TTK4135 Optimization and Control
Spring 2013

Norwegian University of Science and Technology
Department of Engineering Cybernetics

Exercise 6
Solution

Problem 1 (50 %) Unconstrained Optimization

a) Simple example implementations are posted on it's:learning under the Assignment 6 folder. The file names are min_rosenbrock_sd.m (steepest descent) and min_rosenbrock_newton.m.

The condition in the loop of Algorithm 3.1 ensures that the chosen step length α gives a sufficient decrease in the objective function in the direction p_k. This is the first Wolfe condition. It also ensures that this α is not too short, which is part of the motivation for the second Wolfe condition.

The performance of the steepest descent algorithm is illustrated in Figures 1, 2, 3, and 4. Note that the algorithm does not reach the optimum in 2000 iterations from either starting point (it does get fairly close when starting from $x = [1.2,\ 1.2]^\top$). This is related to the zigzag pattern of the iterates, which is characteristic for the steepest descent algorithm. Also note that the step lengths α_k are extremely short, and that the initial guess of α at each iteration is chosen like suggested on page 59 in the textbook.

The performance of the Newton algorithm is illustrated in Figures 5, 6, 7, and 8. Note that from the starting points $x = [1.2,\ 1.2]^\top$ and $x = [-1.2,\ 1]^\top$, the Newton algorithm terminates in 7 and 19 iterations, respectively. This is a very large improvement over the steepest descent algorithm, which did not reach the optimum in 2000 iterations from either starting point. Also notice that there is no zigzag pattern in the iterates. Furthermore, the steps are much longer with the Newton method — many of the steps have a length of 1.

b) A simple example implementation of the BFGS algorithm is posted on it's:learning under the Assignment 6 folder. The file name is min_rosenbrock_bfgs.m. Note that the implementation uses both Wolfe conditions in the line search subfunction, as opposed to just the first.

The performance of the BFGS algorithm is illustrated in Figures 9, 10, 11, and 12. We see that the performance of the BFGS algorithm is superior to that of the steepest descent algorithm, but not as good as that of the Newton algorithm. There is also a slight tendency to a zigzag pattern in the iterates. The BFGS algorithm also spends more CPU time than the Newton algorithm on this example. This is not unexpected, since we are able to supply the Newton algorithm with an exact Hessian that is fairly cheap to evaluate and invert. If this was not the case, we would expect the BFGS algorithm to be faster than the Newton algorithm. Many of the step lenghts α_k taken by the BFGD method are 1.

c) A step length α_k is common for both the Newton method and the BFGS method, see Figures 6, 8, 10, and 12. In partuclar, both algorithms take only full steps ($\alpha_k = 1$) close to the solution. This is one of the requirements for local quadratic convergence rate for Newton's method (see Theorem 3.5 and the subsequent discussion) and for local superlinear convergence rate for the BFGS method (see Theorem 3.6 and the preceding discussion).

Problem 2 (5 %) Cholesky Factorization

Away from the solution, the Hessian matrix $\nabla^2 f(x)$ may not be positive definite. This means that the Newton direction may not be a descent direction. We then want to replace the Hessian matrix by a positive definite approximation, to ensure that the Newton direction is a descent direction. This can be done using modified Cholesky factorization, among other techniques. Modified Cholesky factorization guarantees that the modified Cholesky factors exist and are bounded relative to the norm of the actual Hessian; in addition, it does not modify the Hessian if it is sufficiently positive definite (not close to singular).

Problem 3 (15 %) Gradient Calculation

This problem is about calculating the gradients of the function

$$f(x) = 100(x_2 - x_1)^2 + (1 - x_1)^2 \tag{1}$$

a) The forward-difference scheme is defined in equation (8.1) in the textbook as

$$\frac{\partial f}{\partial x_i}(x) = \frac{f(x + \epsilon e_i) - f(x)}{\epsilon} \tag{2}$$

The approximation of $\nabla f(x)$ for (1) using this scheme becomes

$$\begin{aligned}
\nabla f(x) &= \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \end{bmatrix} \\
&\approx \begin{bmatrix} \frac{f(x+\epsilon e_1)-f(x)}{\epsilon} \\ \frac{f(x+\epsilon e_2)-f(x)}{\epsilon} \end{bmatrix} \\
&= \begin{bmatrix} \frac{100(x_2-(x_1+\epsilon))^2+(1-(x_1+\epsilon))^2-(100(x_2-x_1)^2+(1-x_1)^2)}{\epsilon} \\ \frac{100(x_2+\epsilon-x_1)^2+(1-x_1)^2-(100(x_2-x_1)^2+(1-x_1)^2)}{\epsilon} \end{bmatrix} \\
&= \begin{bmatrix} 200(x_1 - x_2) + 2(x_1 - 1) + 101\epsilon \\ 200(x_2 - x_1) + 100\epsilon \end{bmatrix}
\end{aligned} \tag{3}$$

b) The analytical gradient is

$$\nabla f(x) = \begin{bmatrix} 200(x_1 - x_2) + 2(x_1 - 1) \\ 200(x_2 - x_1) \end{bmatrix} \tag{4}$$

