Optimization in higher dimensions

- Quasi-Newton Methods
- Conjugate Gradient Method

Optimization in higher dimensions

- Quasi-Newton Methods
- Conjugate Gradient Method

Context and Goals

- gradient descent algorithms have linear convergent rate: cost $O(N)$
- too slow for ill conditioned problems
- Newton's method: quadratic convergence
  - initialize close to solution
  - cost $O(N^3)$ per iteration (worst case)

Goal

Find an algorithm converging faster than GD without increasing the computational task!

A bit of history

[Nocedal, Wright, Numerical Optimization 06], Chapters 6-7

$\star$ in the 50s W.C. Davidon used "coodrdinate descent" method (GD on coordinates)

$\star$ the computer would always crash before the simulation was finished

$\star$ Davidon decided to find a way of accelerating the optimization process: he found one of the most creative ideas in nonlinear optimization

$\star$ Fletcher and Powell demonstrated that this algorithm was faster and more reliable than existing methods at the time

$\star$ paradoxically, Davidon's paper was not accepted for publication. It remained a technical report for more than thirty years until it appeared in *SIAM Journal on Optimization* in 1991!

Motivation

Recall the Variable Metric Method and replace A_i^{-1} by S_i:

Algorithm 1 (Generic Variable Metric method)

Choose the starting point x_0

Iteration *i:*

- *compute* $f(x_i), \nabla f(x_i)$ *and eventually* $D^2 f(x_i)$
- *choose a symmetric positive-definite matrix* S_i*: compute the new direction*
$$d_i = -S_i \nabla f(x_i)$$
- *perform a line-search from* x_i *in the direction* d_i *giving a new iterate*
$$x_{i+1} = x_i + t_i d_i = x_i - t_i S_i \nabla f(x_i).$$

$\star$ in the modified Newton method S_i is computed as follows: find the Hessian $D^2 f(x_i)$, modify it to make it "well positive definite", then invert it or solve $S_i d_i = \nabla f(x_i)$

$\star$ in quasi-Newton method we try to skip all of this and compute S_i recursively with one objective: $S_i - (D^2 f(x_i))^{-1} \to 0$

$\star$ in fact, it is enough to have $(S_i - (D^2 f(x_i))^{-1})(x_{i+1} - x_i) \to 0$.

Variable Metric method: quadratic case

$\star$ minimize $f(x) = \frac{1}{2}x^T Ax - b^T x$ with Steepest Descent line-search
$\star$ denote $E(x_i) = f(x_i) - \min f$: error in terms of objective function
$\star$ $x_{i+1} = x_i - t_{opt} S_i \nabla f(x_i)$ is equivalent to a change of coordinates $\xi = S_i^{1/2}x$
$\star$ the step i in the VM method is just a Steepest-Descent step for the matrix $S_i^{1/2}AS_i^{1/2}$. Therefore we have the estimate

$$E(x_{i+1}) \leq \left(\frac{Q-1}{Q+1}\right)^2 E(x_i)$$

where Q is the condition number of $S_i^{1/2}AS_i^{1/2}$
$\star$ if S_i is close to $D^2 f(x_i)^{-1} = A^{-1}$ then $S_i^{1/2}AS_i^{1/2}$ is close to the identity matrix so Q is close to 1.
$\star$ Finally, if Q converges to 1, we eventually get that $E(x_{i+1})/E(x_i) \to 0$, i.e. super-linear convergence

Basic rules for updating S_i

$\star$ Taylor expansion formula tells us that

$$\nabla f(x_{i+1}) - \nabla f(x_i) \approx D^2 f(x_i)(x_{i+1} - x_i)$$

$\star$ Therefore, it is reasonable to request that

$$S_{i+1}(\nabla f(x_{i+1}) - \nabla f(x_i)) = x_{i+1} - x_i$$

called the secant relation (make parallel with the 1D case)

$\star$ With the notations $g_i = \nabla f(x_i)$, $p_i = x_{i+1} - x_i$, $q_i = g_{i+1} - g_i$ we have

$$S_{i+1} q_i = p_i,$$

called the quasi-Newton equation

$\star$ this leaves us with infinitely many possibilities... another goal is that

$S_{i+1} - S_i$ is as simple as possible!

$\star$ initialization? one may simply choose $S_0 = \mathrm{Id}$, multiple of identity, diagonal matrix, etc.

Small rank updates

⋆ idea: find $S_{i+1} = S_i + B_i$ where B_i has low rank

⋆ Rank 1 updates: $B_i = \alpha_i v_i v_i^T$ - one may find B_i such that the quasi-Newton relation holds

$$S_{i+1} = S_i + \alpha_i z_i z_i^T$$

⋆ the quasi-Newton relation $p_i = S_{i+1} q_i$ implies

$$z_i = \omega_i (p_i - S q_i)$$

⋆ in the end we get

$$S_{i+1} = S_i + \frac{1}{(p_i - S_i q_i)^T q_i} [p_i - S_i q_i][p_i - S_i q_i]^T$$

⋆ not possible to guarantee that S_{i+1} is positive definite if S_i is

Rank 2 updates: DFP

⋆ Davidon-Fletcher-Powell: historically, the first "good" quasi-Newton method
⋆ use rank 2 updates: guarantee the positive-definiteness of S_{i+1} under reasonable hypotheses

Proposition 1

Let S be a positive definite symmetric matrix and p and q be two vectors such that $p^T q > 0$. Then the matrix

$$S' = S + \frac{1}{p^T q} pp^T - \frac{1}{q^T S q} Sqq^T S$$

is symmetric positive definite and satisfies $S'q = p$.

