NTNU

Kunnskap for en bedre verden

TTK4135 - Optimization and Control

Exercise #9

Author:
Sondre Pedersen

March 19, 2025

Problem 1: Unconstrained Optimization

In this problem we will find the minimizer of the Rosenbrock function

$$f(x) = 100(x_2 - x_2^2)^2 + (1 - x_1)^2.$$

using linesearch optimization with the steepest descent, Newton and quasi-Newton (BFGS) directions.

(a) Implement the steepest descent algorithm using the backtracking linesearch method. Use initial guess $x_0 = \begin{bmatrix}1.2 & 1.2\end{bmatrix}^\top$ and initial step length $\alpha_0 = 1$.

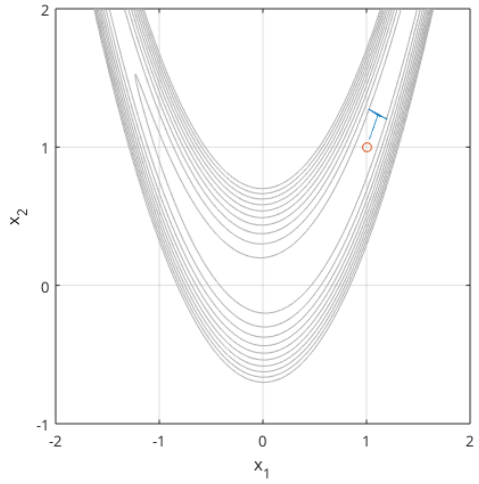
(b) Using the code in (a) as starting point, implement linesearch optimization with the Newtin direction and backtracking linesearch. Use $x_0 = \begin{bmatrix}1.2 & 1.2\end{bmatrix}^\top$ and $x_0 = \begin{bmatrix}-1.2 & 1\end{bmatrix}^\top$, with $\alpha_0 = 1$.

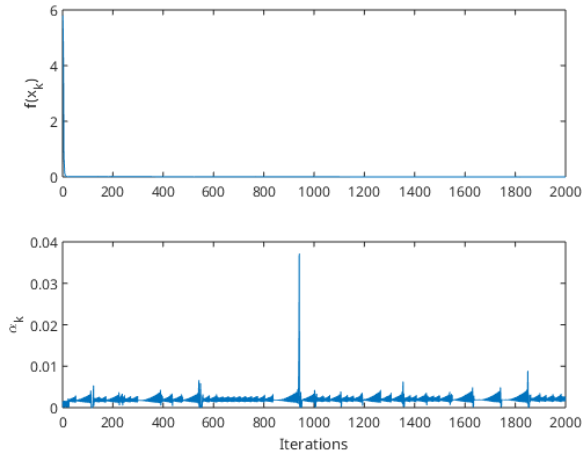
(c) Modify the code to use the BFGS method instead of exact Newton direction.

(d) Study the plots that show α_k at each iteration of k for the Newton and BFGS algorithms. How common is a step length of $\alpha_k = 1$? In particular, what is the step length close to the solution? How does this agree with convergence-rate theory?

(a) Steepest descent

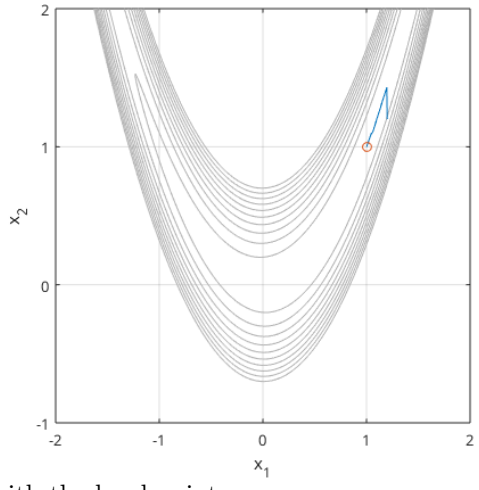
Here are the results of running OptimizeRosenbrock.m with steepest descent:

