Numerical optimization for large scale problems and stochastic optimization

Sandra Pieraccini

Dipartimento di scienze Matematiche
Politecnico di Torino
sandra.pieraccini@polito.it

September 25, 2024

Course outline

1. Unconstrained optimization
2. Constrained optimization
3. Stochastic optimization (Prof. Edoardo Fadda)

Outline of Part I: Introduction

1. Problem setting, definitions and some basic concepts
   - Motivation: why we need to solve large scale optimization problems?
   - The problem(s)
   - Convexity
   - Notation and few reminders

Outline of Part II: Unconstrained optimization, basic methods

Outline of Part III: Unconstrained optimization, variants of Newton method

Outline of Part IV: Constrained optimization

[Slides mostly not available]

Bibliography

1. J. Nocedal, S. J. Wright, *Numerical Optimization*, ed. Springer (1999 edition fully available as e-book form the digital library of Politecnico di Torino)
   In the slides I'll point you to the chapters in which the topics are addressed; not all topics discussed in such chapters are discussed during the lectures; refer to the notes for the topics, but use the textbook as a documentation.
2. *Optimization Algorithms for Data Analysis*, Stephen J. Wright, in The mathematics of data, Mahoney, Duchi, Gilbert eds., IAS/PARK CITY Vol. 25, 2018, pp. 49-97

Part I

Introduction - NW, Chapter 1

Motivation: some examples from Data Analysis

- A data set in a typical analysis problem consists in a set D of m objects

$$D = \{(a_j, y_j), j = 1, 2, \ldots, m\}$$

  where a_j is a vector (or matrix) of **features** and y_j is an **observation** or **label**.
- Each pair (a_j, y_j) has the same size and shape for all $j = 1, 2, \ldots, m$.
- The analysis task consists of discovering a function ϕ such that

$$\phi(a_j) \simeq y_j$$

  for most $j = 1, 2, \ldots, m$. The process of discovering the mapping ϕ is often called **learning** or **training**.
- The mapping ϕ can be used to make predictions $y = \phi(a)$ but also for devising information on the dataset (for example, feature selection)

- The function ϕ is often defined in terms of a vector or matrix of parameters x. The problem of identifying ϕ becomes a data-fitting problem:

Find the parameters x defining ϕ such that $\phi(a_j) \simeq y_j$, $j = 1, 2, \ldots, m$ in some optimal sense.

What do we mean for *optimal*? An optimization problem comes into play.

- Typically many optimization formulations introduce an objective function of the type

$$L_D(x) := \sum_{j=1}^{m} \ell(a_j, y_j; x)$$

where the j-th term $\ell(a_j, y_j; x)$ is a measure of the mismatch between $\phi(a_j)$ and y_j , and x is the vector of parameters that determines ϕ. So the problem becomes

$$\min_{x \in \mathcal{X}} L_D(x)$$

where $\mathcal{X}$ is a suitable set depending on the structure of x and on possible constraints on x (e.g., $\mathcal{X} = \mathbb{R}^n$, $\mathcal{X} = \mathbb{R}^n_+$ ...)

Some examples:

- if $y_j \in \mathbb{R}$ we have a regression problem
- y_j may be *labels*, i.e. they assume only few values in a finite, discrete set, e.g. $y_j \in \{1, 2, \ldots, M\}$ indicating that a_j belongs to one of M classes. Then we have a *classification problem*
- if $M = 2$ we have a binary classification
- y_j may be null. Some problems only have feature vectors a_j and no observations. In this case, the data analysis task may consist of grouping the a_j into clusters (where the vectors within each cluster are considered to be functionally similar). Such problems require also the labels y_j to be learned. For example, in a **clustering problem**, y_j could represent the cluster to which a_j is assigned.

Consider the general optimization problem

$$\begin{aligned} \min \quad & f(x) \\ \text{s.t.} \quad & g(x) \leq 0 \\ & h(x) = 0 \end{aligned}$$

with $f : \mathbb{R}^n \mapsto \mathbb{R}$, $g : \mathbb{R}^n \mapsto \mathbb{R}^m$, $h : \mathbb{R}^n \mapsto \mathbb{R}^p$. The features (and existence!) of functions f, g, h characterize the optimization problem, and the methods most suited for its solution.

A few taxonomy:

- unconstrained problems (no g, h)
- constrained problems

- continuous optimization ($x \in \mathbb{R}^n$)
- discrete optimization (some or all the components of x are required to belong to a discrete set; e.g. (Mixed) Integer Programming)

- deterministic optimization (f, g, h are known deterministically)
- stochastic optimization (one or more among f, g, h are affected by

A few taxonomy on constrained problems

1. linear programming (LP): f, g linear
2. quadratic programming (QP): f quadratic, g linear
3. nonlinear programming (NLP): f and/or g are nonlinear

Notation

Given

$$f(x) = x^2 + 1$$

- The minimum is 1:

$$1 = \min_{\mathbb{R}} f(x)$$

- The minimizer is 0:

$$0 = \arg\min_{\mathbb{R}} f(x)$$

Convexity

Definition

A set $C \subset \mathbb{R}^n$ is convex if $\forall x, y \in C$ the segment joining x and y is contained in C:

$$\alpha x + (1 - \alpha) y \in C \qquad \forall \alpha \in (0, 1)$$

Definition

Given a convex set $C \subset \mathbb{R}^n$, a function $f : C \mapsto \mathbb{R}$ is convex if

$$f(\alpha x + (1 - \alpha) y) \leq \alpha f(x) + (1 - \alpha) f(y) \qquad \forall x, y \in C,\ \alpha \in (0, 1)$$

Definition

Given a convex set $C \subset \mathbb{R}^n$, a function $f : C \mapsto \mathbb{R}$ is concave if and only if $-f$ is convex

Recall the following:

Definition

A vector x^* is a **local** minimum of f if $f(x^*) \leq f(x)$ for all feasible x in a neighborhood of x^*.

Definition

A vector x^* is a **global** minimum of f if $f(x^*) \leq f(x)$ for all feasible x.

Property

If f is convex and the set of feasible solutions is a convex set, a local minimum is also a global minimum.

Setting notations

Notations:

Given $f : \mathbb{R}^n \to \mathbb{R}^m$

- If f is vector valued ($m > 1$) we denote by $f_i(x)$ the components of f, $i = 1, ..., n$
- if $\forall i \, f_i(x)$ is continuosly differentiable at $x \in \mathbb{R}^n$, $f(x)$ is said to be continuously differentiable at x
- the matrix whose rows are the gradients of f_i, $i = 1, ..., m$, is called Jacobian (matrix) of f at x and denoted by $f'(x)$:

$$f'(x)_{ij} = \frac{\partial f_i}{\partial x_j}(x)$$

(sometimes also denoted $J(x)$ or $\nabla f(x)^T$)

Vector and matrix norms

Definition

A vector/matrix norm is a function $\|\cdot\| : V \mapsto \mathbb{R}$ ($V = \mathbb{R}^n, \mathbb{R}^{n\times m}$) s.t.

1. $\|x\| \geq 0 \ \forall x \in V$; $\|x\| = 0 \Leftrightarrow x = 0$
2. $\|\alpha x\| = |\alpha|\|x\| \ \forall x \in V, \forall \alpha \in \mathbb{R}$
3. $\|x + y\| \leq \|x\| + \|y\| \ \forall x, y \in V$ (triangle inequality)

Remark: sub-multiplicative property

For matrix norms we also require the additional property

$$\|AB\| \leq \|A\|\|B\| \quad \forall A \in \mathbb{R}^{n\times p}, B \in \mathbb{R}^{p\times m}$$

Most used vector norms:

- $\|x\|_1 = \sum_{i=1}^{n} |x_i|$
- $\|x\|_2 = \left(\sum_{i=1}^{n} x_i^2\right)^{1/2}$ (Euclidean norm)
- $\|x\|_p = \left(\sum_{i=1}^{n} |x_i|^p\right)^{1/p}$ (p-norm)
- $\|x\|_\infty = \max_i |x_i|$ (max norm)

Most used matrix norms:

- $\|A\|_1 = \max_j \sum_{i=1}^{n} |a_{ij}|$
- $\|A\|_2 = \sqrt{\rho(A^T A)}$, $\rho(B) = max_i|\lambda_i(B)|$ (spectral norm)
- $\|A\|_\infty = \max_i \sum_{j=1}^{n} |a_{ij}|$
- $\|A\|_F = \left(\sum_{i=1}^{n} \sum_{j=1}^{n} a_{ij}^2\right)^{1/2}$ (Frobenious norm)

Rate of convergence

Definition

Let $\{x_k\} \subset \mathbb{R}^n$ and $x^* \in \mathbb{R}^n$ and assume $x_k \to x^*$.
Then x_k is said to converge to x^*

1. linearly, with factor $\sigma \in (0,1)$ if for k large enough

$$||x_{k+1} - x^*|| \leq \sigma ||x_k - x^*||$$

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

3. superlinearly with order $\alpha > 1$ if $x_k \to x^*$ and $\exists\ C > 0$ such that

$$||x_{k+1} - x^*|| \leq C||x_k - x^*||^\alpha$$

(quadratically if $\alpha = 2$)

Part II

Unconstrained optimization: basic methods - NW, Chapters 2,3,5

The problem

Consider the problem

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

where $f : \mathbb{R}^n \mapsto \mathbb{R}$ is a smooth nonlinear function.

Characterization of minima

Theorem (First Order necessary conditions)

If x^* is a local minimum and f is continuosly differentiable in an open neighborhood of x^*, then x^* is a stationary point (namely, $\nabla f(x^*) = 0$).

Theorem (Second Order necessary conditions)

If x^* is a local minimum and $\nabla^2 f$ (the Hessian matrix) is continuos in an open neighborhood of x^*, then $\nabla f(x^*) = 0$ and $\nabla^2 f(x^*)$ is positive semidefinite.

Theorem (Second Order sufficient conditions)

Suppose that $\nabla^2 f$ (the Hessian matrix) is continuos in an open neighborhood of x^*, $\nabla f(x^*) = 0$ and $\nabla^2 f(x^*)$ is positive definite. Then x^* is a strict local minimum of f.

Theorem

If f is convex any local minimum is a global minimum. If f is also differentiable, then any stationary point is a global minimum of f.

Nelder & Mead method

Hand-written notes [no slides available]

Iterative methods for linear systems

Quite often, solution of linear systems comes into play.

