{(n+m) \times (n+m)}$ is the **penalty matrix**

- Q_{xx} penalizes state deviations from the origin
- Q_{uu} penalizes control effort
- Q_{xu} penalizes joint variation in state and input

Putting these elements together yields the **LQR problem**:

$$\begin{aligned} & \underset{\{u_t\}_{t=0}^{\infty}}{\text{minimize}} && \mathbb{E}_{x_0} \sum_{t=0}^{\infty} \begin{bmatrix} x_t \\ u_t \end{bmatrix}^{\top} \begin{bmatrix} Q_{xx} & Q_{xu} \\ Q_{ux} & Q_{uu} \end{bmatrix} \begin{bmatrix} x_t \\ u_t \end{bmatrix} \\ & \text{subject to} && x_{t+1} = Ax_t + Bu_t \\ & && x_0 \sim \mathcal{D}(0, X_0) \end{aligned}$$

Decision variable: $\{u_t\}_{t=0}^{\infty}$

Problem data: (A, B, Q, X_0)

Assumptions:

- (A, B) controllable
- (A, Q_{xx}) observable
- $Q \succ 0$
- $X_0 \succ 0$

Dynamic programming can be used to show that the optimal input sequence $\{u_t\}_{t=0}^{\infty}$ is generated by a **state-feedback policy**

$$u_t = Kx_t$$

which is completely specified by the **gain matrix** $K \in \mathbb{R}^{m \times n}$

Lets us replace optimization over infinite-length input sequences with optimization over finite-dimensional policies (mn parameters)

LQR problem becomes the **LQR policy optimization problem**:

$$\begin{aligned}
 &\underset{K}{\text{minimize}} && \mathbb{E}_{x_0} \sum_{t=0}^{\infty} \begin{bmatrix} x_t \\ u_t \end{bmatrix}^{\top} \begin{bmatrix} Q_{xx} & Q_{xu} \\ Q_{ux} & Q_{uu} \end{bmatrix} \begin{bmatrix} x_t \\ u_t \end{bmatrix} \\
 &\text{subject to} && x_{t+1} = Ax_t + Bu_t \\
 &&& u_t = Kx_t \\
 &&& x_0 \sim \mathcal{D}(0, X_0)
 \end{aligned}$$

Decision variable: K

Eliminate constraints to obtain the equivalent problem

$$\underset{K}{\text{minimize}} \quad C(K) = \begin{cases} \text{trace}(P_K X_0) & \text{if } \rho(A_K) < 1 \\ \infty & \text{else} \end{cases}$$

where P_K is the solution to the **discrete-time Lyapunov equation (DLYAP)**

$$P_K = A_K^T P_K A_K + Q_K$$

equivalently

$$\text{vec}(P_K) = (I - A_K^T \otimes A_K^T)^{-1} \text{vec}(Q_K)$$

where

$$A_K = A + BK$$

$$Q_K = \begin{bmatrix} I \\ K \end{bmatrix}^T Q \begin{bmatrix} I \\ K \end{bmatrix}$$

Alternately, we have the equivalent problem

$$\underset{K}{\text{minimize}} \quad C(K) = \begin{cases} \text{trace}(X_K Q_K) & \text{if } \rho(A_K) < 1 \\ \infty & \text{else} \end{cases}$$

where X_K is the solution to the DLYAP

$$X_K = A_K X_K A_K^\top + X_0$$

equivalently

$$\text{vec}(X_K) = (I - A_K \otimes A_K)^{-1} \text{vec}(X_0)$$

See [►► Appendix](#) for more context & interpretation of P_K and X_K

The unique global minimizer is

$$K^* = -(Q_{uu} + B^\top P B)^{-1} (Q_{ux} + B^\top P A)$$

where P solves the **algebraic Riccati equation (ARE)**

$$P = Q_{xx} + A^\top P A - (Q_{xu} + A^\top P B)(Q_{uu} + B^\top P B)^{-1} (Q_{ux} + B^\top P A)$$

Note: The right-hand side is the **Schur complement** of

$$H = Q + \begin{bmatrix} A & B \end{bmatrix}^\top P \begin{bmatrix} A & B \end{bmatrix}$$

Solve ARE using linear algebraic or **dynamic programming** techniques

- Requires knowledge of problem data (A, B, Q)
- (A, B, Q) not exposed explicitly in cost, gradient, Hessian
 - Policy optimization is a harder problem than model-based LQR!
 - In RL, we don't know (A, B) but we can estimate cost, gradient, Hessian from observed data

Under the given assumptions, the analysis of [26] shows

- f is a real analytic function over its domain
 - Implies f has continuous derivatives of arbitrary order over its domain
- f is coercive
 - $f(K) \rightarrow \infty$ as $\rho(A + BK) \rightarrow 1$ or $\|K\| \rightarrow \infty$
- f has compact sublevel sets
- f is gradient dominated (PL)
- f has a unique stationary point and global minimizer K^*
- The Hessian $\nabla^2 f$ is positive definite at K^* (SOSC)

