Numerical Optimization: Basics

Oswin Krause, NO, 2022

KØBENHAVNS UNIVERSITET

Qick notes on Previous Assignment

- Goal of Experimental part:

  Think about how to show correctness of your code

- This is a major problem in this course: you have to convince us that your code is correct
- You have to convince yourself that your code is correct

Quick notes on Previous Assignment

An example solution

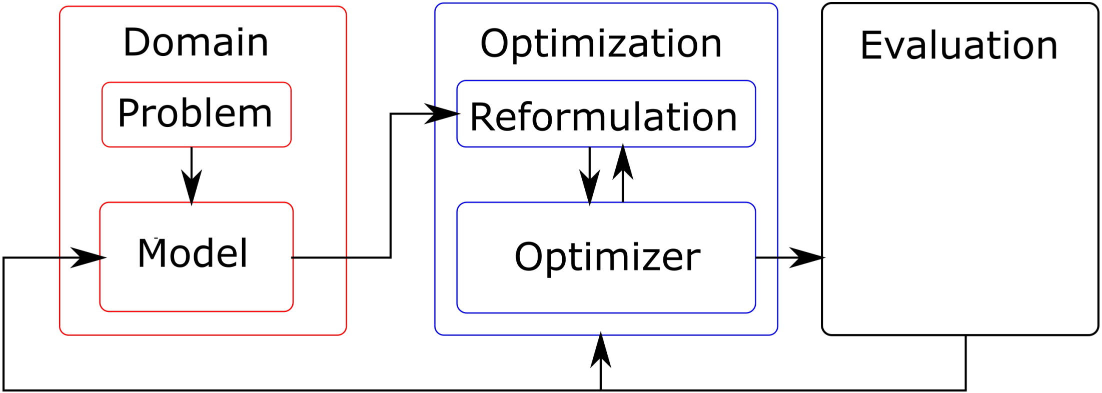
To test my implementation, I compared my code against the gradients computed by autograd.numpy in 2D. I evaluated gradients and Hessians at the location of 100 uniformly randomly sampled points in the range $[-2, 2]^2$*. I measured the absolute difference between the vector/matrix entries computed by my code and the corresponding entry of the ground truth computation.*

Over all estimates, the largest deviation was 10^{-5} *on* f_3*. Compared to a value of the corresponding partial derivative of 2 at that point, this is a very small deviation.*

I further performed the following tests...

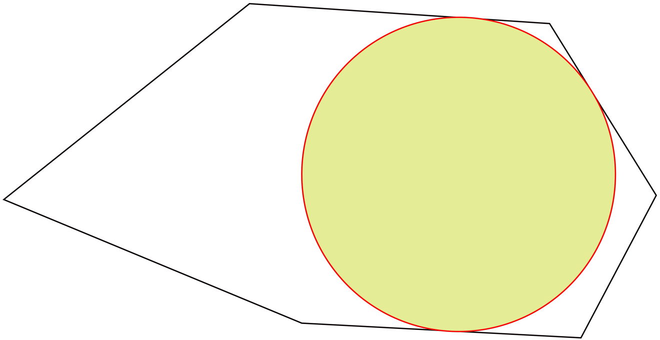
I therefore conclude, that my numerical implementation is close to the result computed by autograd and that my gradients pass basic comparisons with the analytic gradients.

Role of Optimization



Example: Deriving a Problem

Example: Maximum inscribed circle problem

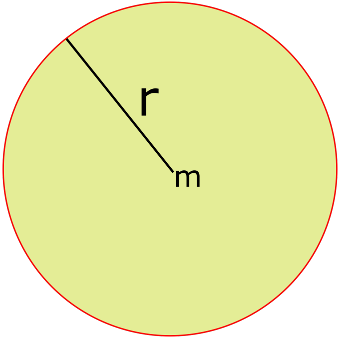


- Task: find the largest circle fitting into a polytope
- Modeling: Find the ball with maximum volume such that all its points are inside a convex polytope

