Modern Optimization Methods for Big Data Problems

MATH11146
The University of Edinburgh
Spring 2017

Peter Richtárik

Modern Optimization Methods for Big Data Problems

Lecture 1

Introduction

January 16, 2017

Outline

Course Organization

The Course at a Glance

- Course
  - SCQF Level 11
  - 10 SCQF credits
- 10 $\times$ 2-hour lecture
  - first 8 lectures cover theory
  - last 2 lectures devoted to solving exercises
- 10 $\times$ 1-hour computer lab
- 3 assignments
  - coding
  - 50% of final mark
- 1 exam
  - theory
  - 2 hours
  - best 2 of 3 problems
  - 50% of final mark

Timetable

Week 1: Monday 16.1, Wednesday 18.1
Week 2: Monday 23.1, Wednesday 25.1
Week 3: no teaching
Week 4: Monday 6.2, Wednesday 8.2
Week 5: Monday 13.2, Wednesday 15.2
Week 6: Monday 27.2, Wednesday 1.3

Mondays

- 14:10–16:00, Lecture, JCMB 6201
- 16:10–17:00, Lab, JCMB 3210

Wednesdays

- 14:10–16:00, Lecture, JCMB 6206
- 16:10–17:00, Lab, JCMB 5205

Lectures

- Weeks 1,2,4,5
  - theory based
  - delivered by Peter Richtárik
- Week 6
  - blackboard problem solving
  - aimed to get your exam preparation started
  - led by Jakub Konečný (http://www.jakubkonecny.com/)
  - may contain up to 1hr of extra theory
- Lecture slides will be distributed before each lecture. I will say some things which are not on the slides, but most stuff is there. Therefore, you do not have to do much writing, and can focus on the material. Ask questions during lectures!
- Essential Background:
  - Linear Algebra
  - Multivariate Calculus
  - Probability
  - Programming

Computer Labs

- **Goal:** practical implementation of material learned in the lectures.
- **Tutors + Assignment Markers:**
  - Sona Galovičová (PhD student, Data Science, School of Informatics)
  - Filip Hanzely (PhD student, Optimization, School of Mathematics)
  - Nicolas Loizou (PhD student, Optimization, School of Mathematics)
- All coding will be done in the **Julia** language. Why Julia?
  - Julia **combines the speed of C, with programming ease of MATLAB/Python**
  - Many influential people think Julia is the future of scientific computing
  - Prior knowledge of Julia is not required (after all, Julia is a young language and hence not many know it!). You will pick it up during the course.
- At the start of each lab, go to

  https://www.juliabox.com/

  and **sign in** with one of the following accounts: **LinkedIn, GitHub, Google. If you do not have any of these three accounts, create one!**

Assignments

Assignment #	Posted	Deadline	% of final mark
1	22.1	5.2 by 10pm	50/3
2	5.2	19.2 by 10pm	50/3
3	19.2	5.3 by 10pm	50/3

- All assignments will be posted via Learn
- If, for any reason, an assignment is posted late, each 1 day of delay results in 2 days of extension!
- Submit electronically via Learn
- You will have 3 upload attempts, the last upload before the deadline will be marked
- All assignments will be Julia-based

Optimization: Basic Terminology

Optimization Problems

$$\begin{array}{ll} \text{minimize} & f(x) \\ \text{subject to} & x \in Q \subseteq \mathbb{R}^n \end{array} \qquad (1)$$

- a set Q **(feasible set / constraint)**
- a real-valued function $f : Q \to \mathbb{R}$ **(objective function / cost function / loss function)**

Condition	**Problem (1) is called**
$Q = \mathbb{R}^n$	**unconstrained**
$Q \neq \mathbb{R}^n$	**unconstrained**
$Q \neq \emptyset$	**feasible**
$Q = \emptyset$	**infeasible**
$\exists C \in \mathbb{R}$ such that $f(x) \geq C$ for all $x \in Q$	**bounded**
$\exists \{x_k\}_{k=0}^{\infty} \subseteq Q$ such that $f(x_k) \to -\infty$	**unbounded**