Takeaway: Descent methods will find the solution to the LQR policy optimization problem

Next: Let's get derivatives and verify the SOSC for LQR

We can obtain a closed-form expression for the **policy gradient** [11, 12]

$$\nabla C(K) = 2(H_{uu}K + H_{ux})X_K$$

and the action of the **policy Hessian** on a matrix $E \in \mathbb{R}^{m \times n}$ [26]

$$\nabla^2 C(K)[E, E] = 2\langle (H_{uu}E + 2B^\top P'_K[E]A_K)X_K, E \rangle$$

where

$$H = H_K = Q + \begin{bmatrix} A & B \end{bmatrix}^\top P_K \begin{bmatrix} A & B \end{bmatrix}$$

and $P'_K[E]$ is explained on the next slide

$P'_K[E]$ denotes the action of the differential of the map $K \mapsto P_K$ on E

$P'_K[E]$ can be evaluated explicitly as the solution to the DLYAP¹⁸

$$P'_K[E] = A_K^\top P'_K[E] A_K + S_K$$

where

$$\begin{aligned} S_K &= L_K + L_K^\top \\ L_K &= E^\top (B^\top P_K A_K + Q_{ux} + Q_{uu} K) \\ &= E^\top (H_{ux} + H_{uu} K) \end{aligned}$$

¹⁸This is easily obtained by differentiating the DLYAP for P_K with respect to K

Note: We did not get the Hessian $\nabla^2 C(K)$ itself

Rather we got the “action” or “quadratic form” of the 4th-order tensor $\nabla^2 C(K)$ “pre- and post-multiplied” by a matrix $E \in \mathbb{R}^{m \times n}$

Results in a scalar i.e. $\nabla^2 C(K)[E, E] \in \mathbb{R}$

Check positive definiteness of $\nabla^2 C(K)$ by checking positivity of $\nabla^2 C(K)[E, E]$ for all matrices E .

Goal: Verify the SOSC

Note: f is twice continuously differentiable on its domain

Seek a stationary point:

$$\nabla C(K) = 2(H_{K,uu}K + H_{K,ux})X_K = 0$$

Since $X_K \succ X_0 \succ 0$ by assumption, this implies

$$H_{K,uu}K + H_{K,ux} = 0$$

It can be shown that this equation has a **unique solution** K^*

At K^* we have

$$\begin{aligned} L_{K^*} &= E^\top (H_{K^*,ux} + H_{K^*,uu} K^*) = 0 \\ P'_{K^*}[E] &= A_{K^*}^\top P'_{K^*}[E] A_{K^*} + L_{K^*} + L_{K^*}^\top \\ &= A_{K^*}^\top P'_{K^*}[E] A_{K^*} \end{aligned}$$

which only has the trivial solution

$$P'_{K^*}[E] = 0$$

and the action-of-the-Hessian becomes

$$\begin{aligned} \nabla^2 C(K^*)[E, E] &= 2 \langle H_{K^*,uu} E X_{K^*}, E \rangle \\ &= 2 \operatorname{trace} \left((E^\top H_{K^*,uu} E) X_{K^*} \right) > 0 \end{aligned}$$

Thus $C(K)$ satisfies the SOSC at K^* , so K^* is the unique global minimizer of $C(K)$.

Wouldn't it be great to get policy derivatives without the hassle?

Notice that $C(K)$ can be expressed in terms of $x = \text{vec}(K)$ as $f(x)$

Notice that $f(x)$ can be expressed with linear-algebraic operations

- Transpose
- Vectorization
- Matricization
- Trace
- Matrix addition
- Matrix multiplication
- Matrix inverse

Use automatic differentiation to compute gradients and Hessians!

Demos

Get the Python code from the [GitHub repository](#)

Demo implements the following optimization algorithms:

Accelerated gradient methods

- Gradient descent (constant step size)
- Polyak momentum
- Nesterov acceleration
- Relativistic acceleration
- RMSprop
- Adam

Line search

- Gradient descent
- Newton (with eigenvalue modification)
- BFGS
- DFP
- SR1

Demo implements the following optimization algorithms: (cont.)