Example (cont): Describing the objects

Ball:

$$B(m, r) = \{x \in \mathbb{R}^n \mid \|x - m\| \leq r\}$$



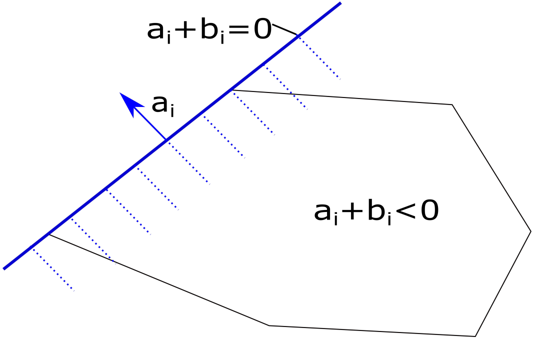
Example (cont): Describing the objects

Convex Polytope:

- K Linear Inequalities

$$a_i^T x + b_i \leq 0,\ i = 1, \ldots, K$$

- Polytope P set of $x \in \mathbb{R}^n$ fulfilling all inequalities



Example (cont): Optimization problem (model)

$$\begin{aligned}\max_{r,m}\ & \mathrm{Vol}(B(m,r)) \\ \text{s.t. }\ & x \in P, \forall x \in B(m,r) \\ & \wedge\ r \geq 0\end{aligned}$$

- Objective: maximize volume of the ball
- Feasible region: Only allow values for (m, r) which lead to balls contained in P
- This formulation is a good model, but can't be computed
- Needs reformulation!

Example (cont): Reformulation objective

- Reformulate $\mathrm{Vol}(B(m, r))$

Example (cont): Reformulation objective

- Reformulate $\mathrm{Vol}(B(m, r))$
- Make use of closed form formula
  - $n = 2$: volume=area

$$\mathrm{Vol}(B(m, r)) = 2\pi r^2$$

  - Does not depend on m

Example (cont): Reformulation objective

- Reformulate $\text{Vol}(B(m, r))$
- Make use of closed form formula
  - $n = 2$: volume=area

$$\text{Vol}(B(m, r)) = 2\pi r^2$$

  - Does not depend on m
- Since $r > 0$
  - Objective monotonic increasing in r

  $\rightarrow$ Optimal r does not change if we just maximize r

Example (cont): Optimization problem (intermediate)

$$
\begin{aligned}
&\max_{r,m}\ r \\
&\text{s.t. } x \in P, \forall x \in B(m, r) \\
&\quad \wedge\ r \geq 0
\end{aligned}
$$

- Need to reformulate constraint

Example (cont): Reformulation constraint

- Goal: only allow balls that are fully contained in the polytope

Example (cont): Reformulation constraint

- Goal: only allow balls that are fully contained in the polytope
- Idea:
  - Check that midpoint is inside
  - And its minimum distance to any point on boundaries is $>= r$

Example (cont): Reformulation constraint

- Goal: only allow balls that are fully contained in the polytope
- Idea:
  - Check that midpoint is inside
  - And its minimum distance to any point on boundaries is $>= r$
- Both conditions are fulfilled if (assuming $\|a_i\| = 1$)

$$a_i^T m + b_i \leq -r, \text{ for } i = 1, \ldots, K$$

- (We might get back to that later, Week 7.)

Optimization problem (final)

$$
\begin{aligned}
&\max_{r,m} r \\
&\text{s.t. } a_i^T m + b_i \leq -r, \text{ for } i = 1, \ldots, K \\
&\quad \wedge r \geq 0
\end{aligned}
$$

This is a linear program

- Objective function is linear
- All constraints are linear

$\rightarrow$ There are specialized algorithms for this type of problem (super fast!)

Example: Optimization Basics

Setup in this course

- We are given an *objective function* $f : \mathbb{R}^n \to \mathbb{R}$.
- A *feasible region*, $\mathcal{C} \subseteq \mathbb{R}^n$
- *Feasible regions* are represented by *constraints*
- Task: Search for a point $x^* \in \mathcal{C}$ where f is minimal

$$x^* = \arg\min_{x \in \mathcal{C}} f(x)$$

Numerical optimization

Goal of numerical optimization:
For a given objective function f with feasible region $\mathcal{C}$, find an optimization algorithm $\mathcal{A}$ that produces a series of candidate points x_k such, that

- Often $f(x_{k+1}) < f(x_k)$
- $(x_k)_k$ converges to some $x' \in \mathcal{C}$
- x' is a minimizer

Numerical optimization

Goal of numerical optimization:
For a given objective function f with feasible region $\mathcal{C}$, find an optimization algorithm $\mathcal{A}$ that produces a series of candidate points x_k such, that

- Often $f(x_{k+1}) < f(x_k)$
- $(x_k)_k$ converges to some $x' \in \mathcal{C}$
- x' is a minimizer

This is very abstract. We need to know:

- How do we recognize a minimizer?
- How do we know $(x_k)_k$ approaches a minimizer?
- How do we measure how well an algorithm does that?
- What properties do we expect of an algorithm?

