Exercises for Lecture Course on Numerical Optimization (NUMOPT)
Albert-Ludwigs-Universität Freiburg – Summer Semester 2025

Exercise 3: Unconstrained Newton-type Optimization

Léo Simpson, Prof. Dr. Moritz Diehl, with contributions from previous teaching assistants

The solutions for these exercises will be given and discussed during the exercise session on May 27th. To receive feedback on your solutions, please hand it in during the exercise session on May 27th, or by e-mail to leo.simpson@imtek.uni-freiburg.de before the same date.

I Root finding of a convex function 1D function

Let $F : \mathbb{R} \to \mathbb{R}$ be a strictly monotonically increasing convex differentiable function such that $F(x^\star) = 0$ for some $x^\star \in \mathbb{R}$.
Show that Newton's method applied to the root-finding problem $F(x) = 0$ is converges $x^\star$.

Solution:

First, let us show that $F(x_{k+1}) \geq 0$ for any k: Since F is convex, it is above its linearizations. In particular, it is above its linearization at the point x_k. But x_{k+1} is a root for the linearization of F at x_k (by definition of the Newton method):

$$F(x_{k+1}) \geq F(x_k) + F'(x_k)(x_{k+1} - x_k) = 0$$

This implies that $F(x_{k+1}) \geq 0$.

Prove that x_k converges x_k is a decreasing sequence for $k \geq 1$:

$$x_{k+1} = x_k - \underbrace{\frac{F(x_k)}{F'(x_k)}}_{>0} \leq x_k$$

Furthermore, it is lower bounded by $x^\star$ (since $F(x_{k+1}) \geq 0 = F(x^\star)$ and F is strictly increasing). Hence, it is a decreasing sequence that is lower bounded: this implies that it converges to some value $\bar{x}$.

Show that $\bar{x} = x^\star$ We have:

$$\underbrace{x_{k+1}}_{\to \bar{x}} = \underbrace{x_k}_{\to \bar{x}} - \underbrace{\frac{F(x_k)}{F'(x_k)}}_{\to \frac{F(\bar{x})}{F'(\bar{x})}}$$

this implies that $F(\bar{x}) = 0$, which implies that $\bar{x} = x^\star$.

II Regularization

Consider a regularized Newton-type step:

$$x_{k+1} = x_k - (B_k + \lambda I)^{-1} \nabla f(x_k)$$

where $x_k \in \mathbb{R}^n$, $B_k \in \mathbb{R}^{n \times n}$ is a (symmetric) Hessian approximation, λ is a positive scalar and I is the identity matrix of suitable dimension.
Prove that when $\lambda \to \infty$, this is similar to a small gradient step:

$$x_{k+1} = x_k - \frac{1}{\lambda} \nabla f(x_k) + \mathcal{O}\left(\frac{1}{\lambda^2}\right)$$

Hint: You can use the following formula for the matrix geometric series:

$$I + A + A^2 + A^3 + \ldots = (I - A)^{-1}$$

for any matrix $A \in \mathbb{R}^{n \times n}$ such that $\rho(A) < 1$.

Solution: Let us choose λ large enough such that $-\frac{1}{\lambda} B_k$ have a spectral radius smaller than 1, and hence such that we can apply the formula from the hint.
Then, we can write:

$$\begin{aligned}
x_{k+1} &= x_k - (B_k + \lambda I)^{-1} \nabla f(x_k) \\
&= x_k - \frac{1}{\lambda} \left(I + \frac{1}{\lambda} B_k \right)^{-1} \nabla f(x_k) \\
&= x_k - \frac{1}{\lambda} \sum_{i=0}^{+\infty} \left(-\frac{1}{\lambda} B_k \right)^i \nabla f(x_k) \\
&= x_k - \frac{1}{\lambda} \nabla f(x_k) - \frac{1}{\lambda} \sum_{i=1}^{+\infty} \left(-\frac{1}{\lambda} B_k \right)^i \nabla f(x_k) \\
&= x_k - \frac{1}{\lambda} \nabla f(x_k) + \frac{1}{\lambda^2} \underbrace{B_k \sum_{i=1}^{+\infty} \left(-\frac{1}{\lambda} B_k \right)^{i-1} \nabla f(x_k)}_{\text{bounded when } \lambda \to +\infty} \\
&= x_k - \frac{1}{\lambda} \nabla f(x_k) + \mathcal{O}\left(\frac{1}{\lambda^2}\right)
\end{aligned}$$

□

III Unconstrained minimization

In this task we will implement different Newton-type methods for solving the problem

$$\underset{x,y\in\mathbb{R}}{\text{minimize}} \quad \underbrace{\frac{1}{2}(x-1)^2 + \frac{1}{2}y^2 + \rho\frac{1}{2}(y-\cos(x))^2}_{=:f(x,y)}.$$

where $\rho > 0$ is a parameter. In the coding parts, we will set $\rho = 5$.
You can use the first part of the provided Python script `plot_objective_fn.py` to get an idea of the shape of the function.