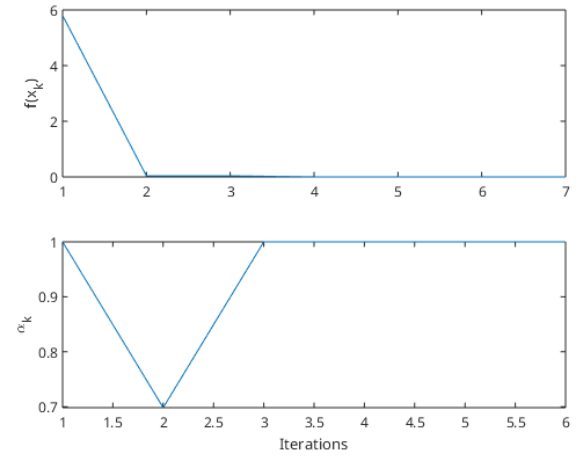




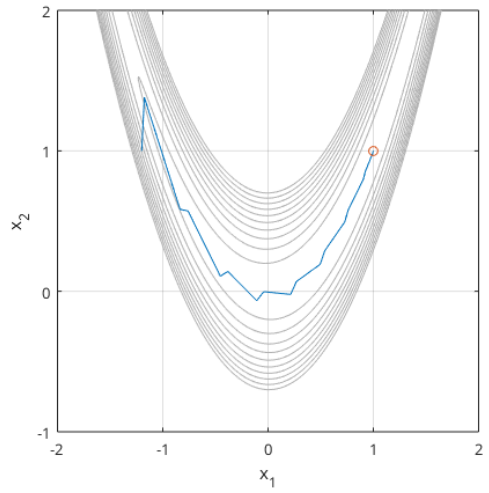
(b) Newton

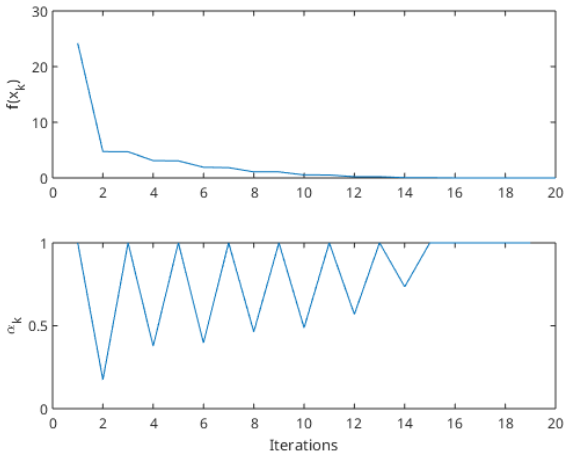
Newton with the easy point:





Newton with the hard point:





(c) BFGS

Come back to this one

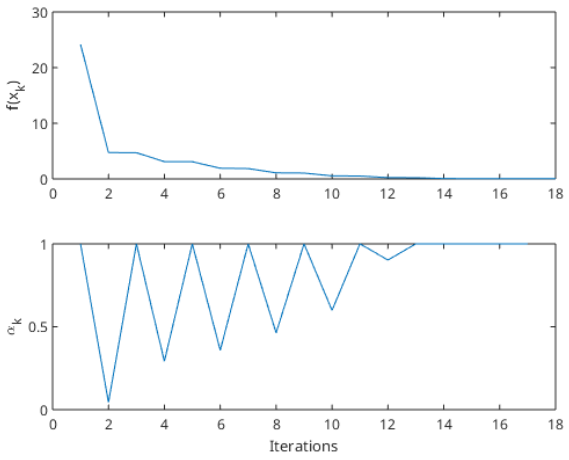
(d) Convergence

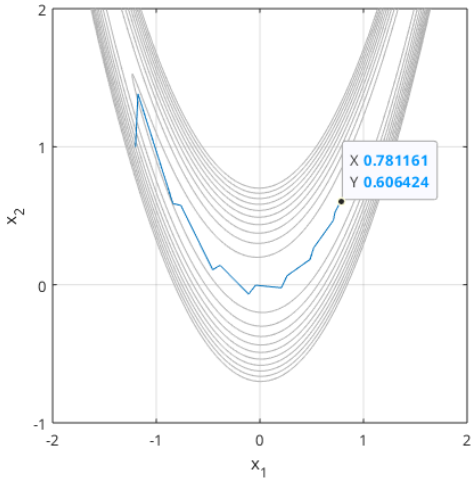
Step length of $\alpha_k = 1$ is expected for Newton and BFGS when getting closer to the solution. This is because they approximate a quadratic equation, and will take a full step right to the solution. This is not true when far away from the optimal point, because the Hessian (or the approximation in BFGS) might not represent local curvature well. At these points the search needs to adjust α_k.

Problem 2: Unconstrained optimization using finite differences

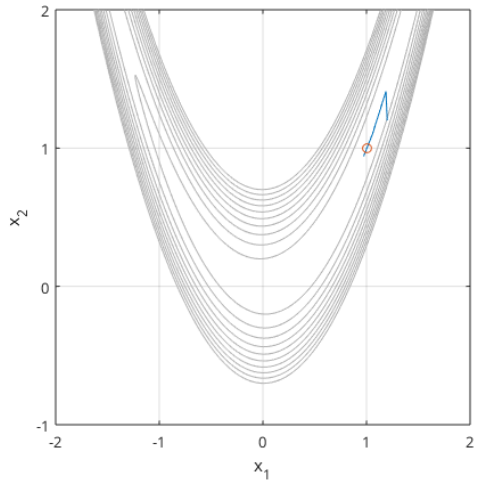
Optimize the Rosenbrock function using Newton's method by using finite differences instead of exact derivatives.

