Optimization Algorithms
Exercise 2

Marc Toussaint
Learning & Intelligent Systems Lab, TU Berlin
Marchstr. 23, 10587 Berlin, Germany

Winter 2020/21

Note: In the lecture (Nov 17) we discussed options within the Newton, esp. to deal with non-pos-dev (non-convex) regions. I updated the slides to make this more clear: Slide 18 describes the minimalistic Newton method which assumes strict convexity and therefore pos-dev Hessian everywhere. Slide 20 describes a possible extension with a gradient direction fallback for non-pos-dev Hessian. See the detailed comments at the end of this exercise sheet.

1 Misc

a) The Gauss-Newton method uses the "approximate Hessian" $2\nabla\phi(x)^\top\nabla\phi(x)$. First show that for any vector $v \in \mathbb{R}^n$ the matrix $vv^\top$ is symmetric and semi-positive-definite.[1] From this, how can you argue that $\nabla\phi(x)^\top\nabla\phi(x)$ is also symmetric and semi-positive-definite?

b) In the context of BFGS, we discussed choosing $H^{-1} = \frac{\delta\delta^\top}{\delta^\top y}$ to fulfill the desired relation $\delta = H^{-1}y$. What are alternatives to choose H^{-1}? More specifically, what are all rank-1 matrices A that fulfill the equation $\delta = Ay$. Note that every rank-1 matrix can be represented as $A = uv^\top$ for some vectores u, v (Singular Value Decomposition theorem). Why is $\frac{\delta\delta^\top}{\delta^\top y}$ an appropriate choice among those?

2 Newton method

Consider as previously the functions

$$f_{\text{sq}}(x) = x^\top Cx \ , \tag{1}$$

$$f_{\text{hole}}(x) = \frac{x^\top Cx}{a^2 + x^\top Cx} \ . \tag{2}$$

with diagonal matrix C and entries $C(i,i) = c^{\frac{i-1}{n-1}}$, where n is the dimensionality of x and we choose $c = 10$.

a) First modify your gradient descent with backtracking from the previous exercise simply by multiplying C^{-1} to the step. Compare the performance to the original gradient descent for both, f_{sq} and f_{hole}.

Performance can best be judged by plotting cost function value over the number of function evaluations.

How does the performance change for higher dimensions, e.g., $n = 100$?

b) In the previous exercise we derived the Hessians $\nabla^2 f_{\text{sq}}(x)$ and $\nabla^2 f_{\text{hole}}(x)$. Now implement a proper Newton method. Note that the Hessian of f_{hole} is not always pos-def – use a gradient direction fallback (see comment below) to robustify the Newton method.

Again, compare performances. If the method in a) performs better than the Newton method in b) for f_{hole}, could we not always just use the method of a)?

[1] A matrix $A \in \mathbb{R}^{n\times n}$ is semi-positive-definite simply when for any $x \in \mathbb{R}^n$ it holds $x^\top Ax \geq 0$. Intuitively: A might be a metric as it "measures" the norm of any x as positive. Or: If A is a Hessian, the function is (locally) convex.

3 Convergence proof

a) Given a function $f : \mathbb{R}^n \to \mathbb{R}$ with $f_{\min} = \min_x f(x)$. Assume that its Hessian—that is, the eigenvalues of $\nabla^2 f$—are lower bounded by $m > 0$ and upper bounded by $M > m$, with $m, M \in \mathbb{R}$. Prove that for any $x \in \mathbb{R}^n$ it holds

$$f(x) - \frac{1}{2m}|\nabla f(x)|^2 \leq f_{\min} \leq f(x) - \frac{1}{2M}|\nabla f(x)|^2 \ .$$

Tip: Start with bounding $f(x)$ between the functions with maximal and minimal curvature. Then consider the minima of these bounds. Note, it also follows:

$$|\nabla f(x)|^2 \geq 2m(f(x) - f_{\min}) \ .$$

b) Consider backtracking line search with Wolfe parameter $\varrho_{\text{ls}} \leq \frac{1}{2}$, and step decrease factor ϱ_α^-. First prove that line search terminates the latest when $\frac{\varrho_\alpha^-}{M} \leq \alpha \leq \frac{1}{M}$, and then it found a new point y for which

$$f(y) \leq f(x) - \frac{\varrho_{\text{ls}}\varrho_\alpha^-}{M}|\nabla f(x)|^2 \ .$$

From this, using the result from a), prove the convergence equation

$$f(y) - f_{\min} \leq \left[1 - \frac{2m\varrho_{\text{ls}}\varrho_\alpha^-}{M}\right](f(x) - f_{\min}) \ .$$