The following table compares the approximated gradient with three different values of ϵ to the analytical gradient:

x	$\nabla f(x)$ $\epsilon = 1 \times 10^{-1}$	$\epsilon = 1 \times 10^{-3}$	$\epsilon = 1 \times 10^{-5}$	Analytical
$\begin{bmatrix}0.5\\0.5\end{bmatrix}$	$\begin{bmatrix}9.100\\10.000\end{bmatrix}$	$\begin{bmatrix}-0.899\\0.100\end{bmatrix}$	$\begin{bmatrix}-0.999\\0.001\end{bmatrix}$	$\begin{bmatrix}-1\\0\end{bmatrix}$
$\begin{bmatrix}1\\1\end{bmatrix}$	$\begin{bmatrix}10.100\\10.000\end{bmatrix}$	$\begin{bmatrix}0.101\\0.100\end{bmatrix}$	$\begin{bmatrix}0.001\\0.001\end{bmatrix}$	$\begin{bmatrix}0\\0\end{bmatrix}$

c) Comparing the analytical gradient to the approximation reveals that the approximation error is a linear term in ϵ. From the table we see that the forward difference scheme can generate very poor approximations of the gradient unless a fairly small value of ϵ is chosen.

Problem 4 (30 %) The Nelder-Mead Method

a) The Nelder-Mead method needs $n+1$ starting points for an n-dimensional function. If we let S denote the initial simplex with vertices $\{z_1, z_2, \ldots, z_{n+1}\}$, the requirement on these $n+1$ points is that the matrix

$$V(S) = [z_2 - z_1, z_3 - z_1, \ldots, z_{n+1} - z_1] \tag{5}$$

is nonsingular.

An example of an invalid set of points in $\mathbb{R}^2$ is $S = \{(0,1),(1,2),(2,3)\}$. In geometric terms, an invalid set of points in $\mathbb{R}^2$ form a line, as the example does. A valid set of points in $\mathbb{R}^2$ form a triangle.

c) Using $x = [1.2,\ 1.2]^\top$ as the initial point and running the algorithm with default settings returns the optimum in less than 60 iterations. The performance of the Nelder-Mead algorithm starting from $x = [1.2,\ 1.2]^\top$ is illustrated in Figures 13 and 14. We see from the shape of the curve illustrating the average value of $f(x)$ (Figure 14) that the average function value decreases at every iteration.

d) We now start the algorithm from $x = [-1.2,\ 1]^\top$, which is a more difficult starting point. The performance of the Nelder-Mead algorithm starting from this point is illustrated in Figures 15 and 16. As expected, the algorithm requires more iterations from this starting point than from $x = [1.2,\ 1.2]^\top$. The average function value decreases at every iteration in this case also.

Note that the number of iterations required to reach the optimum is comparable to the BFGS algorithm. Moreover, the CPU time spent by the Nelder-Mead algorithm is on the same order of magnitude as the Newton and BFGS algorithms. However, since none of these algorithms are optimized for speed and efficiency, the comparison should not be used as a basis for any general conclusions about the respective performance of the algorithms.

e) Denote the average value of the function over the simplex vertices at iteration k as

$$\bar{f}_k \stackrel{\text{def}}{=} \frac{1}{n+1}\sum_{i=1}^{n+1} f(x_i) \tag{6}$$

If f is a convex function, then from the definition of convexity (equation (1.4) in the textbook),

$$f(\alpha x + (1-\alpha)y) \leq \alpha f(x) + (1-\alpha)f(y), \quad \text{for all } \alpha \in [0,\ 1] \tag{7}$$

for any two points x and y. If the shrink step is performed, all x_i are replaced by $(1/2)(x_1 + x_i)$ for $i = 2, 3, \ldots, n+1$. With respect to the definition of convexity, let $\alpha = 1/2$, $x = x_1$, and $y = x_i$. Then,

$$f((1/2)(x_1 + x_i)) \leq (1/2)f(x_1) + (1/2)f(x_i), \quad \text{for } i = 2, 3, \ldots, n+1 \tag{8}$$