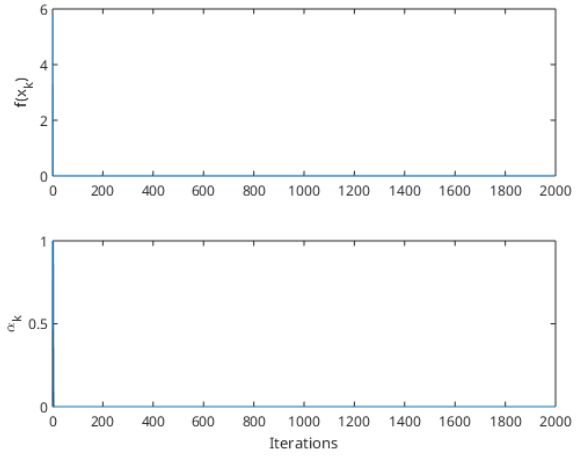
The hard initial guess: $x_0 = \begin{bmatrix} -1.2 & 1 \end{bmatrix}^\top$ looks reasonable.





The easy point ($\begin{bmatrix} 1.2 & 1.2 \end{bmatrix}^\top$) seems to have some issue:





but I am not completely sure why. The number of iterations here is wild. I'll have to look into what is causing this.

Problem 3: Nelder-Mead

This problem will minimize the Rosenbrock function using the Nelder-Mead algorithm (a derivative-free method).

(a) The Nelder-Mead method needs n+1 starting points for an n-dimensional function. What are the conditions on the n+1 initial points? Give an example of an invalid set of starting points. Describe in geometric terms the difference between valid and invalid sets of starting points for a function of two variables.

(b) Study the code for Nelder Mead. Run the provided with different initial points.

(c) Use $x = \begin{bmatrix}1.2 & 1.2\end{bmatrix}^\top$ as the initial point (this point is close to the optimum $x^* = \begin{bmatrix}1 & 1\end{bmatrix}^\top$) and run the algorithm. Then start the algorithm from $x = \begin{bmatrix}-1.2 & 1\end{bmatrix}^\top$, which is a more difficult starting point. Study the output and the plots. Compare results with Problem 1.

(d) Show that if f is a convex function, the shrinkage step in the Nelder-Mead simplex method will not increase the average value of the function over the simplex vertices.

(a) Starting points

The Nelder-Mead method requires n+1 starting points for an n-dimensional function. These points must satisfy the following conditions:

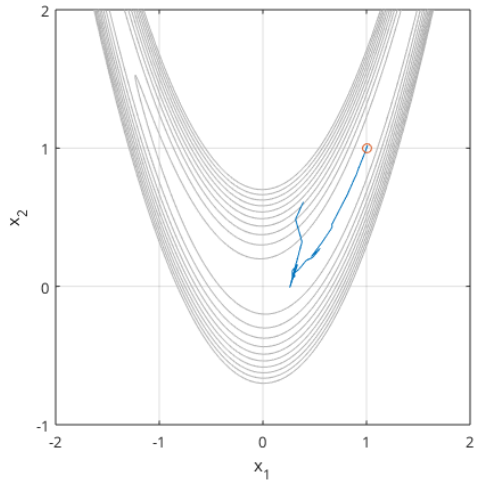
- They must form a non-degenerate simples: In n dimensions, an n-dimensional simples is a set of n+1 independent points. This ensures that the simplex spans an n-dimensional space and can be properly contracted, expanded, or reflected during the optimization process.
- No collinear (or coplanar in higher dimensions) points: If the points lie on the same line in 2D (or on the same plane in 3D), the algorithm can't explore the space effectively.