⋆ Proof: just compute $S'q$ and $xS'x$ and do a bit of linear algebra.

⋆ How to get this idea? Just choose $S_{i+1} = S_i + \alpha uu^T + \beta vv^T$ (rank 2 update)
⋆ then choose $u = p_i$ and $v = S_i q_i$

DFP method

⋆ DFP update:

$$S_{i+1} = S_i + \frac{1}{p_i^T q_i} p_i p_i^T - \frac{1}{q_i^T S_i q_i} S_i q_i q_i^T S_i$$

⋆ the condition $q_i^T p_i > 0$ is equivalent to

$$(\nabla f(x_{i+1}) - \nabla f(x_i)) \cdot (x_{i+1} - x_i) > 0,$$

which is true if f is strictly convex: reasonable assumption near a minimum...

⋆ when using Wolfe line-search we can guarantee that $q_i^T p_i > 0$.

⋆ for the quadratic case DFP becomes the conjugate gradient method

⋆ it turns out DFP is not the best method out there...

- it does not "self-correct" when S_i gets far from the inverse Hessian

Duality: quasi-Newton relation

⋆ any quasi-Newton update can generate another one:

- $S_{i+1} = S_i + B_i(S_i, p_i, q_i)$ such that $S_{i+1}q_i = p_i$
- then $q_i = S_{i+1}^{-1}p_i$ where $S_{i+1}^{-1} = (S_i + B(S_i, p_i, q_i))^{-1}$
- switching the roles of p_i and q_i we get a different update, called the dual update

⋆ how to get the dual of DFP: replace S_i with S_i^{-1} and interchange p_i and q_i

$$S_{i+1}^{-1} = S_i^{-1} + \frac{1}{q_i^T p_i} q_i q_i^T - \frac{1}{p_i^T S_i^{-1} p_i} S_i^{-1} p_i p_i^T S_i^{-1}$$

⋆ a direct computation or Sherman-Morrison's formula gives:

$$S_{i+1} = S_i - \frac{p_i q_i^T S_i + S_i q_i p_i^T}{p_i^T q_i} + \left(1 + \frac{q_i^T S_i q_i}{p_i^T q_i}\right) \frac{p_i p_i^T}{p_i^T q_i}$$

The BFGS update

⋆ BFGS: Broyden, Fletcher, Goldfarb, Shanno

$$S_{i+1} = S_i - \frac{p_i q_i^T S_i + S_i q_i p_i^T}{p_i^T q_i} + \left(1 + \frac{q_i^T S_i q_i}{p_i^T q_i}\right) \frac{p_i p_i^T}{p_i^T q_i}$$

⋆ widely used in most of the codes implemented today
⋆ since BFGS is the dual of DFP, and a matrix is positive-definite if and only if its inverse is positive-definite, the BFGS update maintains positive-definiteness if $p_i^T q_i > 0$ (same hypothesis as for DFP to work...)
[Nocedal, Wright, Numerical Optimization 06], Chapters 6-7
⋆ **Local super-linear convergence:** If an algorithm using BFGS with Wolfe's line-search converges to x^* where f is strongly convex with Lipschitz Hessian then the convergence rate is super-linear
⋆ BFGS has effective self-correcting properties

BFGS: alternative definition

S_{i+1} solves
$$\min \|S - S_i\|$$
subject to $S = S^T, Sq_i = p_i$.

- ⋆ $\|A\| = \|W^{1/2}AW^{1/2}\|_F$, $\|C\|_F^2 = \sum c_{ij}^2$.
- ⋆ The weight matrix W satisfies $Wp_i = q_i$
- ⋆ any other choice of norm would give another quasi-Newton method.

Despite intense research no method better than BFGS was found!