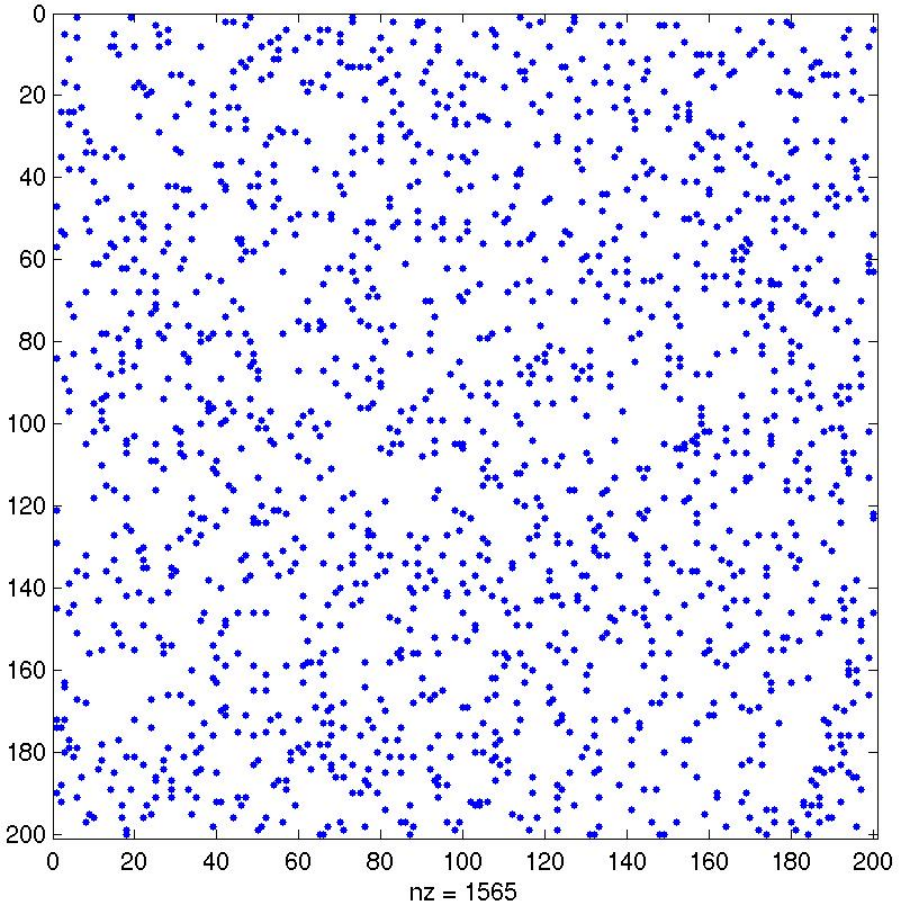
- Effective solution of large scale systems is a must
- Some ideas are common to the solution of nonlinear problems.

Why **iterative methods** for large scale?

Facts:

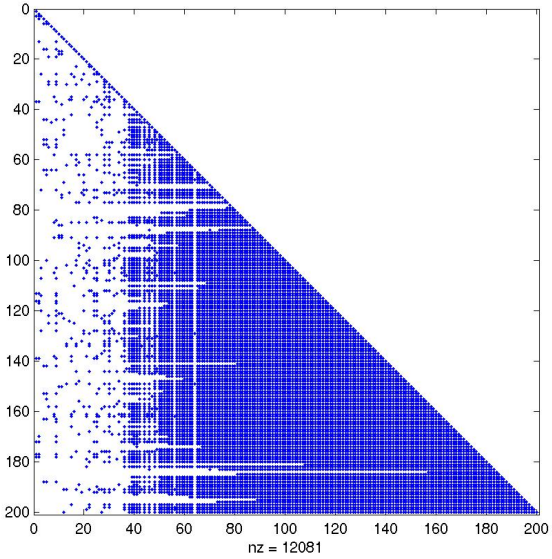
- In large scale linear systems, matrices are frequently **sparse**: few nonzero elements over the total (typically, $\mathcal{O}(n)$ nonzeros over n^2 elements).
- You may exploit sparsity allocating only nonzeros.
- **Direct methods** (such as, e.g., Gauss elimination, Cholesky decomposition...), when applied to large scale problems, are likely to yield the **fill-in** phenomenon.

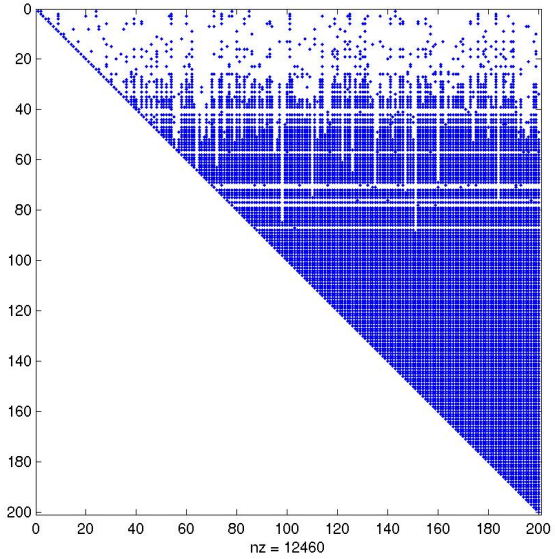
Fill-in phenomenon



Matrix A: 1565 (out of 40000) nonzero elements (less then 4%)

Fill-in phenomenon





Matrices L, U: 12061 and 12460 (out of 40000) nonzero elements (together, more than 60%)

Iterative metodhs

The main idea behind iterative methods if that they **do not modify the structure** of A but, starting from an initial solution $x^{(0)}$, they build a **sequence of approximations** $\{x^{(k)}\}_{k\geq 0}$**, which - under suitable assumptions - converges to** x.

Thus, sparsity is fully exploited.

Stopping criteria

Since an infinite sequence is produced, a stopping criterion is needed in order to stop the iterates.

1. Stop whenever
$$\frac{\|x^{(k+1)} - x^{(k)}\|}{\|x^{(k+1)}\|} \leq \texttt{tol},$$
being `tol` a given tolerance
2. setting $r^{(k+1)} = b - Ax^{(k+1)}$ (residual), stop whenever
$$\frac{\|r^{(k+1)}\|}{\|r^{(0)}\|} \leq \texttt{tol} \qquad \text{or} \qquad \frac{\|r^{(k+1)}\|}{\|b\|} \leq \texttt{tol}$$

No stopping criterion is the perfect one; for example the first one of the two termination criteria based on residual depends on the initial iterate x_0 and may:

1. be too demanding if x_0 is close to the solution (r_0 small)
2. give a poor result if x_0 is far from x (r_0 large)

Remark

Be aware that residual and true error may be poorly correlated, when A is highly ill-conditioned!

Let $\|e^{(k)}\| = \|x^{(k)} - x\|$ denote the error. Then,

$$\frac{\|e^{(k)}\|}{\|x\|} \leq K(A) \frac{\|r^{(k)}\|}{\|b\|}$$

where $K(A)$ is the conditioning number of A. Hence, if

$$\frac{\|r^{(k)}\|}{\|b\|} \leq \texttt{tol} \quad \textit{you are only guaranteed} \quad \frac{\|e^{(k)}\|}{\|x\|} \leq K(A)\texttt{tol}$$

A (very straightforward) remark

The smaller is `tol`...

1. ...the higher is the number of iteration needed to satisfy the criterion...
2. ...and the more accurate we expect the solution.

And viceversa.

This remark is straightforward but crucial to understand the idea behind inexact Newton's methods.

Gradient methods for linear systems

We sketch the main ideas behind the gradient and conjugate gradient methods for linear systems.

- If A is a **symmetric positive definite** matrix, the solution x to the linear system $Ax = b$ is the *unique minimum* of the function

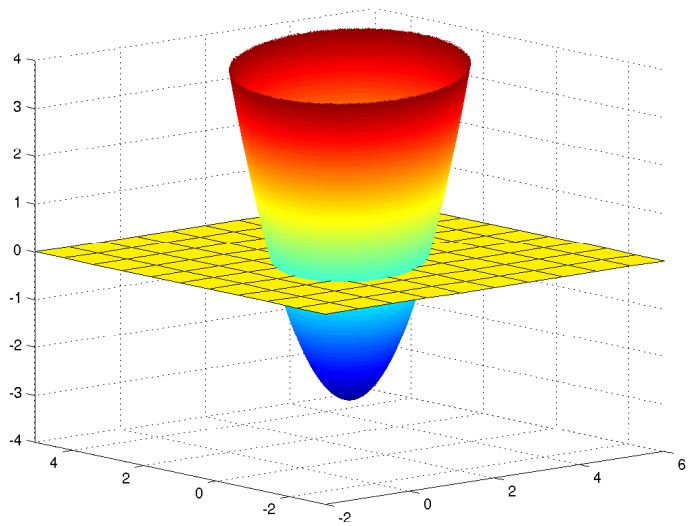
$$J : \mathbb{R}^n \to \mathbb{R}, \qquad J(x) = \tfrac{1}{2} x^T A x - x^T b \,,$$

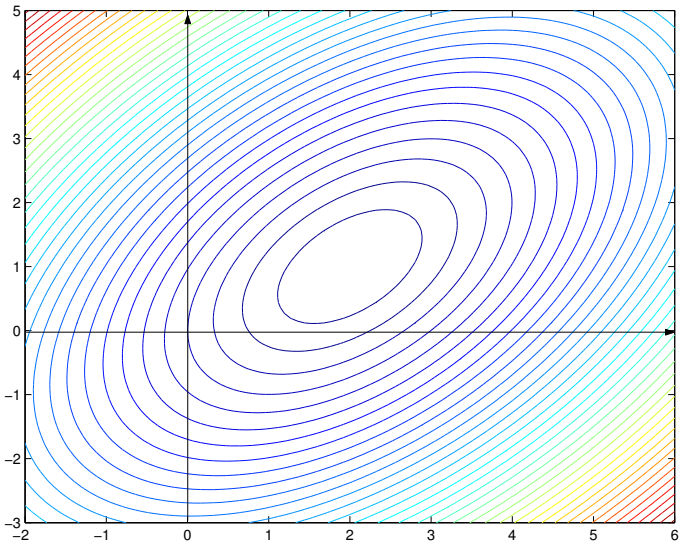
- Indeed, $\nabla J(x) = Ax - b$, and the stationary points are therefore solutions of the linear systems. Since A is SPD, and is the Hessian matrix of J, J is strictly convex, thus there is only a minimum.
- Some iterative methods for solving $Ax = b$ are based on the minimization of J.