Desirable properties of an algorithm

We want $\mathcal{A}$ to be

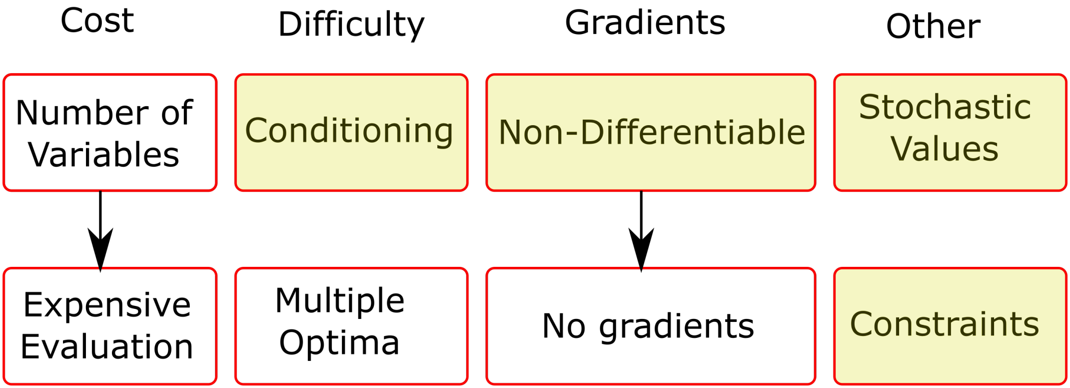
- Robust: Works on as many f as possible
- Efficient: Finds the optimum quickly with small computation cost
- Accurate: Locates the optimum with high precision
- Stable: Numeric implementation does not fail

Discuss (8 min): How can we show that an algorithm is robust, efficient and/or accurate and stable? Are they mutually exclusive?

Difficulties when optimizing a function

Cost	Difficulty	Gradients	Other
Number of Variables	Conditioning	Non-Differentiable	Stochastic Values
Expensive Evaluation	Multiple Optima	No gradients	Constraints

Difficulties when optimizing a function



Unconstrained Continuous Optimization

Setup

For now, we restrict ourselves to simple problems:

- $f : \mathbb{R}^n \to \mathbb{R}$ is a smooth function

$\to$ smooth: infinitely often continuous differentiable.

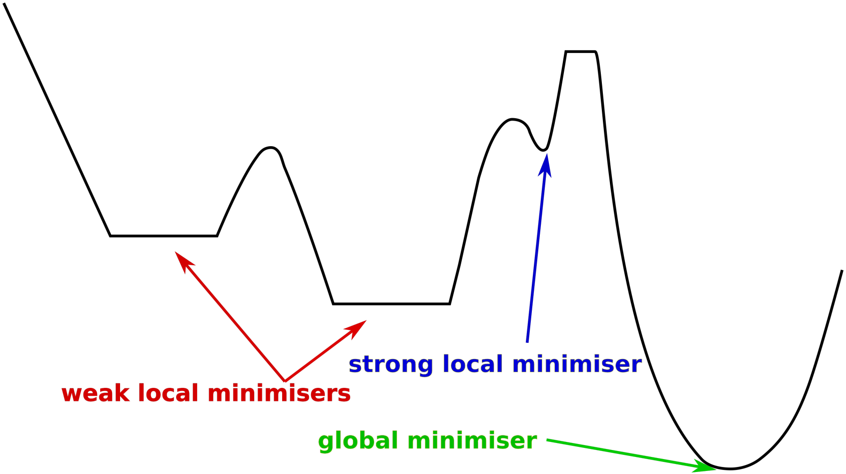
- Unconstrained (Feasible region $\mathcal{C} = \mathbb{R}^n$)
- For convergence proofs of algorithms, we might need stronger conditions

Minimisers I

Definition (Minimisers)

1. x^* is a weak local minimiser of f if there is a neighbourhood $\mathcal{N} \subseteq \mathbb{R}^n$ of x^* with
$$\forall x \in \mathcal{N}, f(x^*) \leq f(x)$$
2. x^* is a strict/strong local minimiser of f if there is a neighbourhood $\mathcal{N} \subseteq \mathbb{R}^n$ of x^* with
$$\forall x \in \mathcal{N} \setminus \{x^*\}, f(x^*) < f(x)$$
3. x^* is an isolated local minimiser of f if there is a neighbourhood $\mathcal{N} \subseteq \mathbb{R}^n$ of x^* such that x^* is the unique minimiser of f in $\mathcal{N}$.