Extreme cases

Dimension 1:

⋆ the quasi-Newton relation is just $S_{i+1} = \dfrac{p_i}{q_i}$ and we get

$$x_{i+1} = x_i - \frac{x_i - x_{i-1}}{f'(x_i) - f'(x_{i-1})} f'(x_i)$$

which is the false position (or secant) method

Large dimension:

⋆ same disadvantage as Newton methods - a $n \times n$ matrix may be too large to store in memory

⋆ it is possible to store only the update vectors and compute matrix - vector products by doing only scalar - products

$$(uv^T)x = u(v^T x) = (v^T x)u$$

⋆ limited memory-BFGS (LBFGS): use only the last m vectors p_i, q_i in order to compute S_{i+1} - good behavior in practice despite being an approximation of BFGS

Computational cost per iteration

$\star$ after the function value, gradient and Hessian are computed (this is non-negligible in some applications)

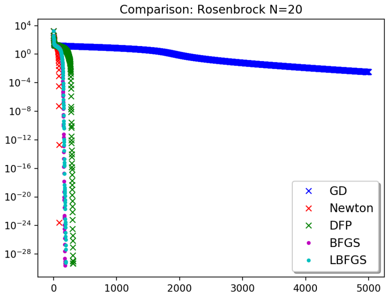
- GD: $O(N)$
- Newton: $O(N^3)$ in worst case (solving a linear system) - it all depends on the structure of the Hessian
- BFGS, DFT: $O(N^2)$ - matrix vector products
- LBFGS: $O(mN)$ where m is the fixed number of gradients to remember

Practical example: the N-dimensional Rosenbrock

$$f(x) = \sum_{i=1}^{N-1} [100(x_{i+1} - x_i^2)^2 + (1 - x_i)^2]$$

with global minimum at $x^* = (1, 1, ..., 1)$.

$\star$ ill conditioning: the optimization process wants to achieve $x_{i+1} \approx x_i^2$ rather than minimizing $(x_i - 1)^2$ and go towards the global minimum!

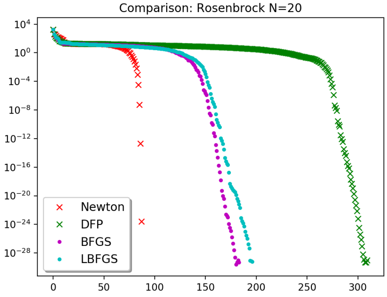


Practical example: the N-dimensional Rosenbrock

$$f(x) = \sum_{i=1}^{N-1} [100(x_{i+1} - x_i^2)^2 + (1 - x_i)^2]$$

with global minimum at $x^* = (1, 1, ..., 1)$.

$\star$ ill conditioning: the optimization process wants to achieve $x_{i+1} \approx x_i^2$ rather than minimizing $(x_i - 1)^2$ and go towards the global minimum!

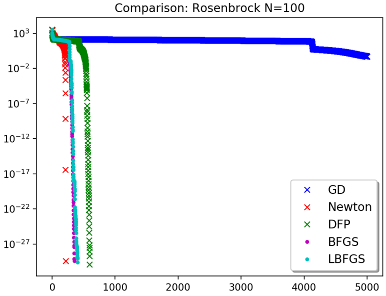


Practical example: the N-dimensional Rosenbrock

$$f(x) = \sum_{i=1}^{N-1} [100(x_{i+1} - x_i^2)^2 + (1 - x_i)^2]$$

with global minimum at $x^* = (1, 1, ..., 1)$.

⋆ ill conditioning: the optimization process wants to achieve $x_{i+1} \approx x_i^2$ rather than minimizing $(x_i - 1)^2$ and go towards the global minimum!

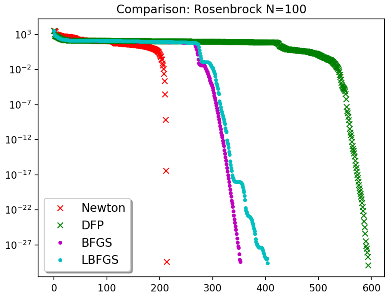


Practical example: the N-dimensional Rosenbrock

$$f(x) = \sum_{i=1}^{N-1} [100(x_{i+1} - x_i^2)^2 + (1 - x_i)^2]$$

with global minimum at $x^* = (1, 1, ..., 1)$.

$\star$ ill conditioning: the optimization process wants to achieve $x_{i+1} \approx x_i^2$ rather than minimizing $(x_i - 1)^2$ and go towards the global minimum!



Conclusion: quasi-Newton methods

- equivalent of the Secant method in higher dimensions
- achieve super-linear convergence without using the Hessian
- for extremely large n BFGS may be costly from a memory point of view: if possible use L-BFGS instead
- BFGS and LBFGS are often available in standard optimization libraries: Example `scipy.optimize.minimize`

Optimization in higher dimensions

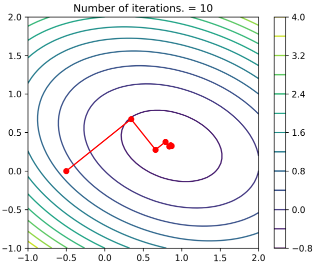
- Quasi-Newton Methods
- Conjugate Gradient Method

Motivation

⋆ if A is symmetric, positive-definite then solving the system $Ax = b$ is equivalent to minimizing the quadratic function

$$f : x \mapsto \frac{1}{2}x^T Ax - b \cdot x$$

⋆ the gradient of this quadratic function is $\nabla f(x) = Ax - b$
⋆ direct method: process details about the matrix A (factorization) and then solve the system: complexity is between $O(n^2)$ and $O(n^3)$.
⋆ iterative algorithms produce an approximation of the solution, which might be good enough for very large n after a few iterations
⋆ for example: the gradient algorithm with Steepest-Descent will quickly converge to the optimum, but we can do better



