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

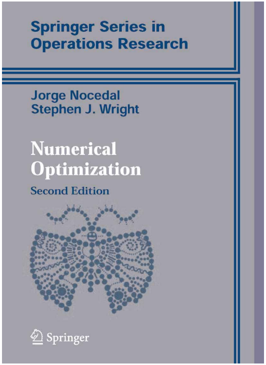
1. Unconstrained optimization preliminaries
2. Unconstrained optimization algorithms
3. Demo on LQR policy optimization

I will cover material from

- Boyd & Vandenberghe's textbook "Convex Optimization" [1]
- Nocedal & Wright's textbook "Numerical Optimization" [2]
- Assorted papers

I can't cover everything today

- See the extended notes
- Take MATH 6321



Intro

So far you have focused on

- Modeling & formulating optimization problems
- Studying & certifying convexity properties

Convexity **decouples** modeling & algorithm design...

- You can *use* convex optimization without *knowing* how solvers work under the hood
  - e.g. CVX, YALMIP

...but knowing how solvers work is important too!

- **Discern** which solver is right for your problem
- **Choose** solver settings in a principled way
- **Predict** how long problems will take to solve
- **Invent** new algorithms

Bonus - algorithms work on some nonconvex problems too!

Unconstrained Optimization Preliminaries

Unconstrained Optimization Problem (hard version)

Find a point x^* that solves

$$x^* = \operatorname*{argmin}_x f(x)$$

where $f : \mathbb{R}^n \to \mathbb{R}$ 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 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

What are we trying to do?

Unconstrained Optimization Problem (easy version)

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

"Smooth"

- $\nabla f(x)$ exists
- $\nabla f(x)$ continuous
- $\nabla^2 f(x)$ exists
- $\nabla^2 f(x)$ continuous

$\nabla f(x)$ is the **gradient**,
$\nabla^2 f(x)$ is the **Hessian**

"Local minimizer"

There exists a nonempty open ball $\mathcal{N} = \{x : \|x - x^*\| < \varepsilon\}$ for which

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

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

Answer: Check optimality conditions.

First-order optimality conditions

First-order necessary condition (FONC) (Theorem 2.2 of [2])

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 [2]

Most optimization algorithms **search for a stationary point**

- FONC $\rightarrow$ only stationary points can be local minimizers

First-order optimality conditions

Global differentiable strict first-order sufficient condition (GD-SFOSC)

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

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

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

Proof: See extended version of these lecture notes.

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

Second-order optimality conditions

UT DALLAS

Second-order necessary condition (SONC) (Theorem 2.3 of [2])

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 [2]

Second-order sufficient condition (SOSC) (Theorem 2.4 of [2])

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 [2]

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-SFOSC

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

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

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

- As you are very familiar with by now!

Invexity

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 [4])

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

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

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

Characterization of invexity (Theorem 1 in [4])

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

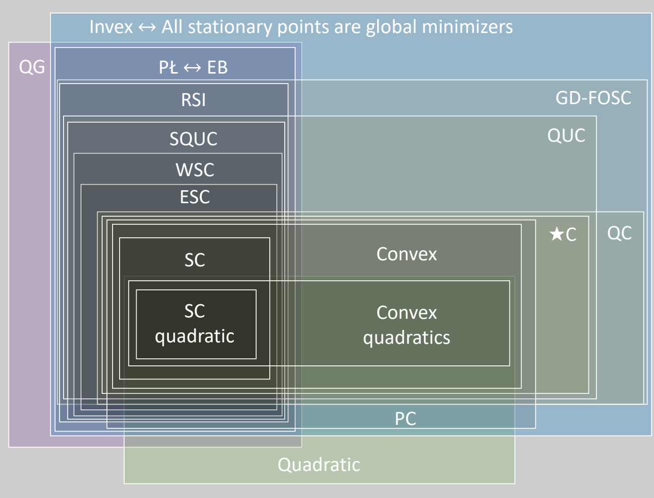


Figure 1: Slope condition nesting diagram when f is continuously differentiable and uniquely minimized. *Caveat emptor - check your assumptions and the literature before claiming an implication / non-implication.*

The zoo of slope conditions

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 [5]
Invex	Rosenbrock function
Polyak-Łojasiewicz inequality	Infinite-horizon LQR cost as function of gain matrix [6] (w/ multiplicative noise [7])
Weakly quasiconvex	"Idealized risk" in learning linear systems [8]

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
- If f continuously differentiable and invex or GD-SFOSC, then x^* is a global minimizer of f, and we solved the "hard" problem

Unconstrained Optimization Algorithms

What are we *really* trying to do?