Gradient method

Remark

J is a quadratic function. For any point x, the direction $-\nabla J(x)$ is orthogonal to the contour lines and represents the **steepest descent** *direction.*





Gradient method

Idea: build an iterative method by moving, at each step, along the direction $-\nabla J(x_k)$

Note that

$$-\nabla J(x) = b - Ax = r(x)$$

Let $x \in \mathbb{R}^n$ be a fixed point and move along $-\nabla J(x)$ with a steplength α:

$$x \to x - \alpha \nabla J(x) = x + \alpha r(x)$$

$$\begin{aligned} J(x+\alpha r) &= \tfrac{1}{2}(x+\alpha r)^T A(x+\alpha r) - (x+\alpha r)^T b \\ &= \tfrac{1}{2}x^T Ax + \tfrac{1}{2}\alpha^2 r^T Ar + \alpha r^T Ax - x^T b - \alpha r^T b \\ &= J(x) - \alpha r^T r + \tfrac{1}{2}\alpha^2 r^T Ar \end{aligned}$$

- Hence, if α is such that
$$\tfrac{1}{2}\alpha^2 r^T Ar - \alpha r^T r$$
is negative, and with an absolute value as large as possible, we get the highest reduction in the value of J moving along r.
- Since this is just a quadratic function of α, a very simple inspection immediately gives
$$\alpha = \frac{r^T r}{r^T Ar}$$

Gradient method (or steepest descent)

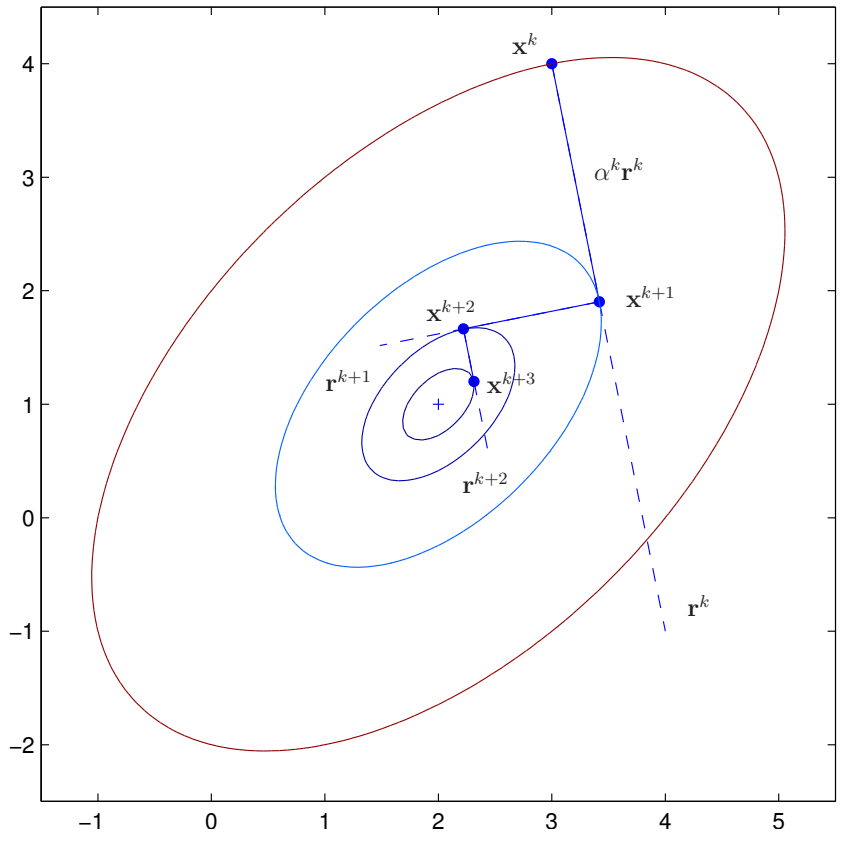
Given x_0, for $k \geq 0$

- Compute $r_k = b - Ax_k$
- Compute $\alpha_k = \frac{r_k^T r_k}{r_k^T A r_k}$
- Update $x_{k+1} = x_k + \alpha_k r_k$

Note that 2 matrix-vector products are performed at each iteration. We can do better:

Given x_0, $r_0 = b - Ax_0$, for $k \geq 0$

- Compute $z_k = Ar_k$
- Compute $\alpha_k = \frac{r_k^T r_k}{r_k^T z_k}$
- Update $x_{k+1} = x_k + \alpha_k r_k$
- Update $r_{k+1} = r_k - \alpha_k z_k$



Convergence properties

Consider the *energy norm* $\|v\|_A = \sqrt{v^T A v}$. Then,

$$\|x_k - x\|_A \leq \left(\frac{K_2(A) - 1}{K_2(A) + 1} \right)^k \|x^0 - x\|_A \ , \qquad k \geq 0.$$

Note also that since $r_{k+1} = r_k - \alpha_k A r_k$,

$$r_{k+1}^T r_k = (r_k - \alpha_k A r_k)^T r_k = r_k^T r_k - \frac{r_k^T r_k}{r_k^T A r_k} r_k^T A r_k = 0$$

Descent methods

In general, a **descent method** is a method with the structure

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

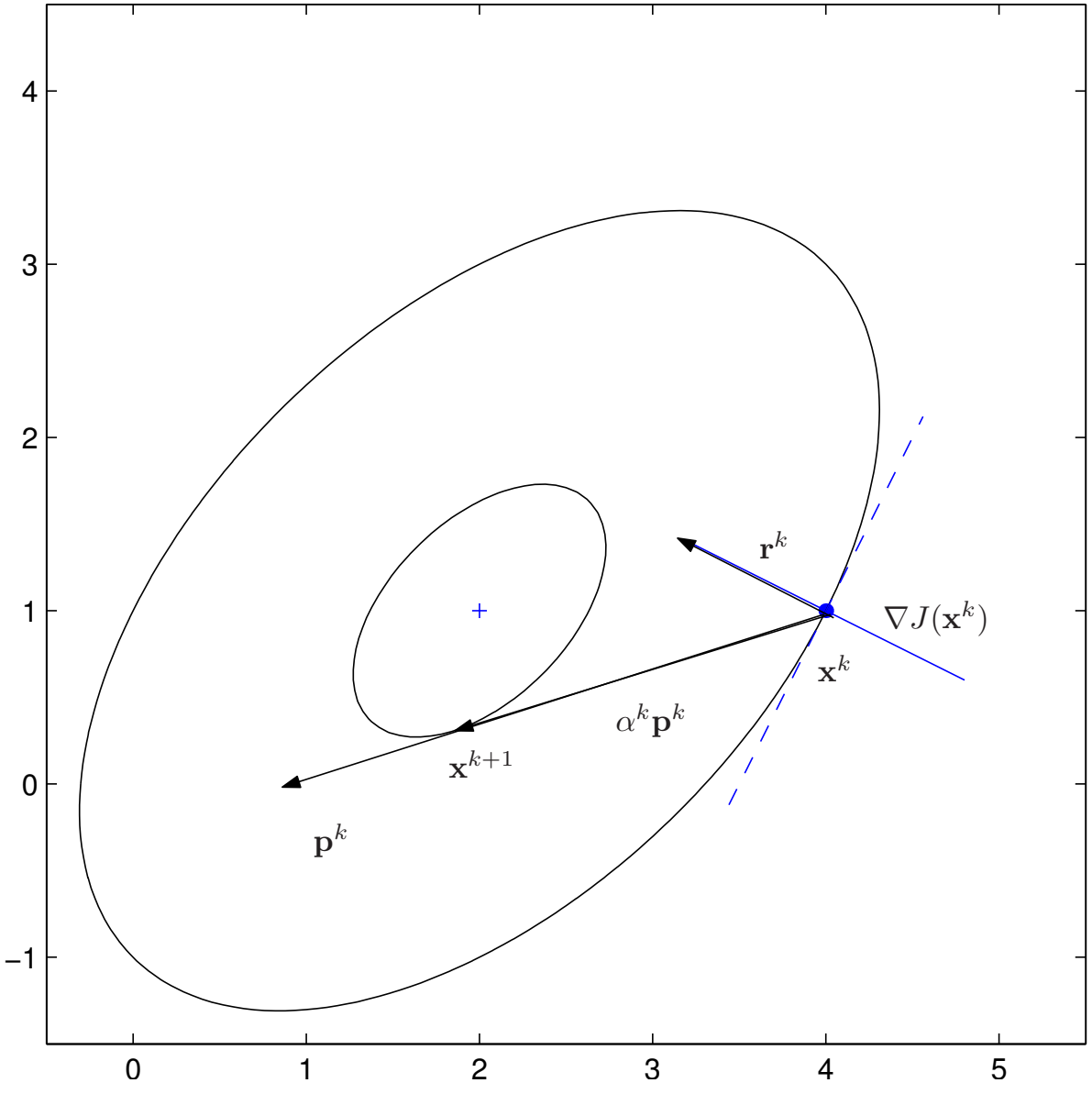
- with p_k a **descent vector**
- α_k a **scalar** representing the **steplength**.

Definition

A vector p_k is a descent vector if, taking a step short enough along thee direction p_k, the function J decreases. This is guaranteed if

$$\nabla J(x_k)^T p_k < 0 \,,$$

(in $\mathbb{R}^2$: p_k and $-\nabla J$ should form an angle $< \frac{\pi}{2}$).


4
3
2
1
0
−1
0
1
2
3
4
5
$\mathbf{r}^k$
$\nabla J(\mathbf{x}^k)$
$\mathbf{x}^k$
$\alpha^k\mathbf{p}^k$
$\mathbf{x}^{k+1}$
$\mathbf{p}^k$

Repeat previous construction with a generic descent direction d:

$$x_{k+1} = x_k + \alpha_k d_k,$$

$$\begin{aligned} J(x_k + \alpha_k d_k) &= \tfrac{1}{2}(x_k + \alpha d_k)^T A(x_k + \alpha_k d_k) - (x_k + \alpha_k d_k)^T b \\ &= \tfrac{1}{2} x_k^T A x_k + \tfrac{1}{2}\alpha^2 d_k^T A d_k + \alpha d_k^T A x_k - x_k^T b - \alpha d_k^T b \\ &= J(x_k) - \alpha_k d_k^T r_k + \tfrac{1}{2}\alpha^2 d_k^T A d_k \end{aligned}$$

with optimal step

$$\alpha_k = \frac{d_k^T r_k}{d_k^T A d_k}$$

How to choose a wise descent direction d_k?