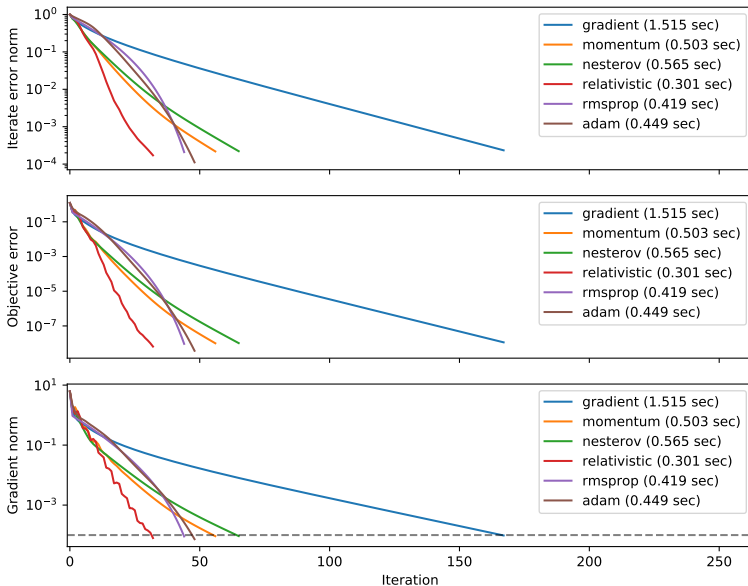
Trust-region

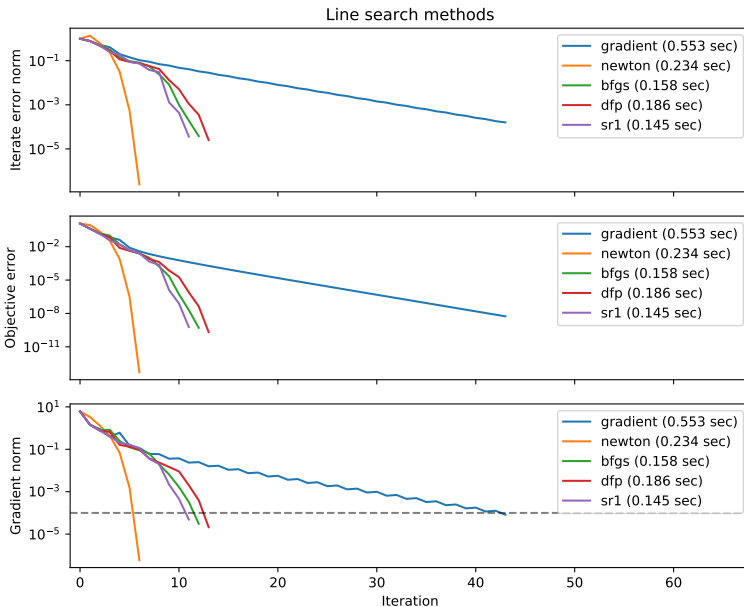
- Dogleg
- 2D-subspace minimization
- Steihaug-CG

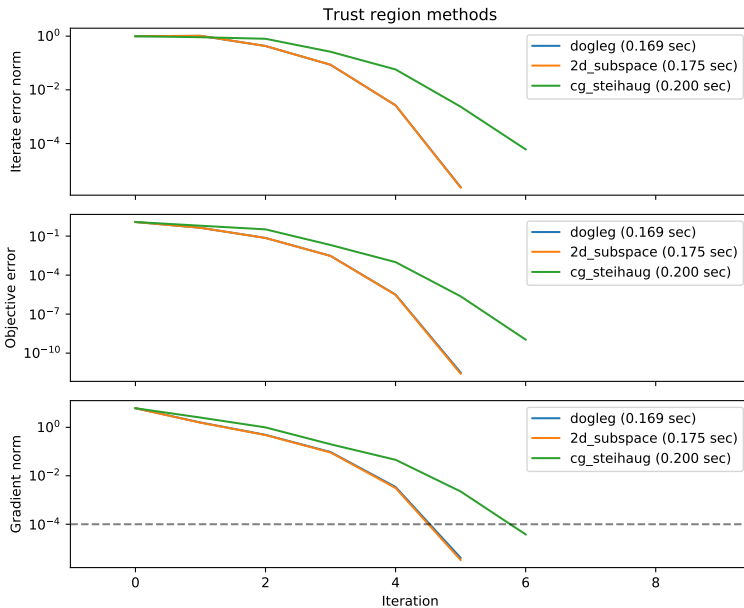
Control-theoretic methods

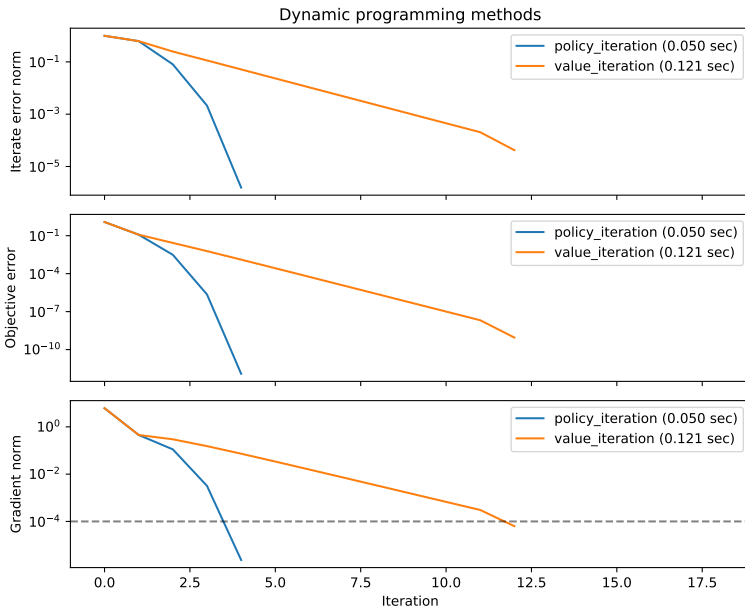
- Policy iteration
 - ⇔ Newton method in value space (P)
 - ⇔ Quasi-Newton method in policy space (K)
- Value iteration
- Riccati equation solver