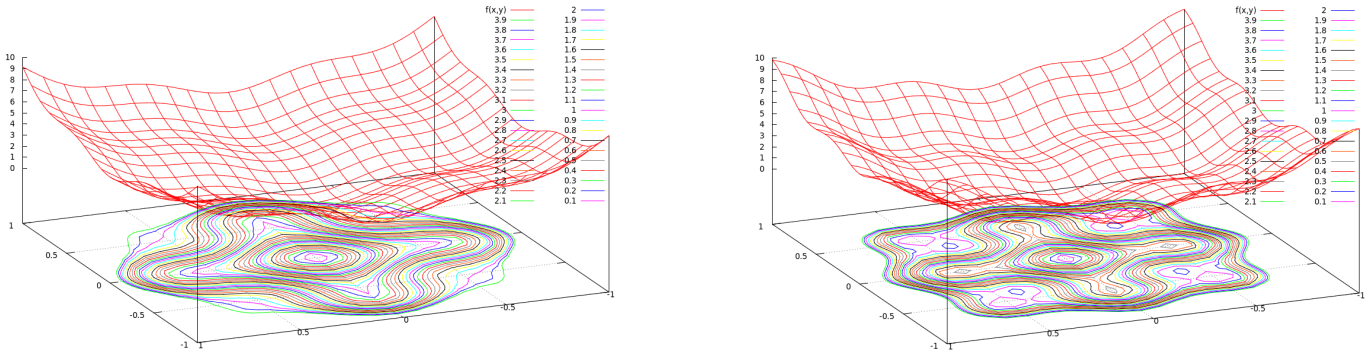
4 Preparatory: Python interfaces to large-scale problems

I'd like to enable you to test your methods also on large-scale problems. This means that we will provide you with interfaces to mathematical programs, and you will only have to code the optimization algorithm.

I have prepared a github repo: `https://github.com/MarcToussaint/optimization-course` So far this is only for Ubuntu – we'll try to make it available also for other systems, please report in the lecture how many people would want it to run on another system.

Please try to install the repo and test the opt-2 jupyter notebook. Report on problems. This test defines a mathematical program for a simple (unconstrained) inverse kinematics problem. (It also calls a Newton solver – but in the future you should test your own solvers and those of external libraries, and not use the one provided.)

5 Voluntary: Gauss-Newton example



In $x \in \mathbb{R}^2$ consider the function

$$f(x) = \phi(x)^\top \phi(x) \ , \quad \phi(x) = \begin{pmatrix} \sin(ax_1) \\ \sin(acx_2) \\ 2x_1 \\ 2cx_2 \end{pmatrix}$$

The function is plotted above for $a = 4$ (left) and $a = 5$ (right, having local minima), and conditioning $c = 1$. The function is non-convex.

Extend your Newton method to use the Gauss-Newton approximation of the Hessian to solve the unconstrained minimization problem $\min_x f(x)$ for a random start point in $x \in [-1, 1]^2$. Compare the algorithm for $a = 4$ and $a = 5$ and conditioning $c = 3$ with gradient descent.

Comments on the Newton method with non-pos-def fallback

Note: The particular method suggested on slide 20 is rather subjective – it is what we found highly practical for large-scale robotics problems. But other extensions might be better in other applications; and existing optimization libraries use other tricks to robustify their Newton method.

- Line 3 of the method on slide 20 says "try to". This assumes that a solver might fail to solve $(\nabla^2 f(x) + \lambda \mathbf{I})\ \delta = -\nabla f(x)$ for δ. This is in particular the case when the solver is based on a Cholesky decomposition, which is highly efficient but only defined for pos-def matrices. The Newton method would have to catch the error signal of this solver.
- Other solvers can solve also non-pos-dev linear equation systems, but then the computed step δ might not point downhill (e.g., it might point to a sattle point or maximum of $f(x)$). To catch this case, line 4 additionally tests whether δ points downhill.
- In these failure cases, the extended Newton method uses the plain gradient direction as the fallback (Line 5).
- Note that the scaling and meaning of α when transitioning between Newton steps and gradient steps is an issue. Both δ's have very different scales and adapting α for one does not translate automatically to the other. A solution might be to maintain separate α's for Newton steps and gradient steps – I have not tested.
- Lines 6, 10 and 14 mention possible heuristics to adapt the damping λ (which is related to adapting the implicit trust region). However, by default, I would not use these options.