- Recall at each step of the gradient method, given r_k, we minimize $J(x)$ along a line identified by r_k.
- More precisely, we minimize $J(x)$ in a 1D vector subspace spanned by r_k:

$$x^{(k+1)} = \text{argmin} J(x^{(k)} + v), \quad v \in \mathcal{L}(r^{(k)})$$

- We want to improve finding at each step a direction $d_k \neq r_k$ in such a way that we minimize $J(x)$ in a vector subspace with increasing dimension. For given vectors d_k we seek for

$$x_1 = \text{argmin} J(x_0 + v), \quad v \in \mathcal{L}(d_0)$$

$$x_2 = \text{argmin} J(x_0 + v), \quad v \in \mathcal{L}(d_0, d_1)$$

$$\vdots$$

$$x^{(k+1)} = \text{argmin} J(x_0 + v), \quad v \in \mathcal{L}(d_0, \ldots, d_k)$$

- Hence

$$x^{(k+1)} = x_0 + c_0 d_0 + \ldots + c_k d_k$$

with $c = (c_0, \ldots, c_k)$ minimizing

$$J(x_0 + c_0 d_0 + \ldots + c_k d^{(k)})$$

- How can we find $c = (c_0, \ldots, c_k)$ minimizing

$$J(x_0 + c_0 d_0 + \ldots + c_k d_k)?$$

- Necessary condition:

$$\frac{\partial J}{\partial c_j} = 0 \qquad \forall j = 0, \ldots, k$$

$$\rightsquigarrow \qquad \frac{\partial J}{\partial c_j} = {d^{(j)}}^T \nabla J(\underbrace{x_0 + c_0 d_0 + \ldots + c_k d_k}_{x^{(k+1)}}) = {d^{(j)}}^T \nabla J(x^{(k+1)}) = 0$$

- i.e.

$$\nabla J(x^{(k+1)}) \perp \mathcal{L}(d_0, \ldots, d_k)$$

- We analyse the first two steps to see the effects.

$k = 1, 2$:

$$x_1 = x_0 + c_0 d_0, \qquad \nabla J(x_1)^T d_0 = 0$$
$$x_2 = x_1 + c_1 d_1, \qquad \nabla J(x_2)^T d_0 = 0,\ \nabla J(x_2)^T d_1 = 0$$

Recall that $\nabla J(x) = Ax - b$

$$\nabla J(x_2) = Ax_2 - b = A(x_1 + c_1 d_1) - b = \nabla J(x_1) + c_1 A d_1$$

Hence

$$\nabla J(x_2)^T d_0 = \nabla J(x_1)^T d_0 + \alpha_1 (Ad_1)^T d_0$$

and therefore

$$\nabla J(x_2)^T d_0 = 0 \rightsquigarrow d^{(0)T} A d_1 = 0$$

Definition

Given $A \in \mathbb{R}^{n \times n}$ s.d.p., two vectors $p, q \in \mathbb{R}^n$ are said to be A-conjugate directions if

$$q^T A p = 0$$

Generalizing the previous argumetns, we may prove that eevery new direction d_k has to be A-conjugate to the previous one.

Proposition

If vectors $d_0, \ldots, d_k$ form an A-conjugate set, they are also linearly independent.

Conjugate gradient method: descent method with optimal step in which the directions are A-conjugate directions.

- **Cons:** Need to find the conjugate directions
- **Pros:** Finite termination property: the method stops after no more than n steps-

How can we find the a-conjugate directions?

Assume at each step the new direction is a combination of the anti-gradient and the previous direction:

$$d^{(k+1)} = -\nabla J(x^{(k+1)}) + \beta_k d_k$$

It can be proven that β_k can be computed in such a way that $d^{(k+1)}$ is A-conjugate to all previous directions:

$$\beta_k = -\frac{(d_k)^T A r^{(k+1)}}{d_k{}^T A d_k}$$

CG algorithm - naive implementation

Given x^0

Compute $r^0 = b - Ax^0$

Set $d^0 = r^0$

For $k \geq 0$

$$\alpha_k = \frac{r_k^T d_k}{d_k^T A d_k}$$

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

$$r_{k+1} = b - Ax_{k+1}$$

$$\beta_{k+1} = -\frac{(d_k)^T A r_{k+1}}{d_k^T A d_k}$$

$$d_{k+1} = r_{k+1} + \beta_{k+1} d_k$$

CG algorithm

Given x^0

Compute $r^0 = b - Ax^0$

Set $d^0 = r^0$

For $k \geq 0$

$$z_k = Ad_k$$

$$\alpha_k = \frac{r_k^T d_k}{d_k^T z_k}$$

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

$$r_{k+1} = r_k - \alpha_k z_k$$

$$\beta_{k+1} = -\frac{(r_{k+1})^T r_{k+1}}{r_k^T r_k}$$

$$d_{k+1} = r_{k+1} + \beta_{k+1} d_k$$

Convergence properties

Recall that **in principle** CG stops in n iterations, but:

1. if n is very large, this is not such an appealling results
2. in any case, due to aritmetic erros, exact conjugacy among directions is lost.

The method is used as an iterative method.

$$\|x_k - x\|_A \leq 2 \left(\frac{\sqrt{K_2(A)} - 1}{\sqrt{K_2(A)} + 1} \right)^k \|x^0 - x\|_A \,, \qquad k \geq 0 \,.$$

Descent methods

Assume that, starting from x_k, we move along a direction p_k with step α:

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

Consider $f(x_k + \alpha p_k)$ as a function of α. Using Taylor expansion around $\alpha = 0$, we have

$$f(x_k + \alpha p_k) = f(x_k) + \alpha p_k^T \nabla f(x_k) + o(||\alpha||)$$

The factor $p_k^T \nabla f(x_k)$ is the rate of change in f along the direction p_k.
Hence:

1. candidate directions should satisfy $p^T \nabla f(x_k) < 0$ (descent directions);
2. let p_k have a fixed length: the local most rapid decrease is obtained for p_k such that

   $$\min p_k^T \nabla f(x_k) = \min ||p_k|| \, ||\nabla f(x_k)|| \cos(\theta)$$

   which is obviously minimized for $\cos(\theta) = -1$, i.e. for $p = -\nabla f(x_k)$.

Steepest descent method

Analogously to the linear case, the steepest descent method is obtained moving at each iterate along the $-\nabla f(x_k)$ direction:

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

with α to be determined.

Newton method

Again, as in the linear case, steepest descent gives the local most rapid decrease, not necessarily the overall most rapid decrease. Alternate methods use different directions.

The Newton method locally approximates f (sufficiently smooth) with a quadratic model:

$$f(x_k + p) \simeq f(x_k) + p^T \nabla f(x_k) + \frac{1}{2} p^T \nabla^2 f(x_k) p =: m_k(p)$$

If $\nabla^2 f(x_k)$ is positive definite, $m_k(p)$ is a convex model for f around x_k: the Newton step consists in minimizing at each step the local model $m_k(p)$. Computing the stationary point of $m_k(p)$, we obtain

$$\nabla m_k(p) = \nabla f(x_k) + \nabla^2 f(x_k) p = 0$$

i.e. $p = -(\nabla^2 f(x_k))^{-1} \nabla f(x_k)$.

Summarizing:

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

1. steepest descent: $p_k^{SD} = \nabla f(x_k)$
2. Newton method: $p_k^N = -(\nabla^2 f(x_k))^{-1} \nabla f(x_k)$

Pros of Newton method: fast (quadratic) local rate of convergence

Cons of Newton method:

- need $\nabla^2 f(x_k)$ being SPD for p_k^N to be a descent direction ($\rightsquigarrow$ suitable modifications for keeping 2nd order information but ensuring a descent step is taken)
- higher computational cost for assembling hessian matrix and solving the linear systems ($\rightsquigarrow$ Quasi-Newton methods)

Modified Newton method

When $\nabla^2 f(x_k)$ is not positive definite, the Newton direction may not satisfy the descent property $\nabla f(x_k)^T p_k^N < 0$, in which case it is unsuitable as a search direction. A possible idea is to use a modified Hessian matrix

$$B_k = \nabla^2 f(x_k) + E_k$$

where

- B_k is positive definte
- B_k is well-conditioned (required to ensure convergence)
- E_k is as small as possible, so that the second-order information in the Hessian is preserved as far as possible.

Several choices are possible. A possible idea is to take $E_k = \tau_k I$ for a given τ_k.

- Let δ be a small positive number, larger than machine precision, say $\delta = \sqrt{\varepsilon_m}$.
- If $\lambda_{\min}(\nabla^2 f(x_k)) \geq \delta$ than the Hessian is already *sufficiently* positive definite: set $\tau_k = 0$
- Otherwise, take $\tau_k = \delta - \lambda_{\min}(\nabla^2 f_k(x_k))$. Then $\forall i = 1, \ldots, n$

$$\lambda_i(\nabla^2 f(x_k) + \tau_k I) = \lambda_i(\nabla^2 f(x_k)) + \tau_k = \lambda_i(\nabla^2 f(x_k)) - \lambda_{\min}(\nabla^2 f(x_k)) + \delta$$

and so all eigenvalues of $\nabla^2 f(x_k) + \tau_k I$ are larger than δ.

The problem is now how to have a good estimate of $\lambda_{\min}(\nabla^2 f(x_k))$.

Again, several strategies are possible. A rather simple one is based on the following idea: suitably initialize $\tau_k^{(0)}$ and modify it if $\nabla^2 f(x_k) + \tau_k I$ fails to be positive definite (detected by attempting to compute Cholesky decomposition)

Set $A := \nabla^2 f(x_k)$ for the sake of simplicity.

- set $\beta = ||A||_F$ (upper bound of $\max |\lambda_i|$)
- if $\min_i a_{ii} > 0$ then $\tau_k^{(0)} = 0$ (it is possible that A is pos.def.)
  otherwise $\tau_k^{(0)} = \beta/2$ (no way that A is pos.def.)
- for $j = 0, 1, 2, \ldots$
  - Attempt to compute the Cholesky decomposition

$$R^T R = A + \tau_k^{(j)} I$$

if successfull, stop; else set $\tau_k^{(j+1)} = \max(2\tau_k^{(j)}, \beta/2)$

This method has still some drawbacks, as the "final" $\tau_k^{(j)}$ can be unnecessarily large and several Cholesky decompositions may be needed.

Step-length selection

- Once the direction (either p_k^{SD}, or p_k^N, or other directions to be discussed later on) is fixed, how to choose the step length?
- It is clear that not all α values are suitable (think to a paraboloid: if a too long step is taken, the value of f may increase!)
- For Newton method, the $\alpha = 1$ choice is the natural one, but it may fail to be effective when we are far from a solution.