Nesterov's thought experiment (Chapter 1.1.2 of [9])

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 **have 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**
    - The faster it solves, the better
    - The less sensitive it is to numerical errors, the better

Algorithm 1 Generic descent

Input: Initial guess x_0

while $\|\nabla f(x_k)\| > \varepsilon$ **do**

$x_{k+1} = x_k + \mathsf{update}(f(x_k), \nabla f(x_k), \nabla^2 f(x_k); \mathsf{parameters})$

$k \leftarrow k+1$

Output: $x_k \approx x^*$

Iterative descent

Each iteration is designed to decrease the function value

Oracle model

Only require evaluations of $f(x)$, $\nabla f(x)$, $\nabla^2 f(x)$ at particular points x

- No explicit knowledge of the entire objective function f

Broad categories

Categories

1. Line-search methods
2. Trust-region methods
3. Accelerated gradient methods

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

Line Search Methods

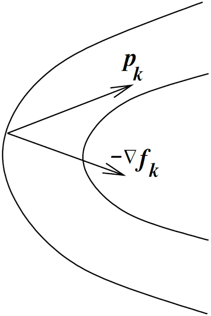
The **line-search method** uses updates of the form

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

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 [2]
- $p_k \in \mathbb{R}^n$ 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

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

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

Descent directions

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

From Taylor's theorem we have a first-order expansion

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

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

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

for all $\alpha > 0$ sufficiently small

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

Search directions

Use search directions of the form:

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

$$\begin{aligned} \textbf{Gradient method} \quad & B_k = I \\ \textbf{Newton method} \quad & B_k = \nabla^2 f_k \\ \textbf{Quasi-Newton method} \quad & B_k \approx \nabla^2 f_k \end{aligned}$$

Each of these p_k is a descent direction under mild assumptions

Motivation 1

Minimize a first-order Taylor series model of f around x_k on a trust region of radius $\|\nabla f(x)\|$

$$\begin{aligned} \underset{p}{\text{minimize}} \quad & f_k + \nabla f_k^\intercal p \qquad \approx f(x_k + p) \\ \text{subject to} \quad & \|p\| \leq \|\nabla f_k\| \end{aligned}$$

Solution is precisely the negative gradient $p_k = -\nabla f_k$

Motivation 2

$-\nabla f_k$ is a descent direction away from stationary points

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

Then

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

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

Motivation 1

Minimize a second-order Taylor series model of f around x_k

$$\underset{p}{\text{minimize}} \quad f_k + \nabla f_k^\intercal p + \frac{1}{2} p^\intercal \nabla^2 f_k p \qquad \approx f(x_k + p)$$

Solution is precisely the Newton direction $p_k = -\nabla^2 f_k^{-1} \nabla f_k$

Motivation 2

$-\nabla^2 f_k^{-1} \nabla f_k$ is a descent direction in regions of positive curvature

Assume $\nabla f_k \neq 0$. Also assume $\nabla^2 f_k \succ 0$ due to either

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

Then

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

Motivation 1

Minimize an approximate second-order Taylor series model of f around x_k

$$\underset{p}{\text{minimize}} \quad f_k + \nabla f_k^\intercal p + \frac{1}{2} p^\intercal B_k p \qquad \approx f(x_k + p)$$

Solution is precisely the quasi-Newton direction $p_k = -B_k^{-1} \nabla f_k$

Motivation 2

$-B_k^{-1} \nabla f_k$ is a descent direction for positive definite B_k

Assume $\nabla f_k \neq 0$ and choose $B_k \succ 0$

Then

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

Step size selection

Idea: Use a **constant step size**

$$\alpha_k = \alpha$$

- Advantage: Simple to implement
- Advantage: Simple to analyze
- 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
- Not covered in [2], we'll skip these for now