Minimisers II



Conditions for Minimizers

Theorem (Necessary conditions)

- *First order necessary condition. If x^* is a local minimiser of f over a neighbourhood $\mathcal{N}$, then $\nabla f(x^*) = 0$, i.e, x^* is a critical point of f.*
- *Second order necessary condition. At a critical point, $\nabla^2 f(x^*)$ must be positive semidefinite.*

Theorem (Second order sufficient conditions)

Assume that x^* is a critical point of f. Then if $\nabla^2 f(x^*)$ is positive definite, x^* is a strict local minimiser

Proof: First Order Necessary Conditions

Proof idea: Show that we can find a better point in any neighbourhood around x^*, when $\nabla f(x^*) \neq 0$

Proof: First Order Necessary Conditions

Proof idea: Show that we can find a better point in any neighbourhood around x^*, when $\nabla f(x^*) \neq 0$

- Assume $\nabla f(x^*) \neq 0$

Proof: First Order Necessary Conditions

Proof idea: Show that we can find a better point in any neighbourhood around x^*, when $\nabla f(x^*) \neq 0$

- Assume $\nabla f(x^*) \neq 0$
- Let $p = -\nabla f(x^*)$. We have

$$p^T \nabla f(x^*) = -\nabla f(x^*)^T \nabla f(x^*) = -\|\nabla f(x^*)\|^2 < 0$$

Proof: First Order Necessary Conditions

Proof idea: Show that we can find a better point in any neighbourhood around x^*, when $\nabla f(x^*) \neq 0$

- Assume $\nabla f(x^*) \neq 0$
- Let $p = -\nabla f(x^*)$. We have

$$p^T \nabla f(x^*) = -\nabla f(x^*)^T \nabla f(x^*) = -\|\nabla f(x^*)\|^2 < 0$$

- Define $g(t) = f(x^* + tp) - f(x^*)$ with
  - $g'(t) = \nabla f(x^* + tp)^T p$
  - $g'(0) = -\|\nabla f(x^*)\|^2 < 0$

Proof: First Order Necessary Conditions

Proof idea: Show that we can find a better point in any neighbourhood around x^*, when $\nabla f(x^*) \neq 0$

- Assume $\nabla f(x^*) \neq 0$
- Let $p = -\nabla f(x^*)$. We have

$$p^T \nabla f(x^*) = -\nabla f(x^*)^T \nabla f(x^*) = -\|\nabla f(x^*)\|^2 < 0$$

- Define $g(t) = f(x^* + tp) - f(x^*)$ with
  - $g'(t) = \nabla f(x^* + tp)^T p$
  - $g'(0) = -\|\nabla f(x^*)\|^2 < 0$
- Statement true, if $\exists T > 0$ such, that for all $0 < t \leq T$ holds $g(t) < 0$

Proof: First Order Necessary Condition

First: Use Taylor expansion on g around 0 to introduce the derivative

- We have $g(0) = 0$ and $g'(0) < 0$
- Remember: f (and thus g are smooth)

Proof: First Order Necessary Condition

First: Use Taylor expansion on g around 0 to introduce the derivative

- We have $g(0) = 0$ and $g'(0) < 0$
- Remember: f (and thus g are smooth)

→ We can create first-order Taylor expansion of g around 0

$$g(0+t) = \underbrace{g(0)}_{0} + t\,\underbrace{g'(0)}_{<0} + \underbrace{\frac{t^2}{2}g''(c)}_{\text{Error Term}},\ c \in [0, t]$$

- Remember: c depends on the chosen t.

Proof: First Order Necessary Condition

First: Use Taylor expansion on g around 0 to introduce the derivative

- We have $g(0) = 0$ and $g'(0) < 0$
- Remember: f (and thus g are smooth)

$\rightarrow$ We can create first-order Taylor expansion of g around 0

$$g(0+t) = \underbrace{g(0)}_{0} + t\,\underbrace{g'(0)}_{<0} + \underbrace{\frac{t^2}{2} g''(c)}_{\text{Error Term}},\; c \in [0, t]$$

- Remember: c depends on the chosen t.
- Next step: pick t small enough that the error term is smaller than $tg'(0)$

Proof: First Order Necessary Condition

Final step: construct points with lower function value.