Line search

- Prototype of line search methods: for all k,
  1. Compute a descent direction p_k
  2. Move along the line $x_k + \alpha p_k$, $\alpha > 0$, seeking for a steplength α_k which guarantees a **sufficient decrease** for $f(x_k + \alpha_k p_k)$
  3. Set $x_{k+1} = x_k + \alpha_k p_k$.
- What's to be intended as *sufficient decrease*?
- $f(x_k + \alpha_k p_k) < f(x_k)$ is **not** enough for convergence.

Recall that we will sometimes refer to the function $\phi : \mathbb{R} \mapsto \mathbb{R}$ defined as

$$\phi(\alpha) = f(x_k + \alpha p_k)$$

- If we only ask for $f(x_{k+1}) < f(x_k)$ it is possible that we achieve at each step a very small decrease in f, relative to the steplengths taken
- The fix to this problem is to ask for an average decrease from $f(x_k)$ to $f(x_{k+1})$ which is at least some prescribed fraction of the initial rate of decrease in that direction, i.e.

$$\phi'(0) = \nabla f(x_k)^T p_k$$

- Indeed, the initial rate of decrease represents the potential of the direction p_k, so we are simply asking to satisfy our expectations on p_k.

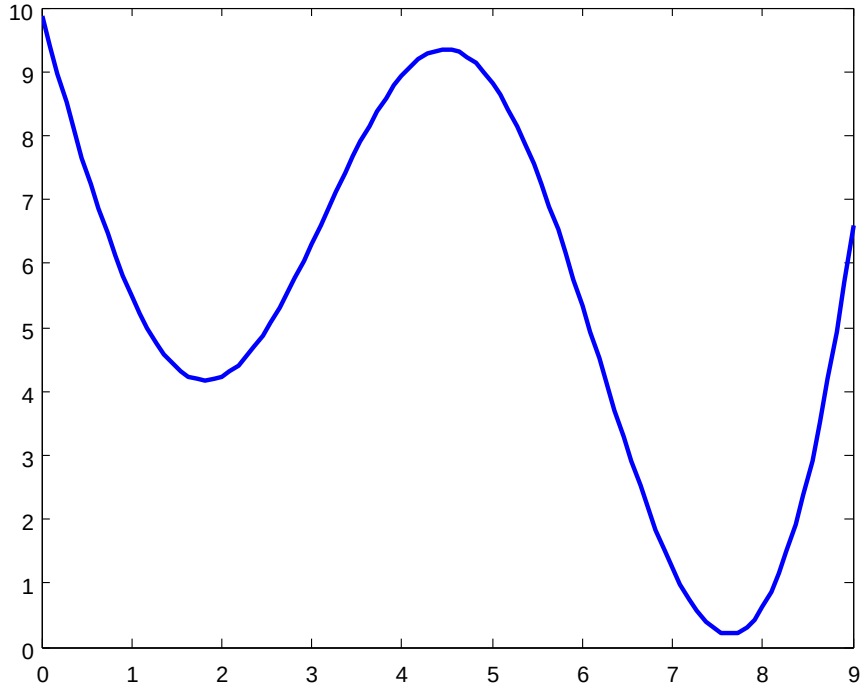
- A popular condition is the following:

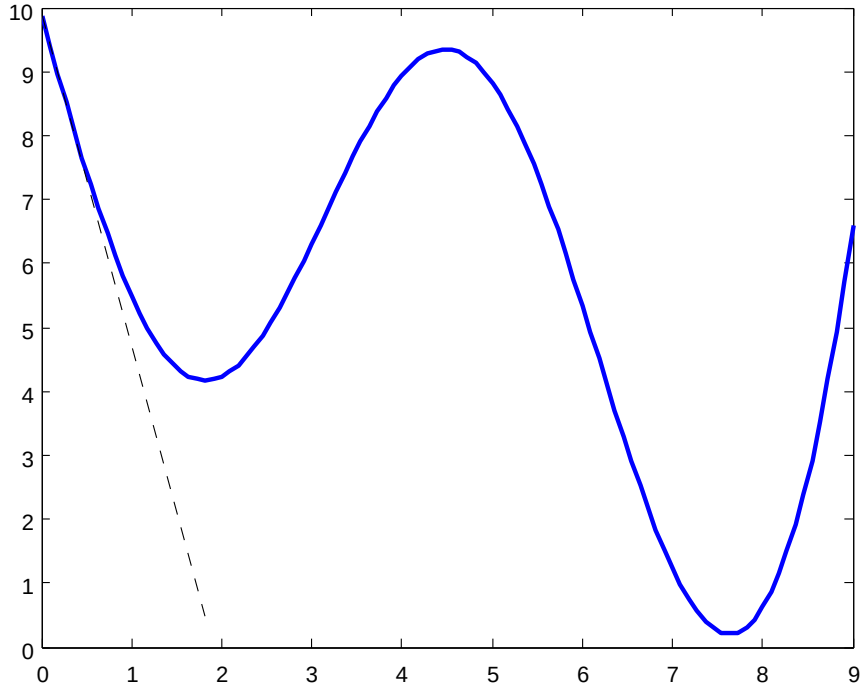
Armijo condition

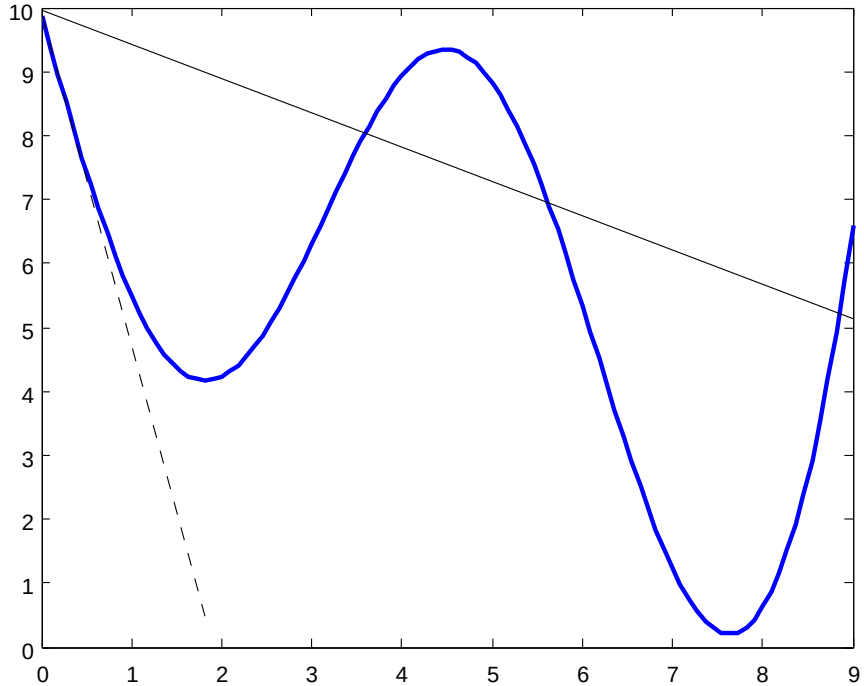
$$f(x_k + \alpha p_k) \leq f(x_k) + c_1 \alpha \nabla f(x_k)^T p_k$$

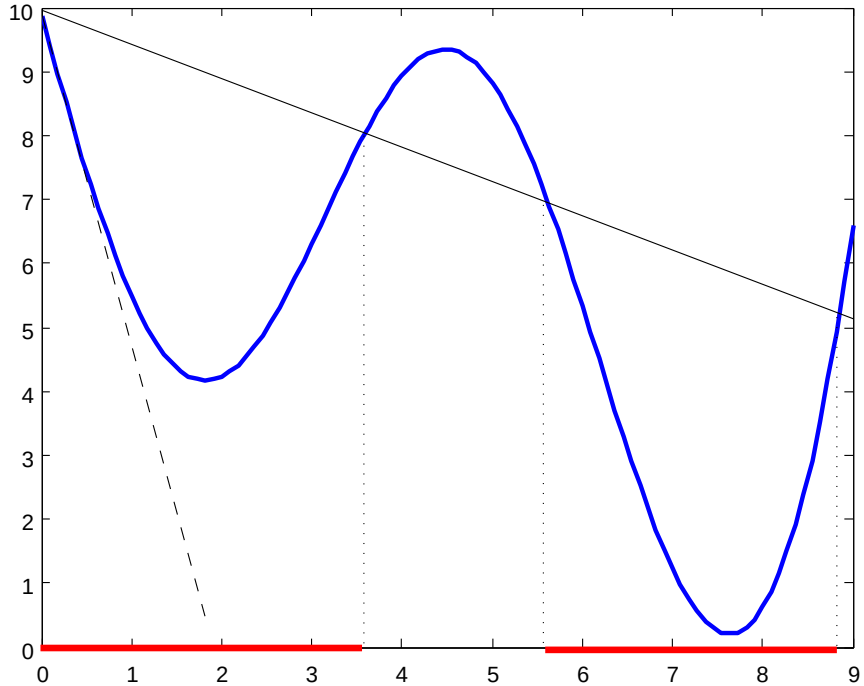
with constant $c_1 \in (0, 1)$

- c_1 is usually very small (typically $c_1 = 10^{-4}$) intended to make Armijo condition easy to satisfy.


10
9
8
7
6
5
4
3
2
1
0
0
1
2
3
4
5
6
7
8
9


10
9
8
7
6
5
4
3
2
1
0
0
1
2
3
4
5
6
7
8
9


10
9
8
7
6
5
4
3
2
1
0
0
1
2
3
4
5
6
7
8
9


10
9
8
7
6
5
4
3
2
1
0
0
1
2
3
4
5
6
7
8
9

- Armijo condition is not enough to ensure that algorithm makes reasonable progress: it is satisfied for very small values of α.
- If α is taken too small, unacceptably short steps are taken.
- Other conditions could be imposed to guarantee that not too short steps are taken (e.g., Wolfe condition)
- Practical implementations instead of imposing a second condition frequently use *backtraking*

Backtracking

- In actual implementations, it is possible to use only Armijo condition if we choose α appropriately, i.e. using a **backtracking strategy**

Backtracking

1. Choose an initial step length $\alpha_k^{(0)}$
2. For $j \geq 0$, if Armijo condition is satisfied, accept $\alpha_k^{(j)}$, otherwise set $\alpha_k^{(j+1)} = \rho \alpha_k^{(j)}$ with a given $\rho \in [\rho_L, \rho_U]$, $\rho_U < 1$

- An acceptable step is always found as eventually $\alpha_k^{(j)}$ becomes small enough.
- On the other hand, by starting from "large" values and reducing only if needed, we ensure to take large steplenghts whenever possible

Some remarks on backtracking:

- Choice of $\alpha_k^{(0)}$ is method-dependent, but it is crucial to choose $\alpha_k^{(0)} = 1$ in Newton/Quasi-Newton methods
- ρ can be a fixed parameter smaller than 1, or can be chosen by interpolation, using already available information

Line search as globalization strategy

A remarkable application of line search strategies is related to the solution of systems of nonlinear equations

$$F(x) = 0, \qquad F : \mathbb{R}^n \mapsto \mathbb{R}^n$$

A popular method to solve this system of equations is Newton method for nonlinear equations: given $x_0 \in \mathbb{R}^n$, compute for $k \geq 1$

$$x_{k+1} = x_k - F'(x_k)^{-1} F(x_k)$$

Under suitable assumptions, the method displays a fast rate of convergence (quadratic) when it is close to a solution, but it suffers a lot from the choice of the initial guess x_0. If a good initial guess is not used, the method is likely to be not able to converge.