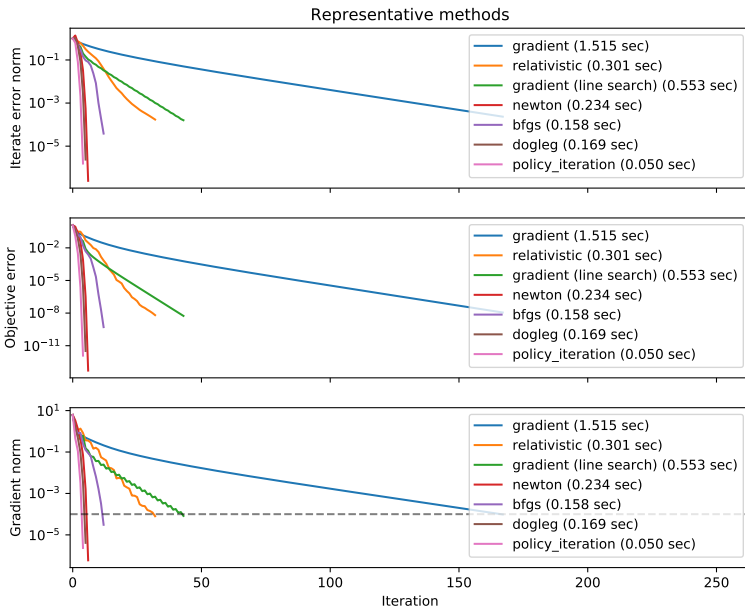
Gradient methods











- [1] Jorge Nocedal and Stephen Wright.
Numerical optimization.
Springer Science & Business Media, 2006.
- [2] Stephen Boyd, Stephen P Boyd, and Lieven Vandenberghe.
Convex optimization.
Cambridge university press, 2004.
- [3] Michael W Botsko.
A first derivative test for functions of several variables.
The American Mathematical Monthly, 93(7):558–561, 1986.
- [4] Rafaël Correa and Jean-Baptiste Hiriart-Urruty.
A first order sufficient condition for optimality in nonsmooth optimization.
Mathematische Nachrichten, 144(1):309–319, 1989.

- [5] Adi Ben-Israel and Bertram Mond.
What is invexity?
The ANZIAM Journal, 28(1):1–9, 1986.
- [6] Moritz Hardt, Tengyu Ma, and Benjamin Recht.
Gradient descent learns linear dynamical systems.
arXiv preprint arXiv:1609.05191, 2016.
- [7] Oliver Hinder, Aaron Sidford, and Nimit Sohoni.
Near-optimal methods for minimizing star-convex functions and beyond.
In *Conference on Learning Theory*, pages 1894–1938. PMLR, 2020.
- [8] Laurent Lessard, Benjamin Recht, and Andrew Packard.
Analysis and design of optimization algorithms via integral quadratic constraints.
SIAM Journal on Optimization, 26(1):57–95, 2016.

- [9] Mahyar Fazlyab, Alejandro Ribeiro, Manfred Morari, and Victor M Preciado.
Analysis of optimization algorithms via integral quadratic constraints: Nonstrongly convex problems.
SIAM Journal on Optimization, 28(3):2654–2689, 2018.
- [10] Stephen Boyd, Laurent El Ghaoui, Eric Feron, and Venkataramanan Balakrishnan.
Linear matrix inequalities in system and control theory, volume 15.
SIAM, 1994.
- [11] Maryam Fazel, Rong Ge, Sham Kakade, and Mehran Mesbahi.
Global convergence of policy gradient methods for the linear quadratic regulator.
In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 1467–1476. PMLR, 10–15 Jul 2018.

- [12] Benjamin Gravell, Peyman Mohajerin Esfahani, and Tyler Summers.
Learning robust control for LQR systems with multiplicative noise
via policy gradient.
CoRR, 2019.
- [13] Jason D Lee, Max Simchowitz, Michael I Jordan, and Benjamin
Recht.
Gradient descent only converges to minimizers.
In *Conference on learning theory*, pages 1246–1257. PMLR, 2016.
- [14] Yurii Nesterov.
Introductory lectures on convex programming volume i: Basic
course.
Lecture notes, 3(4):5, 1998.

- [15] Boris T Polyak.
Some methods of speeding up the convergence of iteration methods.

Ussr computational mathematics and mathematical physics,
4(5):1–17, 1964.

- [16] Yurii E Nesterov.
A method for solving the convex programming problem with
convergence rate $O(1/k^2)$.
In *Dokl. akad. nauk Sssr*, volume 269, pages 543–547, 1983.