- Let $M = \max_{c \in [0,1]} |g''(c)|$.

Proof: First Order Necessary Condition

Final step: construct points with lower function value.

- Let $M = \max_{c \in [0,1]} |g''(c)|$.
- Pick $T = \min\{-\frac{g'(0)}{M}, 1\}$
- We have for $T < 1$

$$Tg'(0) + \frac{T^2}{2} \underbrace{\overbrace{g''(c)}^{c \in [0,1]}}_{\in [-M,M]} = -\frac{(g'(0))^2}{M} + \frac{1}{2}\frac{(g'(0))^2}{M}\underbrace{\frac{g''(c)}{M}}_{\in [-1,1]} < 0$$

Proof: First Order Necessary Condition

Final step: construct points with lower function value.

- Let $M = \max\limits_{c\in[0,1]} |g''(c)|$.
- Pick $T = \min\{-\frac{g'(0)}{M}, 1\}$
- We have for $T < 1$

$$Tg'(0) + \frac{T^2}{2} \underbrace{\overbrace{g''(c)}^{c\in[0,1]}}_{\in[-M,M]} = -\frac{(g'(0))^2}{M} + \frac{1}{2}\frac{(g'(0))^2}{M}\underbrace{\frac{g''(c)}{M}}_{\in[-1,1]} < 0$$

- For $T = 1$, the results holds, since $|g''(c)| \leq |g'(0)|$
- As the error terms shrinks faster with t than the linear term, this holds $\forall t \in (0, T]$.

$\Rightarrow$ There is no neighbourhood around x^* where $f(x^*)$ is minimal

$\Rightarrow$ x^* cannot be local minimum.

Special case: Convex functions

Definition (Convex Functions)

Let $f : \mathbb{R}^n \to \mathbb{R}$ be a function. We call f *convex* if $\forall x, p \in \mathbb{R}^n$ and $\lambda \in (0, 1)$ holds

$$f(x + \lambda p) \leq f(x) + \lambda(f(x + p) - f(x)) \ .$$

Further, if the stricter condition

$$f(x + \lambda p) < f(x) + \lambda(f(x + p) - f(x)) \ .$$

holds, then we call f *strictly convex*

Note: convexity is often written by substitution $y = x + p$. This formulation above is more useful to us.

Characterization of differentiable convex functions

Lemma

Let $\mathcal{C} \subseteq \mathbb{R}^n$ and $f : \mathcal{C} \to \mathbb{R}$ be a twice continuous differentiable function.

f is convex iff

$$\nabla^2 f(x) \text{ is positive semi-definite}$$

(for strict convexity, $\nabla^2 f(x)$ is PD)

Why convexity matters

- One can show: f convex $\Rightarrow$ f has at most one local minimum.

Why convexity matters

- One can show: f convex $\Rightarrow$ f has at most one local minimum.
- For all smooth f
  - Second order necessary condition of minima: $\nabla^2 f$ is PD at a minimum

Why convexity matters

- One can show: f convex $\Rightarrow f$ has at most one local minimum.
- For all smooth f
  - Second order necessary condition of minima: $\nabla^2 f$ is PD at a minimum
  - $\rightarrow$ Continuity: f is convex in a neighbourhood close to a local minimum
  - $\rightarrow$ Algorithms that work well on convex functions, work well close to the optimum
  - $\rightarrow$ We just have to ensure to get there.

Basics of Algorithm Design and Evaluation

Basic Idea of optimization algorithm

- Target function f complex, hard to compute

$\rightarrow$ We can't find minima directly

Basic Idea of optimization algorithm

- Target function f complex, hard to compute

$\rightarrow$ We can't find minima directly

- Idea of basic optimization loop
  1. Approximate $f(x+p)$ around x by simpler function
     - $m : \mathbb{R}^n \rightarrow \mathbb{R}$, $m(p) \approx f(x+p)$
     - Ensure that m can be analyzed easily