- Globalizing strategies are methods specifically designed to help Newton's method when we are far from the solution: typicall slow methods but very robust.
- They are usually designed in such a way that they work only when needed: if close enough to x^*, they are switched off so that Newton's works and eventually fast convergence is maintained.

- The main strategies typically used are essentially strategies based on methods for the minimization of the **merit function**

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

- Note that if x^* solves $F(x) = 0$, it also solves (1). Viceversa, a *global* solution to (1) also solves $F(x) = 0$ (if a solution exists!). Not all local solutions of (1) solves $F(x) = 0$!
  1. **line search** strategies: fixed the direction p_k of movement, we decide length of step along p_k
  2. **trust region** methods: fixed the maximum step length we allow to take, chose a direction which improves the iterations within the steplenght fixed

Nonlinear conjugate method: Fletcher & Reeves method

Fletcher and Reeves extended the conjugate gradient method to a nonlinear functon f with few simple modifications to the CG method:

1. steplength α computed with a line search
2. residual $r = b - Ax$ replaced by ∇f

FR-CG method

Given x^0

Compute $f_0 = f(x_0), \nabla f_0 = \nabla f(x_0)$

Set $p^0 = -\nabla f_0$

For $k \geq 0$

Compute α_k

Compute $x^{k+1} = x^k + \alpha^k p^k$

Compute ∇f_{k+1}

$$\beta^{k+1} = \frac{\nabla f_{k+1}^T \nabla f_{k+1}}{\nabla f_k^T \nabla f_k} = \frac{||\nabla f_{k+1}||^2}{||\nabla f_k||^2}$$

$$p^{k+1} = -\nabla f_{k+1} + \beta^{k+1} p^k$$

It can be shown that, in order to ensure that p_k is a descent direction, it is sufficient to choose α with a line-search strategy as described in previous lectures

Nonlinear conjugate method: Polak-Ribière method

Next to FR-CG, other variants have been proposed in which the main difference relies in the choice of β_k. The Polak-Ribière variant in particular takes

$$\beta^{k+1} = \frac{\nabla f_{k+1}^T(\nabla f_{k+1} - \nabla f_k)}{\nabla f_k^T \nabla f_k}$$

(PR-CG method).

It can be proven that the two methods do coincide for f strongly convex and exact linesearch, but for purely nonlinear functions and inexact line search they behave quite differently. In particular, PR-CG is **more robust** (i.e., less failures) and preforms better, in practice, even if weaker convergence results than for FR-CG can be proven.

Part III

Unconstrained optimization: variants of Newton method - NW, Chapters 6,7

Application of Newton method requires the solution, at each iteration, of the linear system

$$\nabla^2 f(x_k)p = -\nabla f(x_k)$$

in order to compute the Newton direction. The computation of the Newton step may be critical in several respects:

1. assembling $\nabla^2 f(x_k)$ is costly
2. solving the linear system is costly
3. what if $\nabla f^2(x_k)$ is not SPD? (p is not a descent direction!)

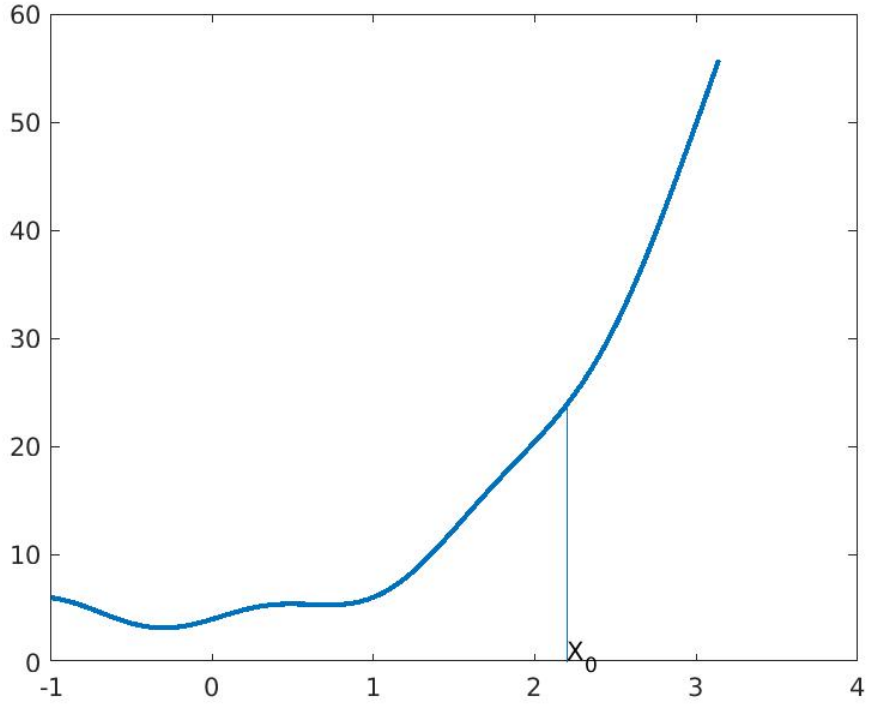
Variants to the Newton method

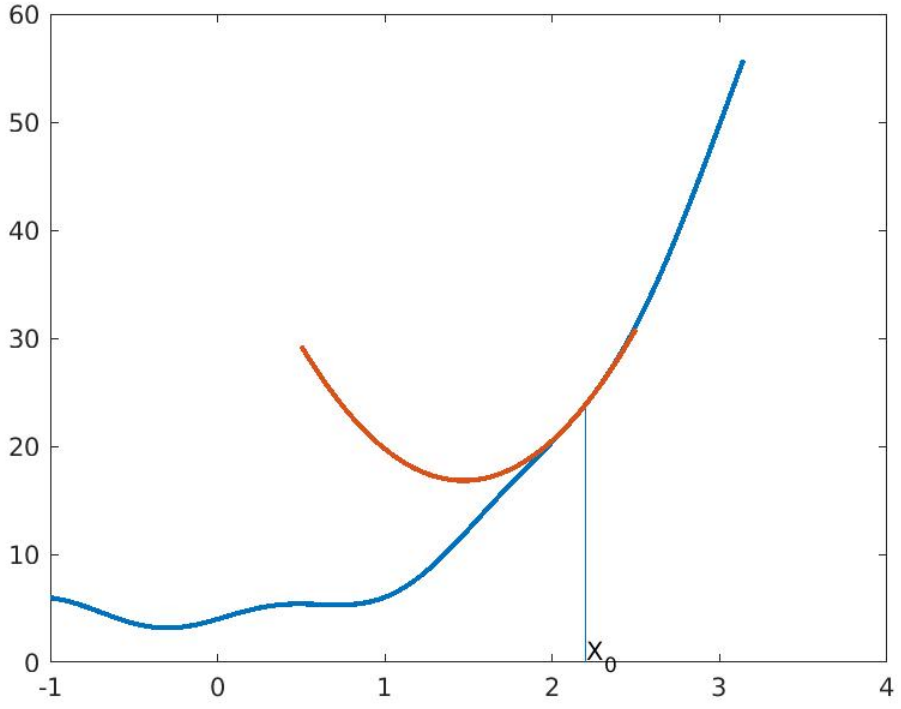
Several methods have been defined in order to improve the computational efficiency of the Newton method while retaining (as much as possible) the convergence properties.

1. Inexact Newton methods
2. Finite-difference and quasi-Newton methods

Inexact Newton method

- Exact solution of linear system $\nabla^2 f(x_k)p = -\nabla f(x_k)$ may require a large computational effort, for large scale problems.
- When x_k is far from the solution, the computational effort required for computing the **exact** Newton's step may be not completely justified: the quadratic model $m_k(p)$ does not push yet towards a solution. This situation is called **oversolving**


60
50
40
30
20
10
0
-1
0
1
2
3
4
x_0


60
50
40
30
20
10
0
-1
0
1
2
3
4
x_0

- A strategy to avoid oversolving: do not try to compute an *exact* Newton step p_k^N by *exactly* solving the linear system...

$$\nabla^2 f(x_k)p + \nabla f(x_k) = 0$$

- ...but rather an *inexact* Newton step p_k^{IN} ensuring

$$\|\nabla^2 f(x_k)p + \nabla f(x_k)\| \leq \text{ a "small" quantity}$$

- In practice, we don't ask the residual to be zero but we want to properly control it.
- How small we want the residual? We ask the following:
  1. When the gradient $\nabla f(x_k)$ is large (i.e. we are far from a stationary point) we allow the **linear** residual to be large
  2. As far as $\nabla f(x_k)$ gets smaller, we ask also the linear residual to be smaller and smaller

- The ultimate target is to reduce the overall computational cost.
- This is accomplished if the cost for solving the linear systems is inversely proportional to the linear residual allowed:
  1. the larger we allow the residual, the cheaper should be the computation of the (approximate) solution
  2. the smaller we ask the residual, the more expensive should be the computation of the (approximate) solution
- A natural way to reach this target is to use iterative solvers for computing the solution of the linear systems, with a stopping criterion which uses a **variable tolerance**: for each linear system to be solved, how large/small is the tolerance, should depend on the present situation.

Inexact Newton's Method (INM)

For $k \geq 0$, set $x_{k+1} = x_k + p_k^{IN}$ where p_k^{IN} satisfies

$$\|\nabla^2 f(x_k)p + \nabla f(x_k)\| \leq \eta_k \, |\nabla f(x_k)\|, \quad \eta_k \in [0, \hat{\eta}), \, \hat{\eta} < 1$$

- η_k are called *forcing terms*: their choice is crucial both for rate of convergence and practical behaviour
- With this approach:
  1. when $\|\nabla f(x_k)\|$ is large the approximation of the Newton step is rough but the cost is low.
  2. As far as we approach a solution and $\|\nabla f(x_k)\|$ gets small, we solve the system with increasing accuracy and increasing computational effort.
  3. A large computational effort is required only at final iterations.