Basic Types of Optimization Problems

Q is often described as a solution of a (finite) system of equations and inequalities:

$$Q = \{x \ : \ g_i(x) = 0, \ i \in E; \ g_i(x) \leq 0, \ i \in I\}.$$

- E is some finite index set, indexing **E**qualities
- I is some finite index set, indexing **I**nequalities

Condition	**Problem (1) is called**
f, g_i are linear	**Linear Program** (LP)
f not linear	**Nonlinear Program** (NLP)
f quadratic, g_i linear	**Quadratic Program** (QP)
f, g_i quadratic	**Quadratically Constrained QP** (QCQP)
f, Q convex	**Convex Program** (CP)
LP with $x_i \in Z$ for some i	**Integer Program** (IP)

We shall deal with **convex problems** and **convex quadratic problems** in this course.

Solution Concepts

Condition	x is called
any x	**a solution/point**
$x \in Q$	**feasible solution/point**
$x \notin Q$	**infeasible solution/point**
$f(x) \leq f(y)$ for all $y \in Q \cap N(x, \delta)$	**local minimizer**
$f(x) \leq f(y)$ for all $y \in Q$	**global minimizer**
$f(x) \leq f(y) + \epsilon$ for all $y \in Q \cap N(x, \delta)$	**ϵ-solution** (local)
$f(x) \leq f(y) + \epsilon$ for all $y \in Q$	**ϵ-solution** (global)

Notation used above:

- For $x = (x_1, \ldots, x_n) \in \mathbb{R}^n$, by $\|x\|$ we denote the **standard Euclidean norm** of x, defines as:

$$\|x\| = \sqrt{\sum_{i=1}^n x_i^2}.$$

- The **δ-neighbourhood** of $x \in \mathbb{R}^n$ is the set:

$$N(x, \delta) \stackrel{\text{def}}{=} \{y \ : \ \|y - x\| \leq \delta\}$$

Optimization Modelling

1. One wants to solve some **problem**
   - Predict what advert a given user will click on
   - Deblur an image
   - Rank a collection webpages based on their relevance to a given query
2. **Represent the solution** of the problem as a mathematical object; typically a vector $x \in \mathbb{R}^n$
   - x_i represents the probability of clicking on advert i
   - x_i represents the color of pixel i
   - x_i represents relevance of website i to the query
3. Choose a set $Q \subseteq \mathbb{R}^n$ of solutions which you are willing the consider. Q is the **feasibility/constraint set.**
   - $Q = \{x \in \mathbb{R}^n \ : \ 0 \leq x_i \leq 1, \ \sum_i x_i = 1\}$
   - $Q = \{x \in \mathbb{R}^n \ : \ 0 \leq x_i \leq 256\}$
   - $Q = \mathbb{R}^n$ or $Q = \{x \in \mathbb{R}^n \ : \ x \geq 0\}$
4. Choose a **cost/loss/objective function** $f : \mathbb{R}^n \to \mathbb{R}$ measuring the quality of each potential solution. In particular, $f(x) < f(y)$ should mean that x is a better solution than y.
5. Solver problem (OPT) using a suitable **optimization algorithm**

Optimization and Data

Sources of data:

1. **Description data:** Data forming the initial logical description of the problem
2. **Representation data:** Our choice of representation may depend on some synthetic / computed / collected data
3. **Feasibility data:** Data describing Q
4. **Objective data:** Data describing f

Optimization Algorithms: Basic Terminology

Direct vs Iterative Methods I

Direct methods

- Perform a **finite number of steps.**
- Intermediary steps can't be used to produce an approximate solution.
- The solution found is (typically) **exact.**

Iterative methods

- Perform an **infinite number of steps** (in principle).
- Intermediary steps generate a sequence of iterates $x^0, x^1, x^2, \ldots$ which get progressively better (in some sense).
- Any solution in the sequence is **approximate** only

Direct vs Iterative Methods II

Example 1

Solve problem (1) with $f(x) = \frac{1}{2}\|Ax - b\|^2$ and $Q = \mathbb{R}^n$.