Basic Idea of optimization algorithm

- Target function f complex, hard to compute

$\rightarrow$ We can't find minima directly

- Idea of basic optimization loop
  1. Approximate $f(x+p)$ around x by simpler function
     - $m : \mathbb{R}^n \rightarrow \mathbb{R}$, $m(p) \approx f(x+p)$
     - Ensure that m can be analyzed easily
  2. Find p with $m(p) < m(0) \approxeq f(x)$

Basic Idea of optimization algorithm

- Target function f complex, hard to compute

$\rightarrow$ We can't find minima directly

- Idea of basic optimization loop
  1. Approximate $f(x+p)$ around x by simpler function
     - $m : \mathbb{R}^n \rightarrow \mathbb{R}$, $m(p) \approx f(x+p)$
     - Ensure that m can be analyzed easily
  2. Find p with $m(p) < m(0) \approxeq f(x)$
  3. Evaluate $f(x+p)$ and correct p (or m) to ensure $f(x+p) < f(x)$

Basic Idea of optimization algorithm

- Target function f complex, hard to compute

$\rightarrow$ We can't find minima directly

- Idea of basic optimization loop
  1. Approximate $f(x+p)$ around x by simpler function
     - $m : \mathbb{R}^n \rightarrow \mathbb{R}$, $m(p) \approx f(x+p)$
     - Ensure that m can be analyzed easily
  2. Find p with $m(p) < m(0) \approxeq f(x)$
  3. Evaluate $f(x+p)$ and correct p (or m) to ensure $f(x+p) < f(x)$
  4. Set $x \rightarrow x+p$ and go to 1.

Example: Line-Search based Gradient Descent

1. Set $m(p) = f(x) + p^T \nabla f(x)$ (first order Taylor)
2. Pick $p = -\nabla f(x)$
3. Find α such, that $f(x + \alpha p) < f(x)$
4. Set $x \to x + \alpha p$ and go to 1.

Example: Trust-Region Newton

1. Set $m(p) = f(x) + p^T \nabla f(x) + \frac{1}{2} p^T \nabla^2 f(x) p$ (second order Taylor)
2. $p = \min_p m(p)$ so, that $\|p\| \leq \Delta$
3. If $f(x+p) > f(x)$
   3.1 Reduce Δ
   3.2 Go to 2.
4. Set $x \to x + p$ and go to 1.

When to stop?

- Let $(x_k)_k$ be the sequence of iterates by the algorithm
- In most cases $f(x^*) \neq 0$ and x^* unknown
- How do we know, when to stop? Discuss (5min)

When to stop?

- Let $(x_k)_k$ be the sequence of iterates by the algorithm
- In most cases $f(x^*) \neq 0$ and x^* unknown
- How do we know, when to stop? Discuss (5min)
- Useful criteria
  - Maximum number of iterations
  - Length of steps(?)
  - Improvement of function value(?)
  - Norm of the gradient

When to stop?

- Let $(x_k)_k$ be the sequence of iterates by the algorithm
- In most cases $f(x^*) \neq 0$ and x^* unknown
- How do we know, when to stop? Discuss (5min)
- Useful criteria
  - Maximum number of iterations
  - Length of steps(?)
  - Improvement of function value(?)
  - Norm of the gradient
- Norm of the gradient?
  - We know $(\nabla f(x_k))_k \to 0$, when $(x_k)_k \to x^*$

  $\to$ use $\|\nabla f(x_k)\| < \epsilon$ as criterion

Measuring convergence Rate

- As algorithm designers we want to know how good algorithms are.
- We can measure their performance on Benchmark functions.
- Discuss: What are good metrics to compare algorithms? (5min)

Measuring convergence Rate

- As algorithm designers we want to know how good algorithms are.
- We can measure their performance on Benchmark functions.
- Discuss: What are good metrics to compare algorithms? (5min)
- Often used:
  - Number of steps until termination
  - Function value after N steps
- Problem: only tells us about performance at one point
  - Algorithms might slow down over time
  - Or speed up

Measuring convergence Rate

- As algorithm designers we want to know how good algorithms are.
- We can measure their performance on Benchmark functions.
- Discuss: What are good metrics to compare algorithms? (5min)
- Often used:
  - Number of steps until termination
  - Function value after N steps
- Problem: only tells us about performance at one point
  - Algorithms might slow down over time
  - Or speed up
- Can we somehow measure the convergence rate of the seuqences:
  - $(\|x_k - x^*\|)_k$
  - $(f(x_k) - f(x^*))_k$
  - $(\|\nabla f(x_k)\|)_k$