Convergence properties

Theorem

Suppose that f is continuously differentiable in a neighborhood of a minimum x^*, and assume that $\nabla^2 f(x)$ is positive definite. Consider the iteration $x_{k+1} = x_k + p_k$ generated by INM, and assume that $\eta_k \leq \hat{\eta}$ for some constant $\hat{\eta} \in [0, 1)$. Then, if the starting point x_0 is sufficiently near x^*, the sequence $\{x_k\}$ converges to x linearly.

Convergence properties

Theorem

Under the same assumptions of previous Theorem, consider the iteration $x_{k+1} = x_k + p_k$ *generated by INM, and assume that the iterates* $\{x_k\}$ *converge to* x^*. *Then the rate of convergence is superlinear if* $\eta_k \to 0$ *and quadratic if* $\eta_k = \mathcal{O}(\nabla f(x_k))$.

Possible choices:

1. $\eta_k = \min(0.5, \sqrt{\|\nabla f(x_k)\|})$ for superlinear
2. $\eta_k = \min(0.5, \|\nabla f(x_k)\|)$ for quadratic

When choosing forcing terms recall that not always the more aggressive choice yield the better overall behaviour. In fact, (see [Eisenstat,Walker,1996])

1. aggressive ghoices of η_k (i.e. yielding smaller forcing terms/faster asymptotic convergence) may decrease the total number of Newton steps (*outer iterations*) but may yield oversolving, i.e. increase number of linear iterations (*inner iterations*), and may also yield less robustness
2. and viceversa: less aggressive choices may reduce number of inner iterations and improve robustness but also increase number of Newton steps.

Note that previous results are **local** ("if the starting point is sufficiently near...").

In order to obtain global convergence, the method can be combined with a backtracking line search strategy, activated with $\alpha_k^{(0)} = 1$, in such a way that whenever possible the "pure" IN step is taken.

This way, the overall method has good both local and global convergence properties.

Gradients and/or Hessian matrices come often into play.

In several situations, you may want to approximate derivatives, or to compute them automatically:

1. f not known in analitic form; we only are given a *black-box* which evaluates f at given point
2. f is known, but derivatives computation is quite costly

We will focus on finite differencing.

Approximating first order derivative

$f'(x_0) = ?$

Approximate f around x_0 with a 1st order Taylor polynomial with Lagrange reminder:

$$f(x) = f(x_0) + f'(x_0)(x - x_0) + \frac{1}{2}f''(\xi)(x - x_0)^2$$

Let $x = x_0 + h$

$$f(x_0 + h) = f(x_0) + f'(x_0)h + \frac{1}{2}f''(\xi_+)h^2$$

$$f'(x_0) = \frac{f(x_0 + h) - f(x_0)}{h} - \frac{1}{2}f''(\xi_+)h$$

(forward finite differences)

Let $x = x_0 - h$

$$f(x_0 - h) = f(x_0) - f'(x_0)h + \frac{1}{2}f''(\xi_-)h^2$$

$$f'(x_0) = \frac{f(x_0) - f(x_0 - h)}{h} + \frac{1}{2}f''(\xi_-)h$$

(backward finite differences)

Approximate f around x_0 with a 2nd order Taylor polynomial with Lagrange reminder:

$$f(x) = f(x_0) + f'(x_0)(x - x_0) + \frac{1}{2}f''(x_0)(x - x_0)^2 + \frac{1}{6}f'''(\xi)(x - x_0)^3$$

$$f(x_0 + h) = f(x_0) + f'(x_0)h + \frac{1}{2}f''(x_0)h^2 + \frac{1}{6}f'''(\xi_+)h^3$$

$$f(x_0 - h) = f(x_0) - f'(x_0)h + \frac{1}{2}f''(x_0)h^2 - \frac{1}{6}f'''(\xi_-)h^3$$

Subtracting:

$$f'(x_0) = \frac{f(x_0 + h) - f(x_0 - h)}{2h} + \frac{1}{6}(f'''(\xi_+) + f'''(\xi_-))h^2$$

(centered finite differences)

Choosing h

- large h: high analytical error (poor accurate estimate)
- small h: numerical cancellation

Best trade-off: $h \simeq \sqrt{\varepsilon_m}|x_0|$

Approximation of 2nd order derivatives

$f''(x_0) = ?$

Approximate f around x_0 with a 3rd order Taylor polynomial with Lagrange reminder:

$$f(x) = f(x_0) + f'(x_0)(x-x_0) + \frac{1}{2}f''(x_0)(x-x_0)^2 + \frac{1}{6}f'''(x_0)(x-x_0)^3 + \frac{1}{24}f^{(4)}(\xi)(x-x_0)^4$$

$$f(x_0+h) = f(x_0) + f'(x_0)h + \frac{1}{2}f''(x_0)h^2 + \frac{1}{6}f'''(x_0)h^3 + \frac{1}{24}f^{(4)}(\xi_+)h^4$$

$$f(x_0-h) = f(x_0) - f'(x_0)h + \frac{1}{2}f''(x_0)h^2 - \frac{1}{6}f'''(x_0)h^3 + \frac{1}{24}f^{(4)}(\xi_-)h^4$$

Summing up:

$$f''(x_0) = \frac{f(x_0+h) - 2f(x_0) + f(x_0-h)}{h^2} + \frac{1}{24}(f''''(\xi_+) + f''''(\xi_-))h^2$$

Finite difference approximations of the gradient

- Consider $f : \mathbb{R}^n \mapsto \mathbb{R}$ and assume you want to approximate $\nabla f(x)$ at a given point x.
- In other words, you need to approximate $\frac{\partial f}{\partial x_i}(x)$ for all $i = 1, \ldots, n$.
- Consider, e.g., the forward finite difference formula obtained in previous slides and apply it to the approximation of $\frac{\partial f}{\partial x_i}(x)$ while "freezing" the unknowns x_j, $j = 1, \ldots, n$, $j \neq i$:

$$\frac{\partial f}{\partial x_i}(x) = \frac{\partial f}{\partial x_i}(x_1, \ldots, x_n) \simeq \frac{f(x_1, \ldots, x_i + h, \ldots, x_n) - f(x_1, \ldots, x_n)}{h}$$

- Note that you may write

$$\begin{aligned}(x_1, \ldots, x_i + h, \ldots, x_n) &= (x_1, \ldots, x_n) + h(0, \ldots, \underbrace{1}_{\text{pos. } i}, \ldots, 0) \\ &= (x_1, \ldots, x_n) + he_i = x + he_i\end{aligned}$$

  where e_i is the i-th vector of che canonical basis in $\mathbb{R}^n$. Then in compact form

$$\frac{\partial f}{\partial x_i}(x) \simeq \frac{f(x + he_i) - f(x)}{h}$$

- Possible variants: backward differences, centered differences

Error estimate

- Taylor formula: if f is twice continuously differentiable,

$$f(x+p) = f(x) + \nabla f(x)^T p + \frac{1}{2} p^T \nabla^2 f(x+tp) p, \quad t \in (0,1)$$

- Let M be such that $\|\nabla^2 f(x)\| \leq M$ in the region of interest and write

$$|f(x+p) - f(x) - \nabla f(x)^T p| \leq \frac{M}{2} \|p\|^2$$

- Now take $p = he_i$ and note that $\nabla f(x)^T e_i = \frac{\partial f}{\partial x_i}(x)$. Replacing in previous inequality and rearranging we have

$$\left| \frac{f(x+he_i) - f(x)}{h} - \frac{\partial f}{\partial x_i}(x) \right| \leq \frac{M}{2} h$$

On a calculator, you have to consider floating point representations of $f(x + he_i)$ and $f(x)$: if u is the machine precision,

$$\tilde{f}(x) = f(x)(1 + \delta_1), \qquad |\delta_1| \leq \mathrm{u}$$

$$\tilde{f}(x + he_i) = f(x + he_i)(1 + \delta_2), \qquad |\delta_2| \leq \mathrm{u}$$

$$\begin{aligned} & \left| \frac{\tilde{f}(x + he_i) - \tilde{f}(x)}{h} - \frac{\partial f}{\partial x_i}(x) \right| \\ = & \left| \frac{f(x + he_i) - f(x)}{h} - \frac{\partial f}{\partial x_i}(x) + \frac{f(x + he_i)\delta_2 - f(x)\delta_1}{h} \right| \leq \frac{M}{2} h + \frac{2N\mathrm{u}}{h} \end{aligned}$$

where N is such that $|f(x)| \leq N$ in the region of interest.

- True error is hence minimized for

$$h^2 = \frac{4N\mathrm{u}}{M}$$

- If the problem is well scaled, N/M if of moderate size. An optimal choice (frequently used in many optimization packages) is therefore

$$h = \sqrt{\mathrm{u}}$$

- Forward difference approximation of $\nabla f(x)$ require $n+1$ function evaluations.
- A possible generalization is given by centered differences:

$$\frac{\partial f}{\partial x_i}(x) \simeq \frac{f(x+he_i) - f(x-he_i)}{2h}$$

- Accuracy increases as the exact error is $O(h^2)$, but cost is $2n$ function evaluations.

Finite difference approximation of Jacobian matrix

- If we need to approximate the Jacobian, we may straightforwardly apply the previous considerations to $F'(x)$ approximating it a column at a time.
- Let $\frac{\partial F}{\partial x_i}(x)$ denote the i-th column of $F'(x)$. We approximate

$$\frac{\partial F}{\partial x_i}(x) \simeq \frac{F(x + he_i) - F(x)}{h}$$

- Cost is again $n + 1$ evaluation of function F
- This cost can be reduced in certain cases
  1. Matrix-free implementations
  2. Sparse Jacobians

Matrix-free implementations

- If n is very large, allocation of $n \times n$ matrix may be prohibitive.
- In some applications allocation of Jacobian matrix is actually **not** needed: what is really needed is the action of the matrix onto vectors.
- This is the case, for example, of many iterative methods for linear systems $Ax = b$ which, given a vector d, only require the result of the matrix-vector product Ad.