Conjugate directions

⋆ A given symmetric positive-definite matrix A defines a scalar product

$$\langle x, y\rangle = x^T Ay$$

⋆ Two (non-zero) directions d_1 and d_2 are called conjugate with respect to A if they are orthogonal w.r.t. the above scalar product:

$$d_1 \text{ and } d_2 \text{ are conjugate } \iff d_1 A d_2 = 0$$

⋆ we may also call two directions which are conjugate w.r.t. A as being A-orthogonal

⋆ why is this useful? suppose $d_1, ..., d_k$ are mutually A-orthogonal and we have the decomposition

$$d = \sum_{j=1}^{k} \alpha_j d_j$$

Then, using the orthogonality property, we can find the coefficients α_i explicitly:

$$d_i^T Ad = \alpha_i d_i^T Ad_i \Rightarrow \alpha_i = \frac{d_i^T Ad}{d_i^T Ad_i} = \frac{\langle d, d_i\rangle}{\langle d_i, d_i\rangle}$$

⋆ **Consequence:** If $d_1, ..., d_k$ are mutually orthogonal then they are linearly independent! (for a proof, use the above formula to see that $d = 0 \Rightarrow \alpha_i = 0$)

Why is this concept useful?

Proposition 2 (Solve a system using Conjugate Directions)

Let A be a symmetric positive-definite matrix and $d_1, ..., d_n$ a (complete) system of n non-zero A-orthogonal vectors. Then the solution x^* to the system $Ax = b$ is given by the formula

$$x^* = \sum_{j=1}^{n} \frac{b^T d_j}{d_j^T A d_j} d_j$$

⋆ An equivalent formulation:

$$x^* = A^{-1} b = \sum_{j=1}^{n} \frac{b^T d_j}{d_j^T A d_j} d_j = \left(\sum_{j=1}^{n} \frac{1}{d_j^T A d_j} d_j d_j^T \right) b$$

which gives us the explicit inverse of A

$$A^{-1} = \sum_{j=1}^{n} \frac{1}{d_j^T A d_j} d_j d_j^T$$

⋆ All this is good when we know a complete family of A-orthogonal directions!

Conjugate Directions: quadratic case

Algorithm 2 (Conjugate Directions method)

Let A be a $n \times n$ symmetric positive-definite matrix, b a vector and $f(x) = \frac{1}{2}x^TAx - b^Tx$ the quad. form associated to A and b.

Let $d_0, .., d_{n-1}$ be a system of A-orthogonal vectors and x_0 a starting point. Then, with the notation $g_i = \nabla f(x_i) = Ax_i - b$, the iterative process

$$x_{i+1} = x_i + \gamma_i d_i, \gamma_i = -\frac{d_i^T g_i}{d_i^T A d_i}, \ i = 1, ..., n$$

converges to the unique minimizer x^* of f in n steps.

⋆ The step γ_i is optimal in the direction d_i: define $q(t) = f(x + td)$ then

$$q'(t) = \nabla f(x + td) \cdot d = d \cdot \nabla f(x) + td^TAd$$

⋆ Proof: just look at x_n and see that it gives exactly the formula for x^*.

⋆ Important idea: $d_k A(x_k - x_0) = 0$ for any $k \geq 0$

⋆ Again: all this is good when we know a complete family of A-orthogonal directions!

Properties of the Conjugate Directions Method

⋆ define for each $i \geq 1$ the linear space $\mathcal{B}_{i-1} = \text{Span}\{d_0, ..., d_{i-1}\}$
⋆ if we define the affine subspaces $M_i = x_0 + \mathcal{B}_{i-1}$ then

$$\{x_0\} = M_0 \subset M_1 \subset ... \subset M_n = \mathbb{R}^n$$

⋆ the Conjugate Directions method generate the minimizers of f in each of the affine spaces M_i

Proposition 3

For every $1 \leq i \leq n$ the vector x_i is the minimizer of f on the affine subspace $M_i = x_0 + \mathcal{B}_{i-1}$. In particular, as shown previously, x_i minimizes f on the line $\{x_{i-1} + td_{i-1} : t \in \mathbb{R}\}$.

Proof: ⋆ Compute the gradient $g_i = \nabla f(x_i) = Ax_i - b$ and note that g_i is orthogonal to $d_0, ..., d_{i-1}$.
⋆ Then obtain that $\langle \nabla f(x_i), x - x_i \rangle = 0$ for $x \in x_0 + \mathcal{B}_{i-1}$.
⋆ f is strictly convex so Euler's inequality tells us that x_i is indeed the minimizer of f in $x_0 + \mathcal{B}_{i-1}$.