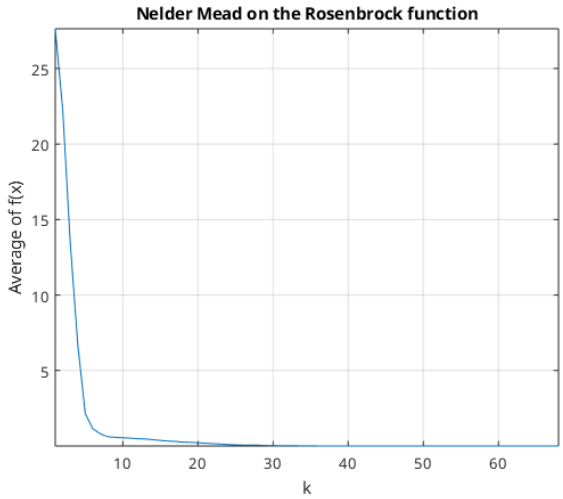
An example of invalid starting points for a function of two variables could be $(0,0)$, $(1,1)$, (2,2)

The geometric difference between valid and ivalid sets is that a valid set forms a shape with the maximal number of dimensions. An invalid set uses points that lie on the same line in 2D, or on the same plane in 3D.

(b) Running Nelder-Mead

Here is an example from starting point $\begin{bmatrix}0.34 & 0.66\end{bmatrix}^\top$.

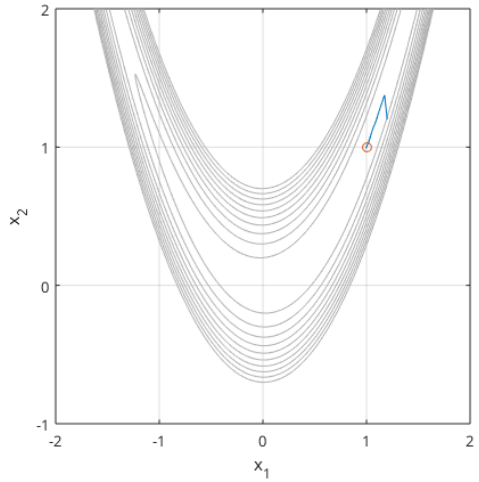


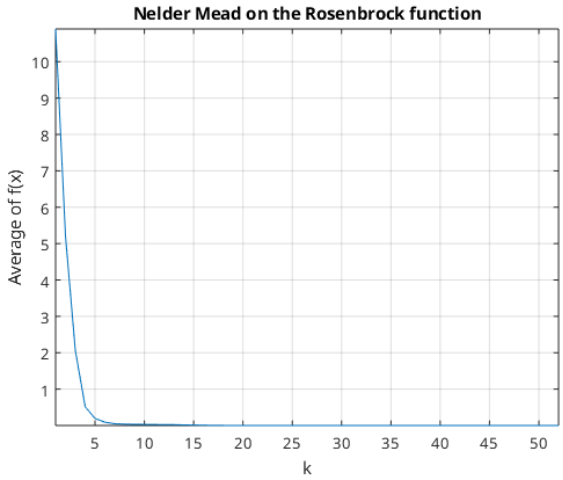


The solution is reached in a little over 60 iterations.

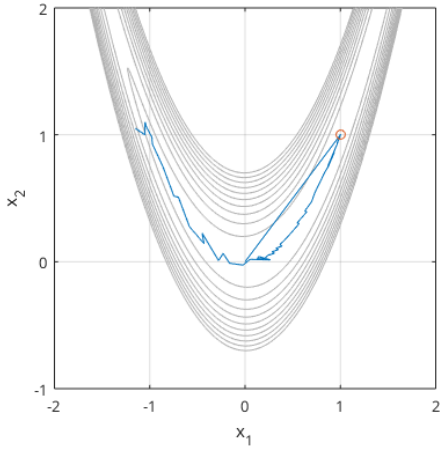
(c) Comparison with previous methods

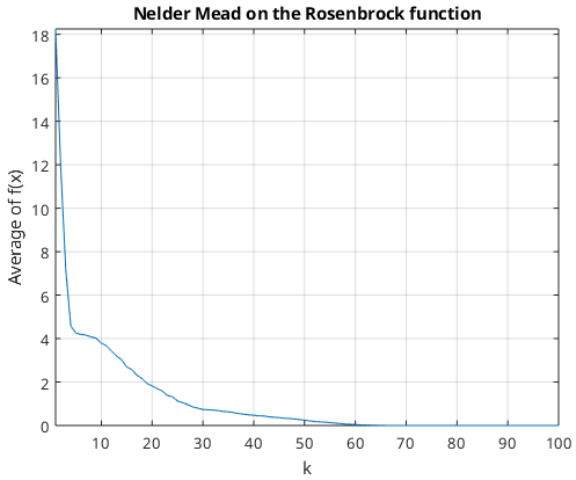
The easy point $\begin{bmatrix}1.2 & 1.2\end{bmatrix}^\top$:





The hard point $\begin{bmatrix}-1.2 & 1\end{bmatrix}^\top$:





For some reason the last iteration from the difficult starting point just goes to (0,0). Other than that they are almost the same as the other methods in number of iterations at least.