- **Direct method:** Set the gradient to zero, which gives $\nabla f(x) = A^\top(Ax - b) = 0$. This results to the linear system $A^\top Ax = A^\top b$. Solve the system using any direct solver for linear systems, such as Gaussian elimination. The result is obtained after a finite number of steps.
- **Iterative method:** Choose some initial guess $x^0 \in \mathbb{R}^n$, compute constant L satisfying the bound $L \geq \lambda_{\max}(A^\top A)$ and perform the following iterative process ("gradient descent"):

$$x^{k+1} = x^k - \frac{1}{L}\nabla f(x^k).$$

  It can be shown that $f(x^k) \leq \frac{C}{k}$ for all k and some constant C (which depends on x^0, A and b).

We will only talk about iterative methods in this course; these are by far the most prevalent and useful in optimization.

Iterative Optimization Methods

All iterative methods can be written in the generic form

$$x^{k+1} = x^k + \alpha^k h^k,$$

where

- h^k is the **search/descent direction**
- α^k is the **stepsize**
- $\alpha^k h^k$ is the **step**

Iterative optimization methods differ in how:

- the direction h^k is computed (a major distinguishing factor)
- the stepsize α^k is determined (a minor distinguishing factor)

Complexity

Two key questions when choosing an optimization method:

- How costly is it to get from x^k to x^{k+1}?
  - in terms of **time,** or
  - in terms of **# of arithmetic operations (additions, multiplications) performed**
- How many iterations k do we need to perform to get a solution of acceptable quality?

Definition 2 (Complexity of an Optimization Method)

Let $\mathcal{M}$ be an (iterative) optimization method for solving (1). Let $\epsilon > 0$ be a desired error tolerance with respect to some measure of success. Let W be the total arithmetic cost performed by $\mathcal{M}$ in each iteration. Let $k(\epsilon)$ be the number of iterations sufficient for the output of $\mathcal{M}$ to be within the desired tolerance ϵ. The **total complexity** of $\mathcal{M}$ is the function

$$\epsilon \mapsto k(\epsilon) \times W.$$

Remark: Typically, both W and $k(\epsilon)$ depend on the problem at hand (i.e., on f and Q).

Classification of Optimization Methods

Definition 3

We say that an (iterative) optimization algorithm is

- **feasible** if all iterates x^k are feasible, i.e., if $x^k \in Q$ for all k
- **infeasible** if it is not feasible
- **deterministic** if, given x^0, always the same sequence of iterates $\{x^k\}$ is produced
- **randomized** if the iterates $\{x^k\}$ form a random process (i.e., if they are random vectors)
- **zero order** if it is only allowed to compute zero-order information about f (i.e., function values)
- **first order** if it is only allowed to compute up to first-order information about f (i.e., function values and gradients)
- **second order** of it is allowed to compute up to second-order information about f (i.e., function values, gradients and Hessian)

Zero vs First vs Second Order Methods

Method	Local information at x about f utilized
zero order	function value $f(x) \in \mathbb{R}$
first order	gradient $\nabla f(x) \in \mathbb{R}^n$
second order	Hessian $\nabla^2 f(x) \in \mathbb{R}^{n \times n}$

Gradient and Hessian

$$\nabla f(x) = \begin{pmatrix} \frac{\partial f(x)}{\partial x_1} \\ \frac{\partial f(x)}{\partial x_2} \\ \vdots \\ \frac{\partial f(x)}{\partial x_n} \end{pmatrix} \in \mathbb{R}^n \qquad \nabla^2 f(x) = \begin{pmatrix} \frac{\partial^2 f(x)}{\partial x_1 \partial x_1} & \frac{\partial^2 f(x)}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f(x)}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f(x)}{\partial x_2 \partial x_1} & \frac{\partial^2 f(x)}{\partial x_2 \partial x_2} & \cdots & \frac{\partial^2 f(x)}{\partial x_2 \partial x_n} \\ & \vdots & & \\ \frac{\partial^2 f(x)}{\partial x_n \partial x_1} & \frac{\partial^2 f(x)}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f(x)}{\partial x_n \partial x_n} \end{pmatrix} \in \mathbb{R}^{n \times n}$$