Build a basis of conjugated directions

⋆ recall the Gram-Schmidt procedure

⋆ define the A-projection of v on u:

$$\text{proj}_u(v) = \frac{\langle u, v\rangle}{\langle u, u\rangle} u = \frac{u^T A v}{u^T A u} u$$

Algorithm 3 (Gram-Schmidt)

```
0. Take a basis $(v_i)$ of $\mathbb{R}^n$: e.g. the canonical basis.
1. $u_1 = v_1$
2. $u_2 = v_2 - \text{proj}_{u_1}(v_2)$
3. $u_3 = v_3 - \text{proj}_{u_1}(v_3) - \text{proj}_{u_2}(v_3)$
...
n. $u_n = v_n - \text{proj}_{u_1}(v_n) - ... - \text{proj}_{u_{n-1}}(v_n)$
In the end normalize the vectors: $d_i = \frac{1}{\sqrt{u_i^T A u_i}} u_i$
```

⋆ in this form the process is not numerically stable: due to rounding errors the vectors u_k may not be exactly orthogonal...

Conjugate Gradient Method

$\star$ we can compute the family of A-orthogonal directions during the optimization algorithm

Algorithm 4 (Conjugate Gradient)

```
Choose arbitrary initialization point $x_0$ and set $d_0 = -g_0 = -\nabla f(x_0) = b - Ax_0$
Loop on: $i = 0, ..., n-1$
    • if $\nabla f(x_i) = 0$ then stop.
    • $x_{i+1} = x_i + \gamma_i d_i$ with $\gamma_i = -\frac{d_i^T g_i}{d_i^T A d_i}$
    • Compute new gradient $g_{i+1} = \nabla f(x_{i+1}) = Ax_{i+1} - b$
    • Compute new direction $d_{i+1} = -g_{i+1} + \beta_i d_i$ with $\beta_i = \frac{g_{i+1}^T A d_i}{d_i^T A d_i}$
```

$\star$ as before γ_i is the optimal step in the direction d_i
$\star$ the parameter β_i is chosen such that $d_{i+1}^T A d_i = 0$
$\star$ the new direction d_{i+1} is given by the projection of the anti-gradient direction $-g_{i+1}$ on the previous direction

Main properties of CG

Proposition 4 (CG is a Conjugate Direction method)

If the algorithm does not terminate at step i then:

- *the gradients $g_0, \dots, g_{i-1}$ at $x_0, \dots, x_{i-1}$ are non-zero and* $Span\{g_0, g_1, \dots, g_{i-1}\} = Span\{g_0, Ag_0, \dots, A^{i-1}g_0\}$
- *The directions $d_0, \dots, d_{i-1}$ are non-zero and* $Span\{d_0, d_1, \dots, d_{i-1}\} = Span\{g_0, Ag_0, \dots, A^{i-1}g_0\}$
- *The directions $d_0, \dots, d_{i-1}$ are A orthogonal*
- *Alternative formulas for γ_i and β_i:*

$$\gamma_i = \frac{g_i^T g_i}{d_i^T A d_i} \text{ and } \beta_i = \frac{g_{i+1}^T g_{i+1}}{g_i^T g_i}.$$

$\star$ A sequence of the type $g_0, Ag_0, A^2g_0, \dots$ is called a Krylov sequence

Consequences and convergence

⋆ x_i is the minimizer of f in the affine subspace

$$x_0 + \text{Span}\{d_0, ..., d_{i-1}\} = x_0 + \text{Span}\{g_0, Ag_0, ..., A^{i-1}g_0\}$$

⋆ x_i is the minimizer of f in the affine subspace generated by x_0 and polynomials of A of degree at most $i-1$ times g_0 (denote this polynomial space by $\mathcal{P}_{i-1}$)

$$x_0 + \{p(A)g_0 : p(z) = \sum_{i=0}^{i-1} p_i z^i\}$$

⋆ error in terms of the objective function: $E(x) = f(x) - \min f = \frac{1}{2}(x-x^*)^T A(x-x^*)$

Proposition 5 (Error for CG)

$$E(x_i) = \min_{p\in\mathcal{P}_{i-1}} \frac{1}{2}(x_0 - x^*)A(\text{Id} - Ap(A))^2(x_0 - x^*)$$

⋆ Proof: write $x_i = x_0 + p(A)g_0$ and recall that $\nabla f(x_i) = A(x_i - x^*)$

Error in terms of the spectrum of A

Corollary

Let Σ be the spectrum of A. Then

$$E(x_i) \leq E(x_0) \min_{p \in \mathcal{P}_i^*} \max_{\lambda \in \Sigma} p^2(\lambda),$$

where $\mathcal{P}_i^*$ is the set of polynomials p of degree at most i such that $p(0) = 1$. Another estimate is

$$E(x_i) \leq \frac{1}{2} |x^* - x_0|^2 \min_{p \in \mathcal{P}_i^*} \max_{\lambda \in \Sigma} \lambda p^2(\lambda),$$

⋆ Proof: use an orthonormal basis made of eigenvectors of A

⋆ denote by Q the condition number of A. Then there exists a polynomial $q \in \mathcal{P}_s^*$ such that

$$\max_{\lambda \in \Sigma} q_s(\lambda)^2 \leq 4 \left(\frac{\sqrt{Q} - 1}{\sqrt{Q} + 1} \right)^{2s}$$