- [17] Sébastien Bubeck et al.
Convex optimization: Algorithms and complexity.
Foundations and Trends® in Machine Learning, 8(3-4):231–357,
2015.

- [18] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton.
On the importance of initialization and momentum in deep learning.
In Sanjoy Dasgupta and David McAllester, editors, *Proceedings of the 30th International Conference on Machine Learning*, volume 28 of *Proceedings of Machine Learning Research*, pages 1139–1147, Atlanta, Georgia, USA, 17–19 Jun 2013. PMLR.

- [19] Guilherme França, Jeremias Sulam, Daniel P Robinson, and René Vidal.
Conformal symplectic and relativistic optimization.
Journal of Statistical Mechanics: Theory and Experiment, 2020(12):124008, 2020.

- [20] Peiyuan Zhang, Antonio Orvieto, Hadi Daneshmand, Thomas Hofmann, and Roy Smith.
Revisiting the role of euler numerical integration on acceleration and stability in convex optimization.
arXiv preprint arXiv:2102.11537, 2021.

- [21] John Duchi, Elad Hazan, and Yoram Singer.
Adaptive subgradient methods for online learning and stochastic optimization.
Journal of Machine Learning Research, 12(61):2121–2159, 2011.
- [22] Diederik P. Kingma and Jimmy Ba.
Adam: A method for stochastic optimization.
In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.
- [23] T. Steihaug.
The conjugate gradient method and trust regions in large scale optimization.
SIAM Journal on Numerical Analysis, 20:626–637, 1983.
- [24] Peter J. Huber.
Robust Estimation of a Location Parameter.
The Annals of Mathematical Statistics, 35(1):73 – 101, 1964.

- [25] Neal Parikh and Stephen Boyd.
Proximal algorithms.
Foundations and Trends in optimization, 1(3):127–239, 2014.
- [26] Jingjing Bu, Afshin Mesbahi, Maryam Fazel, and Mehran Mesbahi.
LQR through the lens of first order methods: Discrete-time case.
arXiv preprint arXiv:1907.08921, 2019.

Appendix: First-order sufficient conditions

Fix $x \in \mathcal{N} \setminus \{x^*\}$ arbitrarily.

Define $F : [0, 1] \rightarrow \mathbb{R}$ as

$$F(t) = f(x^* + t(x - x^*))$$

so F is continuous on $[0, 1]$ and differentiable on $(0, 1)$ with

$$F'(t) = \nabla f(x^* + t(x - x^*))^\top (x - x^*).$$

Apply the Mean-Value Theorem to F , so there exists $t_0 \in (0, 1)$ with

$$\begin{aligned} F'(t_0) &= F(1) - F(0) \\ \Leftrightarrow \quad \nabla f(x^* + t_0(x - x^*))^\top (x - x^*) &= f(x) - f(x^*). \end{aligned} \tag{2}$$

Let $y = x^* + t_0(x - x^*)$. Since $t \in (0, 1)$ we know

$$0 < \|y - x^*\| < \|x - x^*\|.$$

Thus, because \mathcal{N} is a neighborhood of x^* , conclude $y \in \mathcal{N} \setminus \{x^*\}$.

Now apply the third assumption of FOSC with y in place of x to (2) and obtain

$$f(x) - f(x^*) = \nabla f(y)^\top (y - x^*) \cdot \frac{1}{t_0} \geq 0$$

Since $x \in \mathcal{N} \setminus \{x^*\}$ was arbitrary, we proved

$$f(x) \geq f(x^*) \text{ for all } x \in \mathcal{N} \setminus \{x^*\}.$$

This shows x^* minimizes f on \mathcal{N} , i.e. x^* is a weak local minimizer. ■

►► Back to statement

Fix $x \in \mathcal{N} \setminus \{x^*\}$ arbitrarily. First, observe that

$$\nabla f(x)^\top (x - x^*) > 0$$

implies that $\nabla f(x) \neq 0$ since $x - x^* \neq 0$. Therefore, the only possible stationary point in \mathcal{N} is x^* .

By the FONC, this also shows x^* is the only possible local minimizer in \mathcal{N} i.e. if x^* is a local minimizer it is also isolated.

Apply the same construction of $F : [0, 1] \rightarrow \mathbb{R}$ and $y = x^* + t_0(x - x^*)$ as in the proof of FOSC.

Now apply the third assumption of SFOSC with y in place of x to (2) and obtain

$$f(x) - f(x^*) = \nabla f(y)^\top (y - x^*) \cdot \frac{1}{t_0} > 0$$

Since $x \in \mathcal{N} \setminus \{x^*\}$ was arbitrary, we proved

$$f(x) > f(x^*) \text{ for all } x \in \mathcal{N} \setminus \{x^*\}.$$

This shows x^* is an (isolated) local minimizer and minimizes f on \mathcal{N} . ■

►► Back to statement

Exercise: Show that any continuously differentiable convex function f with a stationary point x^* satisfies the global FOSC.