Since the points are ordered so that $f(x_1) \leq f(x_2) \leq \cdots f(x_{n+1})$, we have that $\frac{1}{2}f(x_1) \leq \frac{1}{2}f(x_i)$ for $i = 2, 3, \ldots, n+1$. Hence,

$$(1/2)f(x_1) + (1/2)f(x_i) \leq (1/2)f(x_i) + (1/2)f(x_i) = f(x_i), \quad \text{for } i = 2, 3, \ldots, n+1 \tag{9}$$

We can then write the inequality (8) as

$$f((1/2)(x_1 + x_i)) \leq f(x_i), \quad \text{for } i = 2, 3, \ldots, n+1 \tag{10}$$

Since x_1 is unchanged by the shrink step, the average value of the function over the simplex vertices at the next iteration (after the shrink step is performed) can be written

$$\bar{f}_{k+1} = \frac{1}{n+1}f(x_1) + \frac{1}{n+1}\sum_{i=2}^{n+1} f(x_i) = \frac{1}{n+1}f(x_1) + \frac{1}{n+1}\sum_{i=2}^{n+1} f((1/2)(x_1 + x_i)) \tag{11}$$

Rewriting $\bar{f}_k$ is a similar fashion allows us to compare the two terms as

$$\bar{f}_k = \frac{1}{n+1}f(x_1) + \frac{1}{n+1}\sum_{i=2}^{n+1} f(x_i)$$

and

$$\bar{f}_{k+1} = \frac{1}{n+1}f(x_1) + \frac{1}{n+1}\sum_{i=2}^{n+1} f((1/2)(x_1 + x_i))$$

Using inequality (10), it is then clear that $\bar{f}_{k+1} \leq \bar{f}_k$, which means that the shrinkage step will not increase the average value of the function over the simplex vertices.

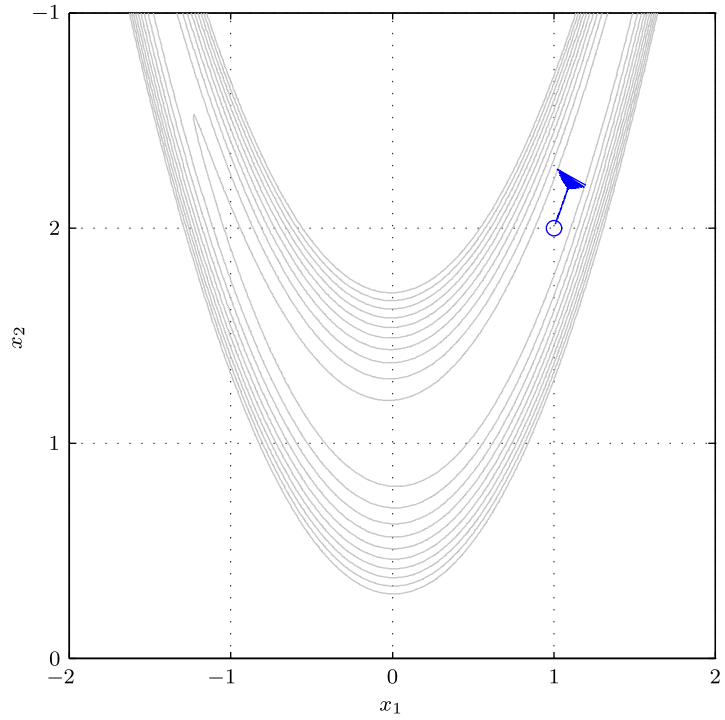

Figure 1: Contour plot showing the progression of the steepest descent algorithm starting from $x = [1.2,\ 1.2]^\top$.