Error estimate in terms of the condition number

⋆ for the Conjugate Gradient algorithm we have

$$E(x_N) \leq 4\left(\frac{\sqrt{Q}-1}{\sqrt{Q}+1}\right)^{2N} E(x_0),$$

where Q is the condition number of A.
⋆ compare this with the error estimate for the Steepest-Descent

$$E(x_N) \leq \left(\frac{Q-1}{Q+1}\right)^{2N} E(x_0)$$

⋆ in order to reduce the initial error by a factor of ε one needs to do $O(Q)$ steps with Steepest Descent compared to $O(\sqrt{Q})$ steps with CG. This is a big difference!
⋆ CG is supposed to converge in n iterations, however rounding errors may prevent the convergence!
⋆ moreover, if A has $k \leq n$ distinct eigenvalues then CG converges in k iterations!
⋆ Often, for n large, the process is stopped before reaching n iterations, when the error estimate is small enough

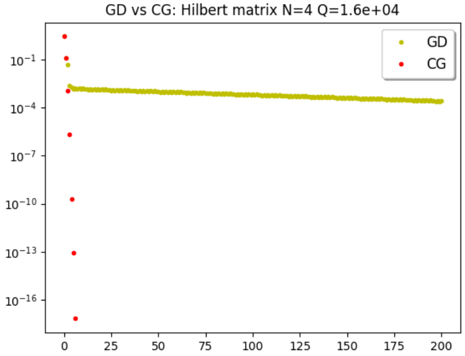
Example: Hilbert matrices

$A = (1/(i+j-1))_{1\leq i,j\leq n}$, ill conditioned

$\star$ below you can see a comparison between GD with optimal step and CG. The residual $|Ax-b|$ is plotted at every iteration

$\star$ the residual decreases slowly for GD: the algorithm tends to go multiple times in the same direction! CG optimizes once and for all in the current direction.

$\star$ small residual does not mean that x is close to x^*: $Ax-b = A(x-x^*)$!



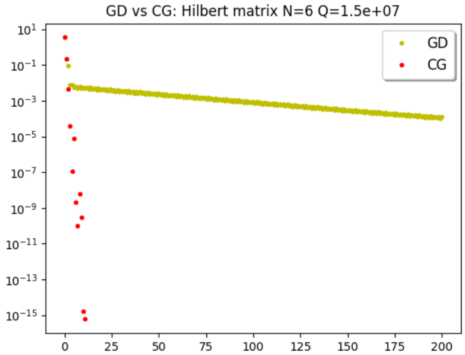
Example: Hilbert matrices

$A = (1/(i+j-1))_{1\leq i,j\leq n}$, ill conditioned

$\star$ below you can see a comparison between GD with optimal step and CG. The residual $|Ax-b|$ is plotted at every iteration

$\star$ the residual decreases slowly for GD: the algorithm tends to go multiple times in the same direction! CG optimizes once and for all in the current direction.

$\star$ small residual does not mean that x is close to x^*: $Ax-b = A(x-x^*)$!



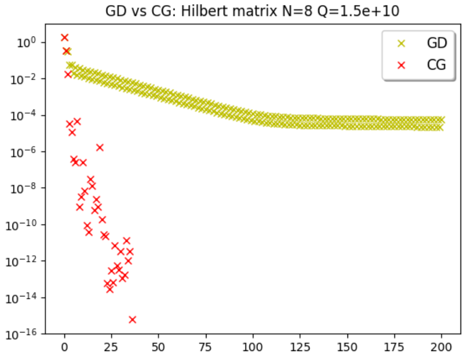
Example: Hilbert matrices

$A = (1/(i+j-1))_{1\leq i,j\leq n}$, ill conditioned

⋆ below you can see a comparison between GD with optimal step and CG. The residual $|Ax-b|$ is plotted at every iteration

⋆ the residual decreases slowly for GD: the algorithm tends to go multiple times in the same direction! CG optimizes once and for all in the current direction.

⋆ small residual does not mean that x is close to x^*: $Ax-b = A(x-x^*)$!



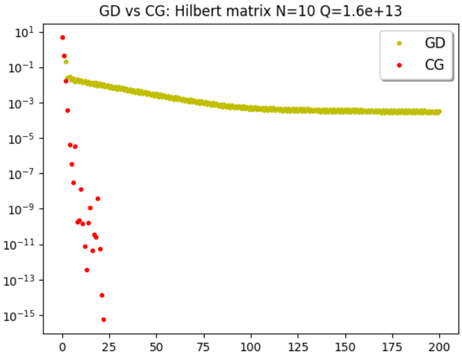
Example: Hilbert matrices

$A = (1/(i+j-1))_{1\leq i,j\leq n}$, ill conditioned

$\star$ below you can see a comparison between GD with optimal step and CG. The residual $|Ax-b|$ is plotted at every iteration

$\star$ the residual decreases slowly for GD: the algorithm tends to go multiple times in the same direction! CG optimizes once and for all in the current direction.