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 later
- Not covered in [2], we'll skip these for now
- See the survey

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

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

- 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

Wolfe conditions

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

$$f\left(x_k + \alpha_k p_k\right) \leq f\left(x_k\right) + c_1 \alpha_k \nabla f_k^T p_k \qquad \text{(Sufficient decrease)}$$
$$\nabla f\left(x_k + \alpha_k p_k\right)^T p_k \geq c_2 \nabla f_k^T p_k \qquad \text{(Curvature condition)}$$

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

Backtracking line search

Backtracking Line Search (Algo. 3.1 in [2], Algo. 9.2 in [1])

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

Ensures α_k is never smaller than it needs to 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 [2]

More sophisticated **Wolfe line search**

- Solve the line search problem more exactly
- Guarantee satisfaction of the Wolfe conditions
- Use interpolation polynomials, bracketing and selection phases
- Tedious to code from scratch
- Implemented in scipy.optimize
- See Chapter 3.5 of [2]

Rates of convergence

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

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

- **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

Rates of convergence

Convergence of line search methods

Under mild assumptions, the iterates x_k converge to x^* according to

Method	Rate	Reference
Gradient	Linear	Theorem 3.4 in [2]
Quasi-Newton	Superlinear	Theorem 3.5 in [2]
Newton	Quadratic	Theorem 3.6 in [2]

Rate constants depend on slope properties of f

Constant step size?

- Similar results
- Easier analysis
- Worse rate constants

Gradient descent on strongly convex functions

Gradient descent on strongly convex functions (eq. (9.18) in [1])

Suppose $f(x)$ is strongly convex with

$$mI \preceq \nabla^2 f(x) \preceq MI \quad \forall x$$

Gradient descent with exact line search leads to linear convergence

$$\frac{f(x_{k+1}) - f(x^*)}{f(x_k) - f(x^*)} \leq \frac{\kappa - 1}{\kappa}$$

- $\kappa = M/m$ is the **condition number**

Special cases

- $\kappa = 1$, then $\frac{\kappa-1}{\kappa} = 0$ so minimum found in **one iteration**
- $\kappa \to \infty$, then $\frac{\kappa-1}{\kappa} \to 1$ so convergence gets **arbitrarily slow**

Preconditioning

Trick: Use change-of-variables to reduce condition number of $\nabla^2 f(x)$

Example

Before	Change-of-variables	After
$f(x) = 4x_1^2 + 100x_2^2$	$y_1 = 2x_1$	$g(y) = y_1^2 + y_2^2$
so $\kappa = 100/4 = 25$	$y_2 = 10x_2$	so $\kappa = 1/1 = 1$

Note: Not straightforward in practice

- Non-quadratic $\rightarrow$ Hessian eigenvalues change over domain
  - Always stuck with poor conditioning on some parts of the domain
- May not be easy to find a good scaling
- **Rule-of-thumb**: Use units that make the decision variables have the same order-of-magnitude on the domain of interest

Newton method

Not hard to show that Newton method is **affine invariant**

- Affine change-of-variables has no effect on the convergence rate
- See Chapter 9.5 of [1]

Per-iteration cost of Newton method is high

- Requires storing $\mathcal{O}(n^2)$ entries of $\nabla^2 f_k$
- Requires solving n linear equations $\nabla^2 f_k \cdot (x_{k+1} - x_k) = -\nabla f_k$
  - $\mathcal{O}(n^3)$ operations using basic techniques if $\nabla^2 f_k$ full
  - Fewer operations if $\nabla^2 f_k$ banded, sparse, low rank
    - Chapter 9.7 of [1]
  - Fewer operations if approximate solver e.g. conjugate gradient used
    - Chapter 5, 7.1 of [2]

Example: Strongly convex function (9.20) from [1]