Experimentally verifying Linear Convergence

Remember linear Convergence (example with $\|x_k - x^*\|$):

$$\frac{\|x_k - x^*\|}{\|x_{k-1} - x^*\|} \leq M$$

Experimentally verifying Linear Convergence

Remember linear Convergence (example with $\|x_k - x^*\|$):

$$\frac{\|x_k - x^*\|}{\|x_{k-1} - x^*\|} \leq M$$

Rewrite:

$$\begin{aligned}
\|x_k - x^*\| &\leq M\|x_{k-1} - x^*\| \\
&\leq M^2\|x_{k-2} - x^*\| \\
&\leq \dots \\
&\leq M^k\|x_0 - x^*\|
\end{aligned}$$

Experimentally verifying Linear Convergence

Remember linear Convergence (example with $\|x_k - x^*\|$):

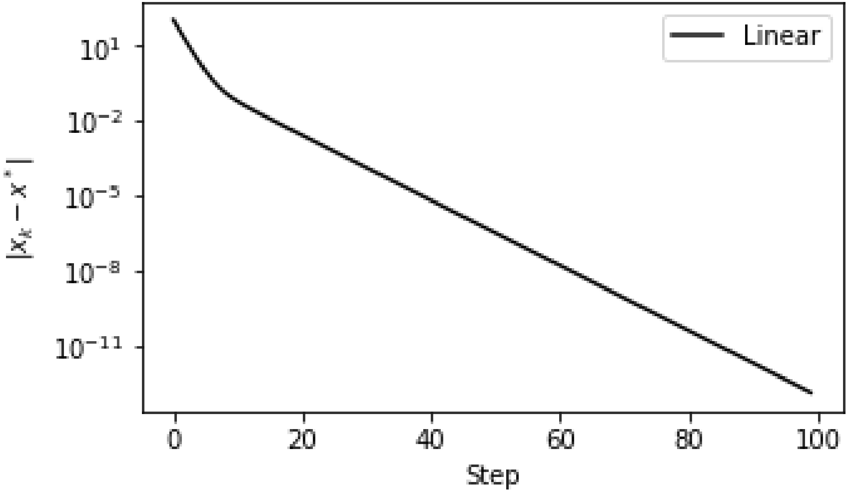
$$\frac{\|x_k - x^*\|}{\|x_{k-1} - x^*\|} \leq M$$

Rewrite:

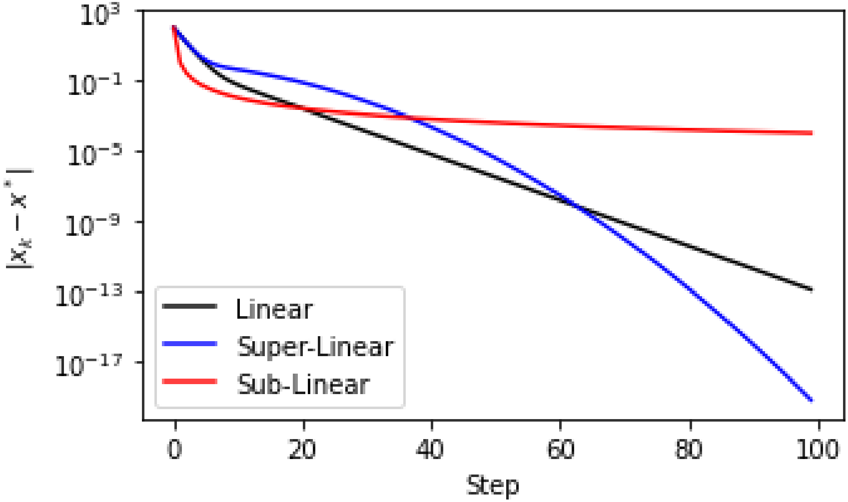
$$\begin{aligned} \|x_k - x^*\| &\leq M\|x_{k-1} - x^*\| \\ &\leq M^2\|x_{k-2} - x^*\| \\ &\leq \dots \\ &\leq M^k\|x_0 - x^*\| \end{aligned}$$

$\log\|x_k - x^*\| \leq k \log M + \log\|x_0 - x^*\|$ is approximately a linear function, when plotted on semi-log plot

Linear convergence rate: line in semi-log plot



Other convergence rates: visible



Quadratic convergence: obvious