$\star$ small residual does not mean that x is close to x^*: $Ax-b = A(x-x^*)$!



Important application: approximate solution of PDEs

- Consider Laplace's equation

$$\text{Find } u \in H_0^1(D) \text{ such that } \begin{cases} -\Delta u = f & \text{in } D \\ u = 0 & \text{on } \partial D \end{cases}$$

  where $f \in L^2(D)$ is a given source.

- It is possible to associate to this a variational formulation:

$$\text{Find } u \in V \text{ such that } \forall v \in V \text{ we have } a(u,v) = \ell(v)$$

  where
  - The Hilbert space V is a Sobolev space $H_0^1(D)$
  - $a(\cdot,\cdot)$ is a bilinear form on V given by $a(u,v) = \int_D \nabla u \cdot \nabla v dx$
  - $\ell(\cdot)$ is a linear form on V given by $\ell(v) = \int_D fv dx$

- Lax-Milgram's theorem assures us that such a problem has a solution on V.

Finite element method

- The finite element method proposes to search for an approximation u_h in a finite dimension subspace $V_h \subset V$.
- the variational formulation is replaced by:

Find $u_h \in V_h$ such that $\forall v_h \in V_h$ we have $a(u_h, v_h) = \ell(v_h)$

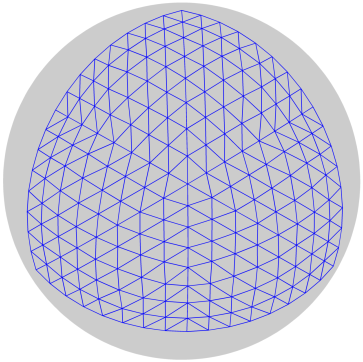
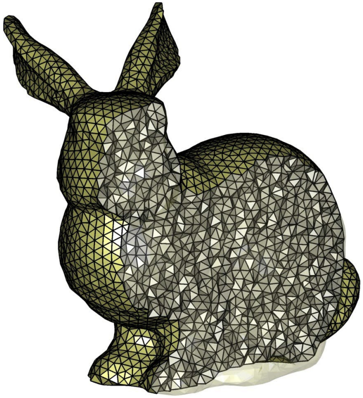
- Advantage : V_h being of finite dimension, we can choose a basis $\mathcal{B} = \{\varphi_i\}_{i=1}^N$ and the variational formulation becomes a linear system $A\bar{u} = b$ with

$$A = (a(\varphi_i, \varphi_j)),\ b = (\ell(\varphi_i))$$

  where $\bar{u}$ are the coordinates of u_h in the basis $\mathcal{B}$.
- The choice of the basis is important: one objective is to have a system given by a sparse matrix

Construct a finite element space

- The domain D is discretized using a mesh $\mathcal{T}_h$ which consists of a partitions in triangles in 2D or tetrahedra in 3D.
- The parameter h which indicates the convergence of the method is typically related to the size of the mesh elements.

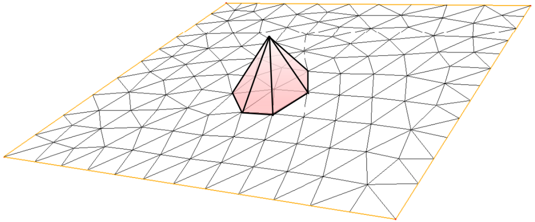
Construct a finite element space (2)

A basis $\{\varphi_1, ..., \varphi_{N_h}\}$ of finite element functions is introduced on the mesh $\mathcal{T}_h$

Example

- N_h is the number of vertices $a_1, ..., a_{N_h}$ of the mesh
- For each $i = 1, ..., N_h$, φ_i is affine on each triangle $T \in \mathcal{T}_h$ and