Key to Higher Order Methods: Taylor's Theorem

Landau's little "oh" notation

By $o(t)$ we mean any function $o : \mathbb{R} \to \mathbb{R}$ such that $\lim_{t\to 0} \frac{o(t)}{t} = 0$. Informally, these are functions which approach zero faster than "linearly"; i.e., faster than the $t \mapsto t$ function does.

Example 4

$t^2 = o(t)$ since $\lim_{t\to 0} \frac{t^2}{t} = t = 0$.

Theorem 5 (Multivariate Taylor's Theorem)

Let $U \subseteq \mathbb{R}^n$ be an open subset of $\mathbb{R}^n$. Fix any $x \in U$ and $h \in \mathbb{R}^n$ such that $x + th \in U$ for all $t \in [0, 1]$.

1. *If $f : U \to \mathbb{R}$ is* **differentiable,** *then*

$$f(x + h) = f(x) + \langle \nabla f(x), h \rangle + o(\|h\|).$$

2. *If $f : U \to \mathbb{R}$ is* **twice differentiable,** *then*

$$f(x + h) = f(x) + \langle \nabla f(x), h \rangle + \frac{1}{2}\langle \nabla^2 f(x)h, h \rangle + o(\|h\|^2).$$

Basic Algorithms

A Quick Review of Basic Methods

Direct Search (Zero-order method)

- Fix a set $\mathcal{D} \stackrel{\text{def}}{=} \{d_1, \ldots, d_t\} \subset \mathbb{R}^n$, and constants $\alpha^0 > 0$, $c > 0$
- If $f(x^k + \alpha d_i) < f(x^k) - c(\alpha^k)^2$ for some i,
  - then set $x^{k+1} = x^k + \alpha^k d_i$
  - otherwise $x^{k+1} = x^k$, $\alpha^{k+1} \leftarrow \alpha^k/2$

Gradient Descent (First-order method)

$$x^{k+1} = x^k - \alpha^k \nabla f(x^k)$$

Accelerated Gradient Descent (First-order method)

$$y^{k+1} = x^k - \alpha^k \nabla f(x^k), \quad x^{k+1} = (1 + \beta^k) y^{k+1} - \beta^k y^k$$

Newton's Method (Second-order method)

$$x^{k+1} = x^k - \alpha^k \left(\nabla^2 f(x^k)\right)^{-1} \nabla f(x^k)$$

Complexity of Basic Methods I

Consider the least-squares function

$$f(x) = \frac{1}{2}\|Ax - b\|^2,$$

where $A \in \mathbb{R}^{m \times n}$ is a data matrix, $x \in \mathbb{R}^n$ is a vector of parameters to be tuned and $b \in \mathbb{R}^m$ is a data vector. Note that

$$\nabla f(x) = A^\top (Ax - b), \qquad \nabla^2 f(x) = A^\top A.$$

Cost of evaluation of $f(x)$:

- Multiply A by x: $O(mn)$
- Subtract b from the result: $O(m)$
- Compute the squared norm of the resulting vector: $O(m)$
- Total cost: $O(mn)$

Cost of evaluation of $\nabla f(x)$:

- Multiply A by x: $O(mn)$

Complexity of Basic Methods II

- Subtract b from the result: $O(m)$
- Multiply the result by $A^\top$: $O(mn)$
- Total cost: $O(mn)$

Cost of evaluation of $(\nabla^2 f(x))^{-1}\nabla f(x)$:

- Form the matrix $\nabla^2 f(x) = A^\top A$: $O(mn^2)$
- Form the gradient $\nabla f(x)$: $O(mn)$
- Solve the $n \times n$ linear system $\nabla^2 f(x)h = \nabla f(x)$ using a direct method (e.g., Gaussian elimination): $O(n^3)$
- Total cost: $O(n^3 + mn^2)$

Convergence

What Quantity Do We Want to Converge?