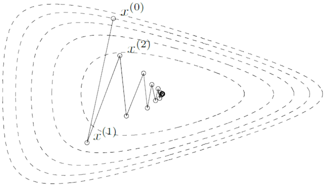


Figure 2: Gradient descent with backtracking line search

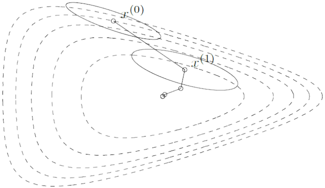


Figure 3: Newton method with backtracking line search

Quasi-Newton Methods

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

Idea: Approximate Hessian with gradient information at multi-points

- In **finite-differencing**, the multi-points are chosen close to x_k i.e. $\{x_k + \varepsilon e_i\}$ where e_i are unit vectors, $0 < \varepsilon << 1$
  - Chapter 8 of [2]
- In **model-based derivative-free methods** the multi-points are spread out over a large trust-region centered on x_k
  - Chapter 9.2 of [2]
- In **quasi-Newton methods**, the multi-points are chosen as the iterates themselves $\{x_0, x_1, \dots, x_k\}$
  - Chapter 6 of [2]

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

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

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

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

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

Answer:

1. Make the gradient of m_{k+1} match the gradient of f at x_{k+1}
2. Make the gradient of m_{k+1} match the gradient of f at x_k
3. Make B_{k+1} symmetric positive definite
4. Make B_{k+1} similar to B_k

DFP and BFGS methods

Davidon-Fletcher-Powell (DFP) update

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

Broyden-Fletcher-Goldfarb-Shanno (BFGS) update

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

Concepts

- B_{k+1} is a **rank-2 update** from B_k
  - Based on most recent iterate x_k and gradient ∇f_k information
- B_k encodes information about curvature from **all previous updates**
- DFP and BFGS are **duals**
- BFGS typically outperforms DFP

SR1 method

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

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

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

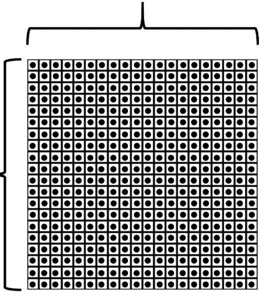
- Actually an **advantage** since true Hessian may be indefinite
- **Fix 1**: Make eigenvalues positive for **line-search** (not recommended by [2])
- **Fix 2**: Use indefinite B_k in a **modified trust-region method** (Algorithm 6.2 in [2])

Large-scale quasi-Newton methods

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

$n = 10{,}000{,}000$

$n = 10{,}000{,}000$



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

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

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

Doing this uses **limited memory**

Special structure of B_k and BFGS updates allows efficient reconstruction of $p_k = -B_k^{-1} \nabla f_k$ **without explicitly constructing** B_k

Limited-memory BFGS (L-BFGS) is derived in Chapter 7.2 of [2]

Trust-Region Methods

Trust-region methods

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

$$f(x+p) \approx m(p) \coloneqq f(x) + \nabla f(x)^\intercal p + \frac{1}{2} p^\intercal \nabla^2 f(x) p$$

Idea: Only trust the model within a small radius Δ of x

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

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

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

Trust-region methods (Figure 2.4 in [2])

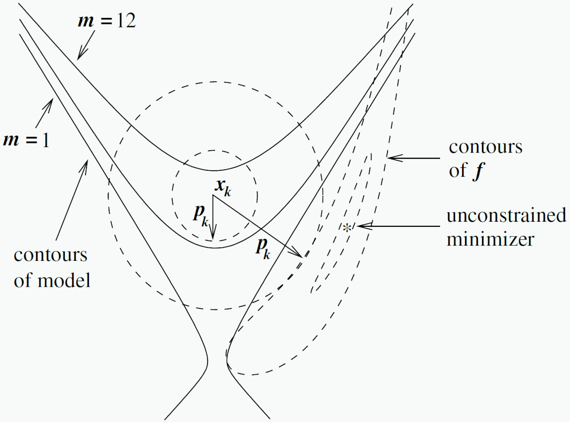


Figure 4: Two possible trust regions (circles) and their corresponding steps p_k. The solid lines are contours of the model function m_k

Trust-region subproblem

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

Case 1: $\Delta \to 0$
Solution is the **gradient step** $p = -\frac{\Delta}{\|\nabla f(x)\|} \nabla f(x)$