Solution:

Since f is convex and continuously differentiable, for any $x, y \in \mathbb{R}^n$,

$$\nabla f(x)^\top (x - y) \geq f(x) - f(y).$$

Since f is convex, any stationary point x^* is also a global minimizer. Taking $y = x^*$ shows that for any $x \in \mathbb{R}^n$

$$\nabla f(x)^\top (x - x^*) \geq f(x) - f(x^*) \geq 0.$$

Exercise: Show that any continuously differentiable quasiconvex function f with a global minimum x^* satisfies the global FOSC.

Solution:

Since f is quasiconvex and continuously differentiable, for any $x, y \in \mathbb{R}^n$ such that $f(x) \geq f(y)$

$$\nabla f(x)^\top (x - y) \geq 0.$$

Taking $y = x^*$ shows for any $x \in \mathbb{R}^n$, so $f(x) \geq f(x^*)$, that

$$\nabla f(x)^\top (x - x^*) \geq 0.$$

Exercise: Show that any continuously differentiable quasr convex function f with a global minimum x^* satisfies the global FOSC.

Solution:

Since f is quasr convex and continuously differentiable, for any $x \in \mathbb{R}^n$

$$\nabla f(x)^\top (x - x^*) \geq \gamma[f(x) - f(x^*)] \geq 0,$$

which follows because x^* is a global min and $\gamma > 0$.

Exercise: Show that any continuously differentiable RSI function f satisfies the global FOSC.

Solution:

Since f is RSI and continuously differentiable, for any $x \in \mathbb{R}^n$

$$\nabla f(x)^\top (x - x^*) \geq \mu \|x - x^*\| \geq 0. \quad (\text{because } \mu > 0 \text{ and } \|x - x^*\| \geq 0)$$

Exercise: Show that the strongly convex quadratic

$$f(x) = \frac{1}{2}x^{\top}Ax - b^{\top}x + c$$

where $A \succ 0$ satisfies the global SFOSC.

Solution:

Clearly f is continuously differentiable everywhere. By assumption $A \succ 0$, so A^{-1} is well-defined and $A^{-1} \succ 0$. Also, the gradient is

$$\nabla f(x) = Ax + b$$

so the only stationary point is $x^* = -A^{-1}b$.

Therefore, for any $x \neq x^* \Leftrightarrow Ax + b \neq 0$ we have

$$\begin{aligned}\nabla f(x)^{\top}(x - x^*) &= (Ax + b)^{\top}(x - (-A^{-1}b)) \\ &= (Ax + b)^{\top}A^{-1}(Ax + b) > 0.\end{aligned}$$



Exercise: Show that any continuously differentiable strictly convex function f with a stationary point x^* satisfies the global SFOSC.

Solution:

Since f is strictly convex and continuously differentiable, for any $x \neq y \in \mathbb{R}^n$,

$$\nabla f(x)^\top (x - y) > f(x) - f(y).$$

Since f is strictly convex, the stationary point x^* is also the unique global minimizer.¹⁹

Taking $y = x^*$ shows that for any $x \neq x^* \in \mathbb{R}^n$

$$\nabla f(x)^\top (x - x^*) > f(x) - f(x^*) > 0.$$

¹⁹See the answer at <https://math.stackexchange.com/questions/337090/if-f-is-strictly-convex-in-a-convex-set-show-it-has-no-more-than-1-minimum> or the notes at <https://ai.stanford.edu/~gwthomas/notes/convexity.pdf>

Exercise: Show that any continuously differentiable strictly quasiconvex function f with a local minimum x^* satisfies the global SFOSC.

Solution:

Since f is strictly quasiconvex and continuously differentiable, for any $x \neq y \in \mathbb{R}^n$ such that $f(x) > f(y)$

$$\nabla f(x)^\top (x - y) > 0.$$

Since f is strictly quasiconvex, the local minimum x^* is also the unique global minimizer.²⁰

Taking $y = x^*$ shows that for any $x \neq x^* \in \mathbb{R}^n$ (so $f(x) > f(x^*)$)

$$\nabla f(x)^\top (x - x^*) > 0.$$

²⁰See the answer of Michael Grant at

<https://math.stackexchange.com/questions/2042356/quasiconvex-fucntions-local-minimum-is-equal-to-global-minimum>

Appendix: Sector IQCs and gradient descent for $\text{RSI} + \text{RLG}$ functions

Sufficient pre-condition: SC + LG implies sector IQC

If f is m -strongly convex with L -Lipschitz gradient, then f satisfies IQC with m, L .