Let x^* be a solution we are interested in (global, local, feasible, ...).

- If the method is **deterministic,** we may wish one of the following quantities to converge to zero:
  - Squared distance to a solution: $a_k \overset{\text{def}}{=} \|x^k - x^*\|^2 \to 0$
  - Function values: $a_k \overset{\text{def}}{=} f(x^k) - f(x^*) \to 0$
  - Squared Norm of the gradient: $a_k \overset{\text{def}}{=} \|\nabla f(x^k)\|^2 \to 0$
- If the method is **randomized,** we may wish one of the following quantities to converge to zero:
  - Expected squared distance to a solution: $a_k \overset{\text{def}}{=} \mathbf{E}[\|x^k - x^*\|^2] \to 0$
  - Expected function values: $a_k \overset{\text{def}}{=} \mathbf{E}[f(x^k) - f(x^*)] \to 0$
  - Expected squared gradient norm: $a_k \overset{\text{def}}{=} \mathbf{E}[\|\nabla f(x^k)\|^2] \to 0$

The above are just some of the most common measures of success, several other quantities are studied as well.

The expectation above is with respect to the coin flips / randomness inherent in the randomized algorithm.

Typical Convergence Rates Encountered in Optimization

Bound on a_k	Bound on k implying $a_k \leq \epsilon$	Typical Recursion	Speed
$a_k \leq \frac{C}{\sqrt{k}}$	$k \geq \frac{C}{\epsilon^2}$		slow
$a_k \leq \frac{C}{k}$	$k \geq \frac{C}{\epsilon}$	$a_{k+1} \leq a_k - \frac{a_k^2}{C}$	ok
$a_k \leq \frac{C}{k^2}$	$k \geq \sqrt{\frac{C}{\epsilon}}$		fast
$a_k \leq a_0 e^{-k/C}$	$k \geq C \log\left(\frac{a_0}{\epsilon}\right)$	$a_{k+1} \leq \left(1 - \frac{1}{C}\right) a_k$	very fast

Recursions & Associated Convergence Rates I

Proposition 1

Let $\{a_k\}_{k=0}^{\infty}$ be a sequence of positive real numbers.

1. *Assume that the recursion $a_{k+1} \leq \left(1 - \frac{1}{C}\right) a_k$ holds for all $k \geq 0$ and some constant $C > 1$. Then*

$$a_k \leq \left(1 - \frac{1}{C}\right)^k a_0 \leq e^{-k/C} \cdot a_0.$$

2. *Assume that the recursion $a_{k+1} \leq a_k - \frac{a_k^2}{C}$ holds for all $k \geq 0$ and some constant $C > 0$. Then for all $k \geq 0$, we have*[1]

$$a_k \leq \frac{Ca_0}{C + a_0 k} \leq \frac{C}{k}.$$

[1] The second inequality only makes sense for $k \geq 0$

Proof of Proposition 1 - I

1. By applying the recurrence repeatedly, we get

$$a_k \leq \left(1 - \frac{1}{C}\right) a_{k-1} \leq \cdots \leq \left(1 - \frac{1}{C}\right)^k a_0.$$

Next, we can use the approximation

$$\left(1 - \frac{1}{C}\right)^k = \left[\left(1 - \frac{1}{C}\right)^C\right]^{k/C} \leq \left(e^{-1}\right)^{k/C} = e^{-k/C}.$$

Proof of Proposition 1 - II

2. Note that

$$\frac{1}{a_{k+1}} - \frac{1}{a_k} = \frac{a_k - a_{k+1}}{a_k a_{k+1}} \geq \frac{a_k - a_{k+1}}{a_k^2} \geq \frac{1}{C},$$

where the second inequality follows from the recursion. Therefore, the quantity $\frac{1}{a_k}$ grows at each step at least by $1/C$. By adding up the contributions, this implies that

$$\frac{1}{a_k} \geq \frac{1}{a_0} + \frac{k}{C}.$$

Therefore,

$$a_k \leq \left(\frac{1}{a_0} + \frac{k}{C}\right)^{-1} \leq \left(\frac{k}{C}\right)^{-1} = \frac{C}{k}.$$