$$\varphi_i(a_j) = 1 \text{ et } \varphi_i(a_j) = 0 \text{ pour } i \neq j$$



Formulation of a matrix system

Decompose the solution u_h in the basis of finite elements

$$u_h = \sum_{i=1}^{N_h} u_j \varphi_i$$

and the variational problem becomes a linear system of size $N_h \times N_h$

$$KU = f$$

where

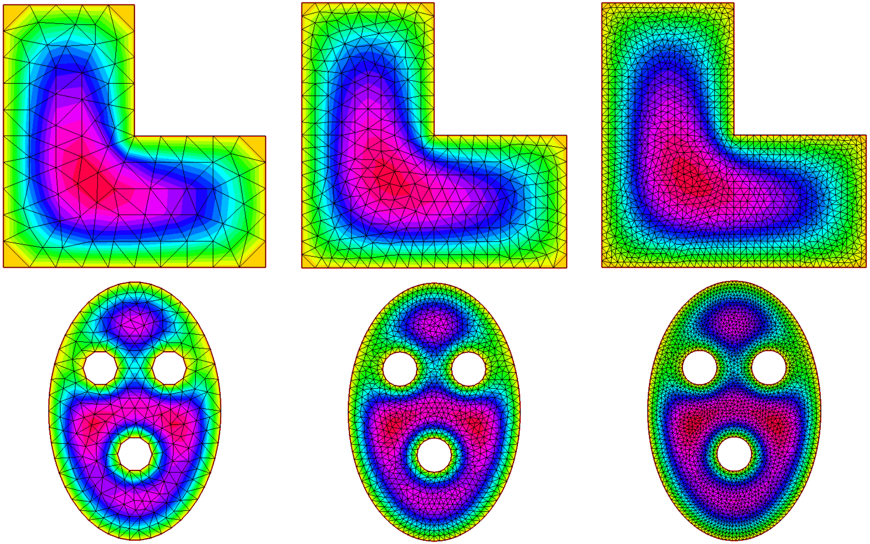
- $U = \begin{pmatrix} u_1 \\ \vdots \\ u_{N_h} \end{pmatrix}$ is the vector of coefficients
- K is the rigidity matrix given by $K_{ij} = a(\varphi_i, \varphi_j)$
- F is the vector $F = (\ell(\varphi_i))_{i=1,\dots,N_h}$.

$\star$ The matrix K will be symmetric and positive-definite so we are in the good framework where CG works!

$\star$ when N_h is large (a few tens of thousands of elements) direct methods will fail to work (computation time, memory limitations)

$\star$ CG will work well even for $N_h > 10^5$

Some results



CG for general functions

Algorithm 5 (Fletcher-Reeves CG on $\mathbb{R}^n$)

```
Choose a starting point $x_0$. Set cycle counter $k = 1$.
Cycle $k$: Initialization of the cycle: Given $x_0$ compute $g_0 = \nabla f(x_0), d_0 = -g_0$
Inner Loop: for $i = 0, ..., n-1$
    • if $g_i = 0$ terminate, otherwise set $x_{i+1}$ as the minimizer of $f(x_i + td_i)$
    • compute $g_{i+1} = \nabla f(x_{i+1})$
    • set $d_{i+1} = -g_{i+1} + \beta_i d_i$ with $\beta_i = \frac{g_{i+1}^T g_{i+1}}{g_i^T g_i}$
When the loop is finished replace $x_0$ with $x_n$ and restart.
```

$\star$ note that in the inner loop we have a Steepest Descent line-search: this is not applicable in general. A line-search procedure should be used instead!

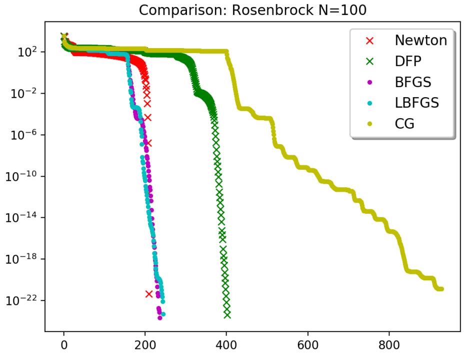
$\star$ It can be proved that in the non-degenerate case the convergence is quadratic in the number of cycles i.e.

$$|x^{k+1} - x^*| \leq C|x^k - x^*|^2$$

where x^k is the sequence of starting points for cycles

Comparison with previous methods

$\star$ again on the Rosenbrock function for $N = 100$
$\star$ in general nonlinear-CG converges faster than GD but not necessarily faster than quasi-Newton methods



Conclusion on Conjugate Gradient method

- when a complete system of A-orthogonal directions is known everything is explicit
- it can be made into an iterative algorithm with a convergence ratio way better than Steepest Descent
- it converges in n iterations (theoretically). In practice, for large n, we usually stop the process once the error estimate
$$E(x_N) \leq 4\left(\frac{\sqrt{Q}-1}{\sqrt{Q}+1}\right)^{2N} E(x_0)$$
is satisfying.
- cost of a step in CG:
$$O(n) + \text{ cost of a matrix-vector multiplication } d \to Ad.$$
This is particularly efficient when A is sparse (has few non-zero elements)
- Disadvantage: sensitivity to the condition number!

Conclusions: unconstrained optimization in ND

- Gradient Descent algorithms: sensitive to conditioning!
- Newton methods: fast convergence under right hypotheses. Major practical inconveniences:
  - compute Hessian matrix and (possibly) store it
  - doesn't necessarily decrease the function value
  - solve a linear system at every iteration
- variable metric methods: compute an approximation of the inverse Hessian
  - BFGS: rank 2 updates, standard in available implementations
  - even better for large n: L-BFGS - limit memory by using only information from the previous m iterations
- Conjugate Gradient methods: less sensitive to conditioning than Steepest Descent
- Newton-Gauss: non-linear least squares
- Nedler-Mead: gradient free method

Practical discussion

⋆ get used to the structure of algorithms which are already implemented: in the practical session you will play with tools from `scipy.optimize`

⋆ keep in mind to minimize the number of function evaluations in your codes: not all functions to be optimized are computed in a cheap way

- when the value of a function or its gradient are used multiple times store them in some variables
- in some computations involving physical simulations the gradient can often be computed using existing information from the solution given by the model: there is no point computing it multiple times