Proof: See [8]

Sector-bound IQC implies RSI + RLG

The sector-bound IQC

$$\begin{bmatrix} x - x^* \\ \nabla f(x) \end{bmatrix}^\top \begin{bmatrix} -2mL & m + L \\ m + L & -2 \end{bmatrix} \begin{bmatrix} x - x^* \\ \nabla f(x) \end{bmatrix} \geq 0$$

for any $x \in \mathbb{R}^n$ with constants $m > 0, L > 0$ implies

- 1 Restricted secant inequality (RSI) with parameter $\frac{mL}{m+L}$
- 2 Restricted Lipschitz gradient (RLG) with parameter $m + L$

Proof: On next 2 slides

Proof: Expanding the sector IQC,

$$\nabla f(x)^\top (x - x^*) \geq \frac{mL}{m+L} \|x - x^*\|^2 + \frac{1}{m+L} \|\nabla f(x)\|^2$$

For claim 1,

$$\nabla f(x)^\top (x - x^*) \geq \frac{mL}{m+L} \|x - x^*\|^2$$

so f is RSI with parameter $\frac{mL}{m+L}$.

For claim 2, using Cauchy-Schwarz

$$\|\nabla f(x)\| \|x - x^*\| \geq \nabla f(x)^\top (x - x^*) \geq \frac{1}{m+L} \|\nabla f(x)\|^2$$

This holds trivially if $x = x^*$, so for $x \neq x^*$ divide through by $\|\nabla f(x)\|$ and get

$$\|x - x^*\| \geq \frac{1}{m+L} \|\nabla f(x)\|$$

equivalently

$$\|\nabla f(x)\| \leq (m+L) \|x - x^*\|$$

so f is RLG with parameter $m+L$.

» Back to statement

Question: Why care about RSI + RLG?

Answer: Lets us quickly prove linear convergence of gradient descent

Gradient descent converges for RSI + RLG objectives

Suppose f satisfies

$$\nabla f(x)^\top (x - x^*) \geq R \|x - x^*\|^2, \quad R > 0, \text{ for any } x \in \mathbb{R}^n, \quad (\text{RSI})$$

$$\|\nabla f(x)\| \leq L \|x - x^*\|, \quad L > 0, \text{ for any } x \in \mathbb{R}^n. \quad (\text{RLG})$$

Then the iterates x_k of gradient descent

$$x_{k+1} = x_k - \alpha \nabla f(x_k)$$

with step size $\alpha = \frac{R}{L^2}$ converge to x^* at the linear rate

$$\frac{\|x_{k+1} - x^*\|}{\|x_k - x^*\|} \leq 1 - \left(\frac{R}{L}\right)^2 \text{ for all } k = 0, 1, \dots$$

Proof:

First, note that, by RSI and using Cauchy-Schwarz,

$$\|\nabla f(x)\| \|x - x^*\| \geq \nabla f(x)^\top (x - x^*) \geq R \|x - x^*\|^2$$

so together with RLG

$$R \|x - x^*\| \leq \|\nabla f(x)\| \leq L \|x - x^*\|$$

which shows that $0 < R \leq L$ is implicit in the assumption.

Proof: (cont.)

Using the gradient descent update, we have

$$\begin{aligned}
 \|x_{k+1} - x^*\| &= \|(x_k - x^*) - \alpha \nabla f(x_k)\| \\
 &= \|x_k - x^*\|^2 - 2\alpha \nabla f(x_k)^\top (x_k - x^*) + \alpha^2 \|\nabla f(x_k)\|^2 \\
 &\leq \|x_k - x^*\|^2 - 2\alpha R \|x_k - x^*\|^2 + \alpha^2 L^2 \|x_k - x^*\|^2 \\
 &\hspace{15em} \text{(by RSI and RLG)} \\
 &= (1 - 2\alpha R + \alpha^2 L^2) \|x_k - x^*\|^2
 \end{aligned}$$

So long as the multiplier $1 - 2\alpha R + \alpha^2 L^2 \in [0, 1)$ we obtain convergence.

Proof: (cont.)

Define the scalar function

$$\phi(\alpha) = 1 - 2\alpha R + \alpha^2 L^2$$

and find its derivatives

$$\phi'(\alpha) = -2R + 2\alpha L^2$$

$$\phi''(\alpha) = 2L^2 > 0$$

Thus ϕ is convex quadratic in α , so the minimum is when

$$\phi'(\alpha) = -2R + 2\alpha L^2 = 0$$

$$\Leftrightarrow \quad \alpha = \frac{R}{L^2}$$

Proof: (cont.)

Using the optimal step size,

$$\begin{aligned} 1 - 2\alpha_k R + \alpha_k^2 L^2 &= 1 - 2\frac{R}{L^2} R + \left(\frac{R}{L^2}\right) L^2 \\ &= 1 - \left(\frac{R}{L}\right)^2 \end{aligned}$$

Earlier we established that $R/L \in (0, 1]$,
so indeed the multiplier $1 - \left(\frac{R}{L}\right)^2 \in [0, 1)$, completing the proof.