  (See lectures on iterative methods for linear systems)

Matrix-free implementations - continued

- Recall the following: for $f : \mathbb{R}^n \mapsto \mathbb{R}$ the directional derivative of f in the direction d is defined as

$$D(f(x); d) := \lim_{h \to 0} \frac{f(x + hd) - f(x)}{h} = \nabla f(x)^T d$$

- Note that $F'(x)d$ is nothing but the vector of directional derivatives of F_i along d:

$$F'(x)d = \begin{pmatrix} \nabla F_1(x)^T d \\ \nabla F_2(x)^T d \\ \vdots \\ \nabla F_n(x)^T d \end{pmatrix}$$

Matrix-free implementations - continued

- Based on previous remarks, if only matrix-vector products $F'(x)d$ are needed, instead of...
  1. approximate (and allocate in matrix A) $\partial F_i/\partial x_j$
  2. compute Ad
- ...we may, given a small h, directly approximate $F'(x)d$ by

$$F'(x)d \simeq \frac{F(x+hd) - F(x)}{h}$$

- This requires no more allocation of a $n \times n$ matrix, bust just writing a function which computes previous expression.

Sparse Jacobians

- If the allocation of Jacobian is really needed, when the matrix is sparse and structured the cost can be still reduced from $n + 1$ function evaluations to just a few (precise number depending on the structure).
- I'll show this on an easy example, but can be generalized to more complex cases.

- Let $F(x)$ be such that each component $F_i(x)$ depends only on x_{i-1},x_i, x_{i+1}:

$$F(x) = \begin{pmatrix} F_1(x_1, x_2) \\ F_2(x_1, x_2, x_3) \\ F_3(x_2, x_3, x_4) \\ \vdots \\ F_n(x_{n-1}, x_n) \end{pmatrix}$$

- Structure of $F'(x)$ is clearly of the kind

$$F'(x) = \begin{pmatrix} \times & \times & & & \\ \times & \times & \times & & \\ & \times & \times & \times & \\ & & \times & \times & \times \\ & & & \times & \times \end{pmatrix}$$

- Assume we want to approximate $F'(x)$ columnwise as before:

$$\frac{\partial F}{\partial x_i}(x) \simeq \frac{F(x + he_i) - F(x)}{h}$$

  and apply perturbation $p = he_1$ for computing 1st column (i.e. perturb only x_1).

- Since only F_1 and F_2 do depend on x_1, only 1st and 2nd element of $\frac{\partial F}{\partial x_1}(x)$ are affected by this perturbation: from the third on, the value computed is - correctly - 0.

$$F_1(x + p) = F_1(x + he_1) \qquad F_2(x + p) = F_2(x + he_1)$$
$$F_k(x + p) = F_k(x + he_1) = F_k(x), \quad k \geq 3$$

$$\frac{F(x + he_1) - F(x)}{h} = \begin{pmatrix} \frac{F_1(x+he_1) - F_1(x)}{h} \\ \frac{F_2(x+he_1) - F_2(x)}{h} \\ \frac{F_3(x) - F_3(x)}{h} \\ \vdots \end{pmatrix} \simeq \begin{pmatrix} \frac{\partial F_1}{\partial x_1}(x) \\ \frac{\partial F_2}{\partial x_1}(x) \\ 0 \\ \vdots \end{pmatrix}$$

- It's useless to waste time to compute something whose value (=0) we exactly know!
- What happens if we consider the perturbation $p = h(e_1 + e_4)$, i.e. we perturb both x_1 and x_4?

$$F_1(x+p) = F_1(x + h(e_1 + e_4)) = F_1(x + he_1)$$

$$F_2(x+p) = F_2(x + h(e_1 + e_4)) = F_2(x + he_1)$$

$$F_3(x+p) = F_3(x + h(e_1 + e_4)) = F_3(x + he_4)$$

$$F_4(x+p) = F_4(x + h(e_1 + e_4)) = F_4(x + he_4)$$

$$F_5(x+p) = F_5(x + h(e_1 + e_4)) = F_5(x + he_4)$$

$$F_k(x+p) = F_k(x + h(e_1 + e_4)) = F_k(x), \quad k \geq 6$$

$$\frac{F(x+h(e_1+e_4))-F(x)}{h} = \begin{pmatrix} \frac{F_1(x+he_1)-F_1(x)}{h} \\ \frac{F_2(x+he_1)-F_2(x)}{h} \\ \frac{F_3(x+he_4)-F_3(x)}{h} \\ \frac{F_4(x+he_4)-F_4(x)}{h} \\ \frac{F_5(x+he_4)-F_5(x)}{h} \\ 0 \\ \vdots \end{pmatrix} \simeq \begin{pmatrix} \frac{\partial F_1}{\partial x_1}(x) \\ \frac{\partial F_2}{\partial x_1}(x) \\ \frac{\partial F_3}{\partial x_4}(x) \\ \frac{\partial F_4}{\partial x_4}(x) \\ \frac{\partial F_5}{\partial x_4}(x) \\ 0 \\ \vdots \end{pmatrix}$$

$$F'(x) = \begin{pmatrix} \times & \times & & & \\ \times & \times & \times & & \\ & \times & \times & \times & \\ & & \times & \times & \times \\ & & & \times & \times \end{pmatrix}$$

With just **one** perturbation ($p = h(e_1 + e_4)$) we have already approximated **two** columns of $F'(x)$

- The obvious generalization is to consider a perturbation $p = h(e_1 + e_4 + e_7 + ...)$ which allows to compute with just **one** extra evaluation of F columns 1,4,7... of the approximated Jacobian.
- We repeat the procedure with perturbations $p = h(e_2 + e_5 + e_8 + ...)$ and $p = h(e_3 + e_6 + e_9 + ...)$. With only **four** function evaluation (one for $F(x) + 3$ perturbations) we are able to compute the whole approximation of the Jacobian matrix.

- Previous example was simple but can be generalized to arbitrary functions $F : \mathbb{R}^n \mapsto \mathbb{R}^m$ by means of graphs and graphs coloring.
- Column incidence graph:
  - construct a graph with n nodes
  - draw an arc connecting node i and node k if at least one component of F depend both on x_i and x_k
  - Assign "colors" to nodes: nodes not connected by arcs may have the same coloring
- The number of different colors used, corresponds to the number of perturbations needed: if nodes i_1, i_2,...,i_ℓ have the same color, we use perturbation $p = h(e_{i_1} + e_{i_2} + ... + e_{i_\ell})$
- Literature exists devoted to algorithms providing optimal coloring (i.e. minimizing number of colors).

Finite difference approximation of the Hessian

- Consider first the situation in which you can exactly compute ∇f at given points.
- Note that $\nabla^2 f$ is the Jacobian of ∇f, so you can just use the previous techniques applying them to $F = \nabla f$.
- Taylor's expansion applied to ∇f:

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

- Taking $p = he_i$, we get

$$\nabla^2 f(x)e_i = \frac{(\nabla f)}{\partial x_i}(x) \simeq \frac{\nabla f(x+he_i) - \nabla f(x)}{h}$$

  with error $\mathcal{O}(h)$
- Repeating for all $i = 1, ..., n$ we get an approximation of $\nabla^2 f(x)$ with $n+1$ gradient evaluations.

Some remarks

- Note that this way the approximation of $A \simeq \nabla^2 f(x)$ may fail to be symmetric. It is worthwile to "adjust" the approximation by replacing it with its symmetric part:
$$\frac{1}{2}(A + A^T)$$
- A matrix-free approximation can be obtained also for the Hessian:
$$\nabla^2 f(x)p \simeq \frac{\nabla f(x + hp) - \nabla f(x)}{h}$$
at the cost of one further gradient evaluation.
- Similarly to Jacobian, a sparse Hessian can also be built at a reduced cost; actually other formulae can be used in order to exploit simmetry and obtain the approximation at a even lower computational cost.

If the gradient is not available, you may directly approximate second order derivatives using only function values with centered finite differences:

$$\frac{\partial f}{\partial x_i \partial x_j}(x) = \frac{f(x + he_i + he_j) - f(x + he_i) - f(x + he_j) + f(x)}{h^2} + \mathcal{O}(h)$$

Suitable formulae are available for sparse Hessian in order to reduce the number of function evaluations, which would be quite high.

Part V

Constrained optimization - Chapters 12,16,14,17

Consider the general optimization problem

$$\begin{aligned} \min \quad & f(x) \\ \text{s.t.} \quad & g(x) \leq 0 \\ & h(x) = 0 \end{aligned}$$

with $f : \mathbb{R}^n \mapsto \mathbb{R}$, $g : \mathbb{R}^n \mapsto \mathbb{R}^m$, $h : \mathbb{R}^n \mapsto \mathbb{R}^p$.

Definition (Feasible set)

The feasible set X is the set of points in $\mathbb{R}^n$ which satisfy the constraints, namely

$$X = \{x \in \mathbb{R}^n | g(x) \leq 0, h(x) = 0\}$$

Definition (Active set)

The active set, at a given $x \in X$, is the set of equality constraints h_j + the subset of inequality constraints g_i at which $g_i(x) = 0$

Definition (Lagrangian function)

The Lagrangian function for the general problem is

$$\mathcal{L}(x, \lambda, \mu) = f(x) + \sum_{i=1}^{m} \lambda_i g_i(x) + \sum_{j=1}^{p} \mu_j h_j(x)$$

or more compactly

$$\mathcal{L}(x, \lambda, \mu) = f(x) + \lambda^T g(x) + \mu^T h(x)$$

Definition (LICQ)

Given the point x^* and the active set $\mathcal{A}(x^*)$, we say that the linear independence constraint qualification (LICQ) holds if the set of gradients of active constraints form a linearly independent set.

Theorem (FO necessary conditions, Karush-Kuhn-Tucker conditions)

Suppose that x^* is a local solution and that the LICQ holds at x^*. Then there are Lagrange multiplier vectors λ^* and μ^* such that the following conditions are satisfied at (x^*, λ^*, μ^*):

$$\begin{aligned} \nabla_x \mathcal{L}(x^*, \lambda^*, \mu^*) &= 0 \\ h(x^*) &= 0 \\ g(x^*) &\leq 0 \\ \lambda^* &\geq 0 \\ (\lambda^*)^T g(x^*) &= 0 \end{aligned}$$

Remark

Since $(\lambda^*)^T g(x^*) = \sum_{i=1}^n \lambda_i^* g_i(x^*)$, *from the last three conditions of the KKT conditions, it follows that for all* $i = 1, \ldots, n$ *at least one among* λ_i^* *and* $g_i(x^*)$ *has to be zero.*

Definition (Strict complementarity)

Given a local solution x^* and Lagrange multipliers λ^*, μ^* satisfying the KKT conditions, the strict complementarity condition holds if exactly one among λ_i^* and $g_i(x^*)$ is equal to 0.