Randomized Decomposition

Projected Gradient Descent

Definition 6 (Euclidean Projection)

The **projection** of $y \in \mathbb{R}^n$ onto Q is defined as the closest point in Q to y, i.e.,

$$\Pi_Q(y) = \arg\min_{x \in Q} \|x - y\|.$$

Projected Gradient Descent

Consider optimization problem (1):

$$\text{minimize} \quad f(x) \quad \text{subject to} \quad x \in Q \subseteq \mathbb{R}^n.$$

Projected gradient descent (PGD) is defined via:

$$x^{k+1} = \Pi_Q\left(x^k - \alpha^k \nabla f(x^k)\right).$$

Remark: PGD is a generalization of GD to constrained optimization. Indeed, if $Q = \mathbb{R}^n$, we recover GD as a special case of PGD.

Big Data Optimization I

Big Data Optimization

When (1) is described by **big amounts of data,** then any (or both) of the following can be true:

- it is complicated to evaluate $\nabla f(x)$
- it is complicated to project onto Q

We can use the following tricks to get around this:

1. **Work in a random subspace of $\mathbb{R}^n$.** If n is big, a good idea is to replace, at iteration k, function f with the function

$$\phi^k(y) \stackrel{\text{def}}{=} f(x^k + S^k y),$$

where $S^k \in \mathbb{R}^{n \times a}$ is a random matrix with much fewer columns than n (i.e., $a \ll n$).

   - That is, we are constraining our attention to directions living in (at most) a dimensional random subspace of $\mathbb{R}^n$ spanned by the columns of S^k.

Big Data Optimization II

- The idea is that the gradient of ϕ^k, given by

$$\nabla \phi^k(y) = (S^k)^\top \nabla f(x^k + S^k y),$$

can be much easier to compute than the gradient of f, which might significantly reduce the cost of one iteration of PGD.

2. **Work with a random estimate of f.** In data science, objective functions are often of the form

$$f(x) = \frac{1}{b} \sum_{i=1}^{b} f_i(x),$$

where f_i typically represents loss related to data point i, and f represents the average loss across the entire data set. The idea is to estimate the gradient $\nabla f(x)$ by $\nabla f_i(x)$, where i is a random index chosen uniformly at random.

- $\nabla f_i(x)$ is a **stochastic gradient** of f
- $\nabla f_i(x)$ is an **unbiased estimator** of $\nabla f(x)$:

$$\mathbf{E}[f_i(x)] = \nabla f(x)$$

Big Data Optimization III

- **The (expected) cost of computing $\nabla f_i(x)$ is b times lower than the cost of evaluating $\nabla f(x)$.**

3. **Work with a random estimate of Q.** It is often the case that the constraint Q arises as an intersection of a (very) large number of simpler constraints:

$$Q = Q_1 \cap Q_2 \cap \cdots \cap Q_c$$

In iteration k, choose an index $i \in \{1, 2, \ldots, c\}$ uniformly at random, and use Q_i in place of Q in an algorithm.

Convexity

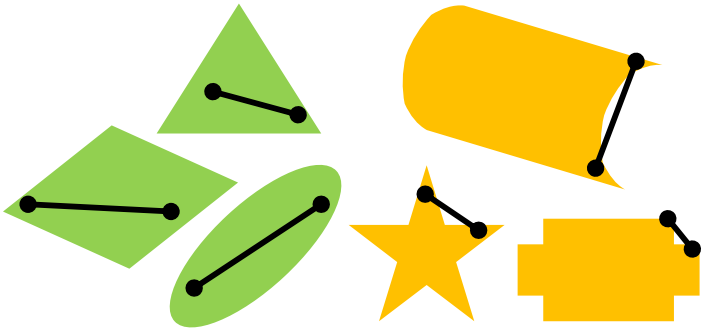
Convex Sets

Definition 7 (Convex sets)

A set $Q \subseteq \mathbb{R}^n$ is **convex** if for all $\alpha \in [0,1]$ and $x, y \in Q$ we have

$$\alpha x + (1-\alpha)y \in Q.$$

That is, Q is convex if with every two points it also contains the line segment joining them.