1. Derive (on paper) the gradient vector and the Hessian matrix of the function $f(x, y)$.

Solution: Compute the gradient first:

$$\nabla f(x,y) = \begin{bmatrix} (x-1) - \rho\sin(x)(\cos(x)-y) \\ y + \rho(y-\cos(x)) \end{bmatrix}$$

Differentiate again to get the Hessian:

$$\nabla^2 f(x,y) = \begin{bmatrix} 1+\rho\sin(x)^2 - \rho\cos(x)(\cos(x)-y) & \rho\sin(x) \\ \rho\sin(x) & 1+\rho \end{bmatrix}$$

2. Write the function in the form of the squared-norm of some nonlinear function, i.e.

$$f(x,y) = \frac{1}{2}\left\|R(x,y)\right\|^2 \tag{1}$$

for some function $R : \mathbb{R}^2 \to \mathbb{R}^3$ (often called the *residual function*).

Solution:

$$R(x,y) := \begin{bmatrix} x-1 \\ y \\ \sqrt{\rho}(y-\cos(x)) \end{bmatrix}$$

3. Derive the Gauss-Newton Hessian approximation.

Solution:

$$\nabla R(x,y) = \begin{bmatrix} 1 & 0 & \sqrt{\rho}\sin(x) \\ 0 & 1 & \sqrt{\rho} \end{bmatrix},$$

$$\implies B_{\mathrm{GN}}(x,y) = \nabla R(x,y)\nabla R(x,y)^\top = \begin{bmatrix} 1 & 0 & \sqrt{\rho}\sin(x) \\ 0 & 1 & \sqrt{\rho} \end{bmatrix}\begin{bmatrix} 1 & 0 \\ 0 & 1 \\ \sqrt{\rho}\sin(x) & \sqrt{\rho}\sin(x) \end{bmatrix}$$

$$= \begin{bmatrix} 1+\rho\sin(x)^2 & \rho\sin(x) \\ \rho\sin(x) & 1+\rho \end{bmatrix}$$

4. Under which condition(s) on the point (x, y) for the Gauss-Newton Hessian approximation coincide with the exact Hessian? Interpret the result.

Solution: We have:

$$\nabla^2 f(x,y) - B_{\text{GN}}(x,y) = \begin{bmatrix} -\rho\cos(x)(\cos(x)-y) & 0 \\ 0 & 0 \end{bmatrix}$$

Hence, the Gauss-Newton Hessian approximation is exact if and only if one of the following conditions is verified:

$$\begin{cases} & \cos(x) = 0 \\ \text{or} & y = \cos(x) \end{cases}$$

In general, the Gauss-Newton Hessian approximation is exact when for each of the residual, either the value is zero, or the curvature is zero.

5. Complete the file `unconstrained_newton.py` to implement your own Newton method with the three following Hessian approximations:

   - The exact Hessian;
   - The Gauss-Newton Hessian;
   - The steepest descent Hessian: αI with $\alpha = 10$.

   The initial guess will be $(x_0, y_0) = (0, 10)$ and the termination condition is $\|\nabla f(x_k, y_k)\|_\infty \leq 10^{-6}$ (but the algorithm also stops if the maximum number of iterations $N_{\text{max}} = 50$ is reached). Use the option `plot="3D"` to visualise the iterations on the 3D plot.

6. Use the option `plot="values"` and complete the corresponding code to plot the function values as a function of the number of iterations for each Hessian approximation.

   Compare the convergence of the algorithms.

Solution: See Python file `unconstrained_newton_sol.py`

IV Hanging chain, revisited

We revisit the hanging chain problem from the previous exercise sheet.
So far, we assumed that the springs had a rest length $L = 0$, which might have been an assumption that was too strong.
When we have $L > 0$, the potential energy takes the more complicate following form:

$$V_{\text{chain}}(y, z) = \frac{D}{2} \sum_{i=0}^{N} \left(\left\| \begin{bmatrix} y_i \\ z_i \end{bmatrix} - \begin{bmatrix} y_{i+1} \\ z_{i+1} \end{bmatrix} \right\|_2 - L_i \right)^2 + gm \sum_{i=0}^{N+1} z_i \tag{2}$$

Here, the decision variables are $y_1, \ldots, y_N$ and $z_1, \ldots, z_N$ while the extremities of the chain are fixed:

$$y_0 = 0, \quad y_{N+1} = 2, \quad z_0 = 0, \quad z_{N+1} = 1 \tag{3}$$

In this task, we will solve the unconstrained minimization problem of the hanging chain using a Newton type method, with backtracking line-search for globaliation. More preciesly:

- the function and gradient evaluation is already manually implemented in `hanging_chain_functions.py`
- for the Hessian approximation, you will start with $B_0 = 100I$. Then, you should implement a BFGS update at each iteration, and compare the results.
- for the line-search, you should use $\beta = 0.9$ and stop when the Armijo criterion is satisfied with $\gamma = 0.1$.
- The algorithm should stop either when the maximum number of iterations $N_{\max} = 1000$ is reached, or when $\|\nabla V(y, z)\|_\infty < 10^{-3}$.

Complete the file `hanging_chain_next_episode.py` to solve the problem and visualize the solution at each iteration. Compare the convergence speed with or without BFGS updates.

Solution: See Python file `hanging_chain_next_episode_sol.py`