Case 2: $\Delta \to \infty$
Solution is the **Newton step** $p = -\nabla^2 f(x)^{-1} \nabla f(x)$

Case 3: $0 < \Delta < \infty$
Solution follows a **curved path**

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

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 [2])

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

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

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

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

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

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

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

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

1. Dogleg method
2. 2d-subspace minimization
3. Conjugate gradient
4. Iterative solution via dual

Quadratic convergence of trust region (Theorem 4.9 in [2])

Under mild assumptions, using the trust region method with any of the approximate subproblem solutions leads to **quadratic convergence**.

Least-squares Methods

Least-squares problems

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

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

where

- $r(x) : \mathbb{R}^n \to \mathbb{R}^m$ 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

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

The gradient and Hessian of the objective are then

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

Idea: Approximate the Hessian as

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

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 algorithm

Gauss-Newton

- Line-search quasi-Newton using the Hessian approximation $B_k = J_k^\intercal J_k$

Can be solved very efficiently

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

$$J_k^\intercal J_k p_k = -J_k r_k$$

  which can be solved with QR or SVD techniques

  - Chapter 10.2 of [2]

Levenberg-Marquadt

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

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
  - Chapter 10.3 of [2]

Accelerated Gradient Methods

Motivation

Large-scale machine learning objectives & gradients are expensive

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

Too many accelerated gradient methods to cover right now

Here are some famous methods & references

- Polyak momentum [10] (Beautiful interactive explainer)
- Nesterov acceleration
  - Original paper (Russian) [11]
  - Nesterov's lecture notes [9]
  - Chapter 3.7 of Bubeck's book [12]
  - Sutskever paper [13]
- Conformal symplectic / relativistic acceleration [14]
  - Zhang et al. [15] question the connection between symplectic integration of the descent ODE and acceleration
- RMSprop (never published, only in Hinton's lecture notes)
- AdaGrad [16]
- Adam & AdaMax [17]

Optimization of Nonsmooth Functions

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

- Ex. sparsity-promoting ℓ_1 regularization $\min f(x) + \|x\|_1$

What can we do?

Smoothing methods [Vandenberghe's notes]
Subgradient methods [Boyd's notes]
Proximal methods [Parikh & Boyd tutorial]

Calculating Derivatives

Let's face it, calculating derivatives by hand is tedious

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

Finite differences	Chapter 8.1 of [2]
Automatic differentiation	Chapter 8.2 of [2]
Symbolic differentiation	Chapter 8 intro of [2]

Constrained Optimization

So far we did not permit explicit constraints on decision variables

What can we do?

1. Form the Lagrangian dual problem & apply unconstrained algorithms
2. Use specialized algorithms that handle constraints natively
   - Simplex algorithm for LPs
   - Projected gradient
   - Proximal algorithms
   - Augmented Lagrangian
   - Alternating direction method of multipliers (ADMM)
   - Interior point methods

LQR Policy Optimization

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

- Attractive theoretical properties
  - Nontrivial slope characteristics
  - Baseline solution from control theoretic techniques
- Practical utility
  - Fundamental problem in control theory
  - Design useful controllers automatically
- Springboard for harder problem classes, algorithms, & analysis
  - Nonlinear systems
  - Model-free reinforcement learning

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

Decision variable: K

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

Assumptions:

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

You already know how to solve LQR using convex programming (from past homework)

$$\begin{aligned} \underset{P}{\text{maximize}} \quad & \operatorname{trace}(PX_0) \\ \text{subject to} \quad & \begin{bmatrix} -P + Q_{xx} + A^\intercal PA & Q_{xu} + A^\intercal PB \\ Q_{ux} + B^\intercal PA & Q_{uu} + B^\intercal PB \end{bmatrix} \succeq 0, \\ & P \succeq 0 \end{aligned}$$

- Form a linear matrix inequality (LMI) constraint
  - Schur complement of algebraic Riccati inequality (ARI)
- Put the semidefinite program (SDP) into CVX and hit run
- Optimal gain $K^* = -(Q_{uu} + B^\intercal PB)^{-1}(Q_{ux} + B^\intercal PA)$

What's wrong with solving LQR via SDP?