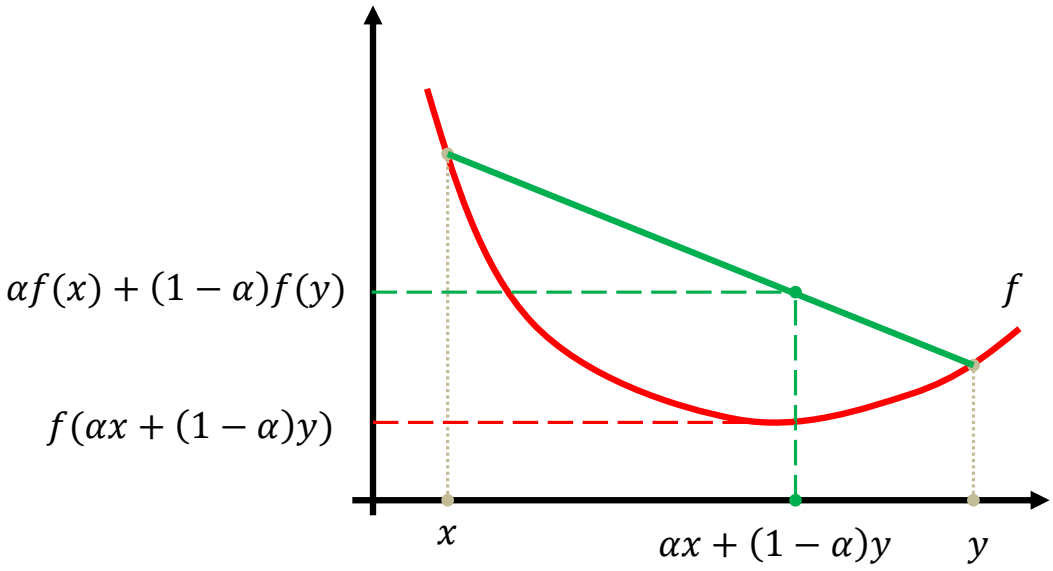
convex non-convex

Convex Functions

Definition 8 (Convex functions)

Let $Q \subseteq \mathbb{R}^n$ be a convex set. A function $f : Q \to \mathbb{R}$ is **convex** on Q if for all $x, y \in Q$ and $0 < \alpha < 1$, we have

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



Convex Functions: The Basics

Exercise 1

Show, from the definition, that if functions $f_1, \ldots, f_k : Q \to \mathbb{R}$ are convex, and $c_1, \ldots, c_k$ are nonnegative scalars, then

1. $f(x) = \sum_i c_i f_i(x)$ *is convex*
2. $f(x) = \max_i f_i(x)$ *is convex*

Theorem 9 (Sufficient conditions for convexity)

Let Q be an open convex set.

1. **First-order sufficient condition.** *Assume $f : Q \to \mathbb{R}$ is continuously differentiable. Then f is convex on Q if and only if*

$$f(y) \geq f(x) + \langle \nabla f(x), y - x \rangle, \qquad \forall x, y \in Q.$$

2. **Second-order sufficient condition.** *Assume $f : Q \to \mathbb{R}$ is twice continuously differentiable. Then f is convex on Q if and only if its Hessian matrix $\nabla^2 f(x)$ is positive semidefinite for all $x \in Q$.*

Strong Convexity

Definition 10 (Strong convexity)

Let $Q \subseteq \mathbb{R}^n$ be an open convex set and assume that f is continuously differentiable on Q. Let $\lambda \geq 0$. We way that f is **λ-strongly convex** on Q if

$$f(y) \geq f(x) + \langle \nabla f(x), y - x \rangle + \frac{\lambda}{2}\|y - x\|^2, \qquad \forall x, y \in Q. \tag{2}$$

Exercise 2

Show that if f is λ-strongly convex function for some $\lambda \geq 0$, then it is convex.