►► Back to statement

Appendix: LQR policy optimization

The system is closed-loop when the inputs u_t are determined by the state-feedback gain matrix K as

$$u_t = Kx_t$$

In this case, the system is autonomous and

$$\begin{aligned}x_{t+1} &= Ax_t + Bu_t \\&= Ax_t + BKx_t \\&= A_K x_t\end{aligned}$$

The closed-loop matrix A_K is called **Schur stable** if $\rho(A_K) < 1$ where $\rho(A_K) = \max_i |\lambda_i(A_K)|$ is the **spectral radius** of A_K where $\{\lambda_i(A_K)\}$ is the set of eigenvalues of A_K

Unroll closed-loop dynamics back to $t = 0$ using matrix powers:

$$\begin{aligned}x_t &= A_K x_{t-1} \\&= A_K^2 x_{t-2} \\&\vdots \\&= A_K^t x_0\end{aligned}$$

Also, the stage cost can be simplified in closed-loop as

$$\begin{aligned}\begin{bmatrix} x_t \\ u_t \end{bmatrix}^\top \begin{bmatrix} Q_{xx} & Q_{xu} \\ Q_{ux} & Q_{uu} \end{bmatrix} \begin{bmatrix} x_t \\ u_t \end{bmatrix} &= \begin{bmatrix} x_t \\ Kx_t \end{bmatrix}^\top \begin{bmatrix} Q_{xx} & Q_{xu} \\ Q_{ux} & Q_{uu} \end{bmatrix} \begin{bmatrix} x_t \\ Kx_t \end{bmatrix} \\&= x_t^\top \begin{bmatrix} I \\ K \end{bmatrix}^\top \begin{bmatrix} Q_{xx} & Q_{xu} \\ Q_{ux} & Q_{uu} \end{bmatrix} \begin{bmatrix} I \\ K \end{bmatrix} x_t \\&= x_t^\top Q_K x_t\end{aligned}$$

P_K characterizes the **accumulated state and input costs** as

$$P_K = \sum_{t=0}^{\infty} (A_K^t)^{\top} Q_K (A_K^t)$$

and

$$\begin{aligned} \mathbb{E}_{x_0} \sum_{t=0}^{\infty} \begin{bmatrix} x_t \\ u_t \end{bmatrix}^{\top} \begin{bmatrix} Q_{xx} & Q_{xu} \\ Q_{ux} & Q_{uu} \end{bmatrix} \begin{bmatrix} x_t \\ u_t \end{bmatrix} &= \mathbb{E}_{x_0} \left[\sum_{t=0}^{\infty} x_0^{\top} (A_K^t)^{\top} Q_K (A_K^t) x_0 \right] \\ &= \mathbb{E}_{x_0} \left[x_0^{\top} \left(\sum_{t=0}^{\infty} (A_K^t)^{\top} Q_K (A_K^t) \right) x_0 \right] \\ &= \mathbb{E}_{x_0} [x_0^{\top} P_K x_0] \\ &= \mathbb{E}_{x_0} [\text{trace}(x_0^{\top} P_K x_0)] \\ &= \mathbb{E}_{x_0} [\text{trace}(P_K x_0 x_0^{\top})] \\ &= \text{trace}(P_K \mathbb{E}_{x_0} [x_0 x_0^{\top}]) \\ &= \text{trace}(P_K X_0) \end{aligned}$$

X_K characterizes the **accumulated state covariances** as

$$X_K = \sum_{t=0}^{\infty} (A_K^t) X_0 (A_K^t)^{\top}$$

and

$$\begin{aligned} \mathbb{E}_{x_0} \sum_{t=0}^{\infty} x_t x_t^{\top} &= \mathbb{E}_{x_0} \left[\sum_{t=0}^{\infty} (A_K^t) x_0 x_0^{\top} (A_K^t)^{\top} \right] \\ &= \sum_{t=0}^{\infty} (A_K^t) \mathbb{E}_{x_0} [x_0 x_0^{\top}] (A_K^t)^{\top} \\ &= \sum_{t=0}^{\infty} (A_K^t) X_0 (A_K^t)^{\top} \\ &= X_K \end{aligned}$$

X_K also characterizes the **accumulated state and input costs** as

$$\begin{aligned}\mathbb{E}_{x_0} \sum_{t=0}^{\infty} \begin{bmatrix} x_t \\ u_t \end{bmatrix}^{\top} \begin{bmatrix} Q_{xx} & Q_{xu} \\ Q_{ux} & Q_{uu} \end{bmatrix} \begin{bmatrix} x_t \\ u_t \end{bmatrix} &= \mathbb{E}_{x_0} \left[\sum_{t=0}^{\infty} x_t^{\top} Q_K x_t \right] \\ &= \mathbb{E}_{x_0} \left[\sum_{t=0}^{\infty} \text{trace} (x_t^{\top} Q_K x_t) \right] \\ &= \mathbb{E}_{x_0} \left[\sum_{t=0}^{\infty} \text{trace} (Q_K x_t x_t^{\top}) \right] \\ &= \text{trace} \left(Q_K \mathbb{E}_{x_0} \left[\sum_{t=0}^{\infty} x_t x_t^{\top} \right] \right) \\ &= \text{trace}(Q_K X_K) \\ &= \text{trace}(X_K Q_K)\end{aligned}$$

P_K and X_K are **dual** to each other

- Observe the DLYAPs
- Observe the aggregate cost equivalences we derived
- In a sense they encode the same information