- Requires knowledge of problem data (A, B, Q)
- In reinforcement learning, we do not know (A, B, Q)
- But we can estimate objective, gradient, Hessian from observed data
- (A, B, Q) not exposed explicitly in objective, gradient, Hessian

Eliminate constraints to obtain the equivalent problem

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

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

$$P_K = A_K^\intercal P_K A_K + Q_K$$

equivalently

$$\mathrm{vec}(P_K) = (I - A_K^\intercal \otimes A_K^\intercal)^{-1} \,\mathrm{vec}(Q_K)$$

where

$$\begin{aligned} A_K &= A + BK \\ Q_K &= \begin{bmatrix} I \\ K \end{bmatrix}^\intercal Q \begin{bmatrix} I \\ K \end{bmatrix} \end{aligned}$$

Under the given assumptions, the analysis of [18] shows

- f is a **real analytic** function over its domain
  - Implies f has **continuous derivatives of arbitrary order** over its domain
- f is **gradient dominated**, thus invex
- f has a **unique stationary point and global minimizer** K^*

Takeaway

Descent methods will solve the LQR policy optimization problem

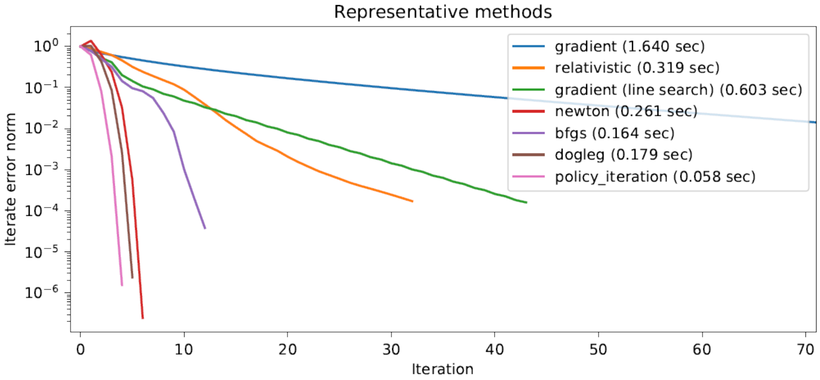
---

Notice that $f(K)$ is expressed with linear algebraic operations

Takeaway

Use automatic differentiation to compute gradients and Hessians

Grab the Python code



[1] Stephen Boyd, Stephen P Boyd, and Lieven Vandenberghe.
Convex optimization.
Cambridge university press, 2004.

[2] Jorge Nocedal and Stephen Wright.
Numerical optimization.
Springer Science & Business Media, 2006.

[3] Michael W Botsko.
A first derivative test for functions of several variables.
The American Mathematical Monthly, 93(7):558–561, 1986.

[4] Adi Ben-Israel and Bertram Mond.
What is invexity?
The ANZIAM Journal, 28(1):1–9, 1986.

[5] Stephen Boyd, Laurent El Ghaoui, Eric Feron, and Venkataramanan Balakrishnan.
Linear matrix inequalities in system and control theory, volume 15.
SIAM, 1994.

[6] 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.

[7] Benjamin Gravell, Peyman Mohajerin Esfahani, and Tyler Summers.
Learning robust control for LQR systems with multiplicative noise via policy gradient.
CoRR, 2019.

[8] Moritz Hardt, Tengyu Ma, and Benjamin Recht.
Gradient descent learns linear dynamical systems.
arXiv preprint arXiv:1609.05191, 2016.

[9] Yurii Nesterov.
Introductory lectures on convex programming volume i: Basic course.
Lecture notes, 3(4):5, 1998.

[10] Boris T Polyak.
Some methods of speeding up the convergence of iteration methods.
Ussr computational mathematics and mathematical physics, 4(5):1–17, 1964.

[11] Yurii E Nesterov.
A method for solving the convex programming problem with convergence rate o $(1/k\hat{}\ 2)$.
In *Dokl. akad. nauk Sssr*, volume 269, pages 543–547, 1983.

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

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] 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.