Direct Search

Direct Search

Consider an unconstrained version of (1):

$$\min_{x \in \mathbb{R}^n} f(x)$$

We now describe a method which "works" under the assumption that f is "smooth".

- Pick any finite set of directions $\mathcal{D} \stackrel{\text{def}}{=} \{d_1, d_2, \ldots, d_t\} \subset \mathbb{R}^n$.
- The method will only move along these directions, with positive stepsizes.
- In order to ensure that any point is reachable from any starting point by a sequence of steps of such a method, we need to assume that the set of directions forms a **positive spanning set.** That is, we assume that

$$\mathbb{R}^n = \left\{ \sum_{i=1}^{m} s_i d_i \ : \ s_1, \ldots, s_t \geq 0 \right\}.$$

SDS: Simplified Direct Search

SDS Algorithm (Konečný and R. 2014, [1])

Input: initial point $x^0 \in \mathbb{R}^n$
directions $d_1, d_2, \ldots, d_t \in \mathbb{R}^n$
stepsize parameter $\alpha > 0$
forcing (greediness) constant $c > 0$

for $k = 0, 1, 2, \ldots$ **do**

1) If there is a direction d_i which leads to a **sufficient decrease** in the objective function, namely

$$f(x^k + \alpha d_i) < f(x^k) - c\alpha^2,$$

then **take a step:**

$$x^{k+1} = x^k + \alpha d_i$$

2) **Otherwise,** halve the stepsize: $\alpha \leftarrow \frac{\alpha}{2}$

end for

Convergence of SDS

Theorem 11 ([1])

Assume that f is L-smooth, and bounded below by $f^* \in \mathbb{R}$. Let $x^0 \in \mathbb{R}^n$ be any starting point and let $\{x^k\}$ be the iterates produced by SDS.

- *There exists constant $C > 0$ such that for any $\epsilon > 0$*

$$k \geq \frac{C}{\epsilon^2} \quad \Rightarrow \quad \|\nabla f(x^k)\| \leq \epsilon.$$

- *Assume that f is* **convex** *and that it has a minimizer, denoted by x^*. Further, assume that $\sup_{x \in \mathbb{R}^n} \{\|x - x^*\| \ : \ f(x) \leq f(x^0)\}$ is finite. Then there exists constant $C > 0$ such that for any $\epsilon > 0$*

$$k \geq \frac{C}{\epsilon} \quad \Rightarrow \quad f(x^k) - f(x^*) \leq \epsilon.$$

- *If f is* **strongly convex** *(in which case f has a unique minimizer, which we denote x^*), then there exists constant $C > 0$ such that*

$$k \geq C \log\left(\frac{1}{\epsilon}\right) \quad \Rightarrow \quad f(x^k) - f(x^*) \leq \epsilon.$$

Zero-Order Methods: More Reading

We will not deal with zero-order (aka "derivative-free") methods in this course any further. If interested in this topic, read:

A recent survey paper

Methodologies and software for derivative-free optimization
A. L. Custdio, K. Scheinberg, and L. N. Vicente
Chapter 37 of Advances and Trends in Optimization with Engineering Applications, *MOS-SIAM Book Series on Optimization*, SIAM, Philadelphia, 2017
http://www.mat.uc.pt/ lnv/papers/dfo-survey.pdf

Book

Introduction to Derivative-Free Optimization
A. R. Conn, K. Scheinberg, and L. N. Vicente
MPS-SIAM Series on Optimization, SIAM, Philadelphia, 2009

References

[1] Jakub Konečný and Peter Richtárik. Simple Complexity Analysis of Simplified Direct Search, 2014 (*arXiv:1410.0390*)