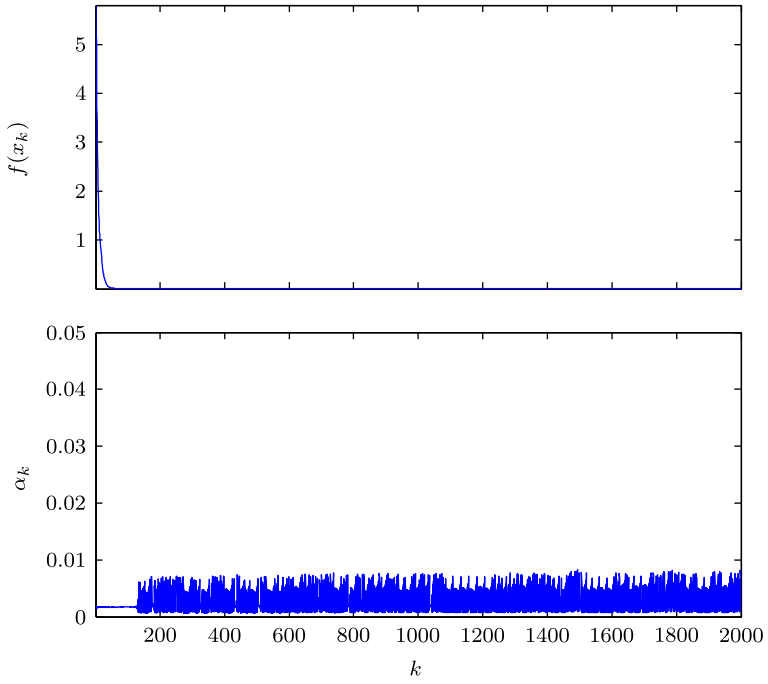

Figure 2: Plots of $f(x_k)$ and α_k for the steepest descent algorithm starting from $x = [1.2,\ 1.2]^\top$.

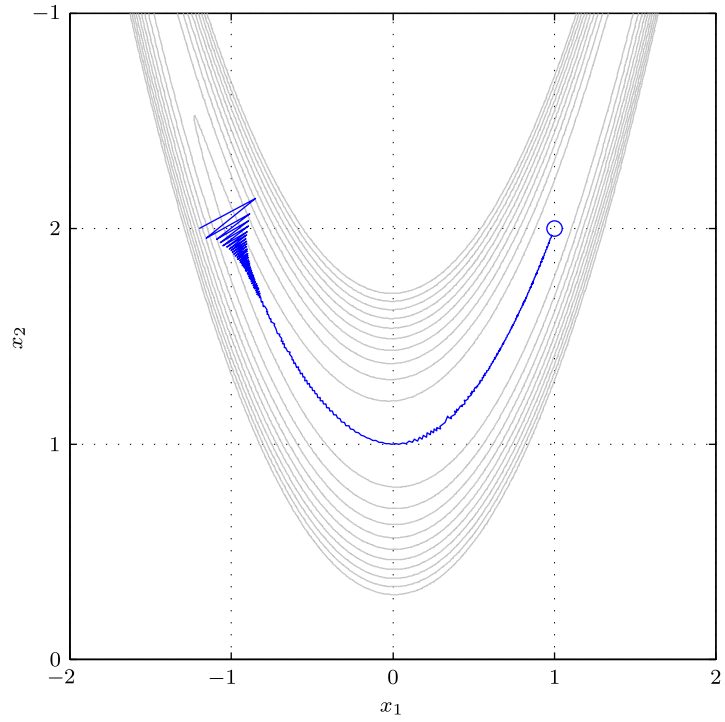

Figure 3: Contour plot showing the progression of the steepest descent algorithm starting from $x = [-1.2,\ 1]^\top$.

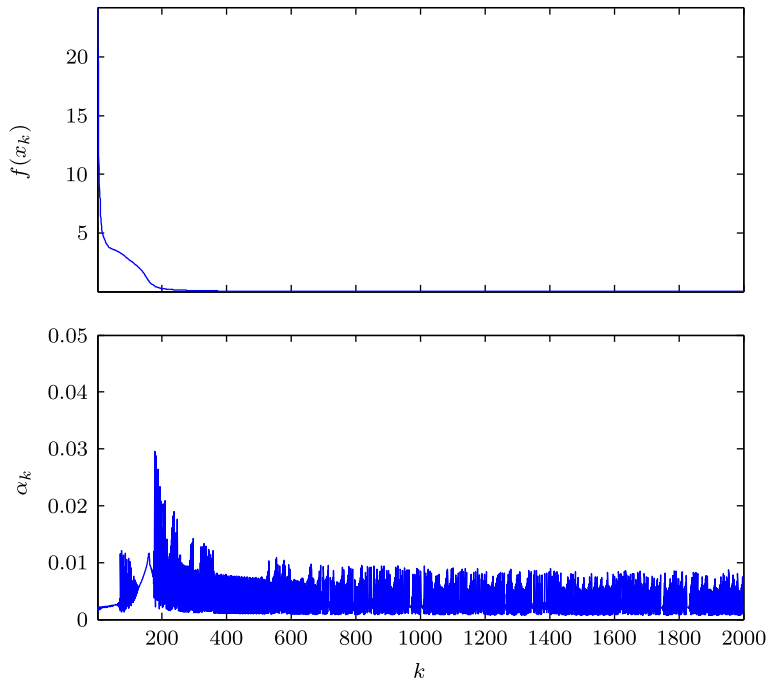

Figure 4: Plots of $f(x_k)$ and α_k for the steepest descent algorithm starting from $x = [-1.2,\ 1]^\top$.

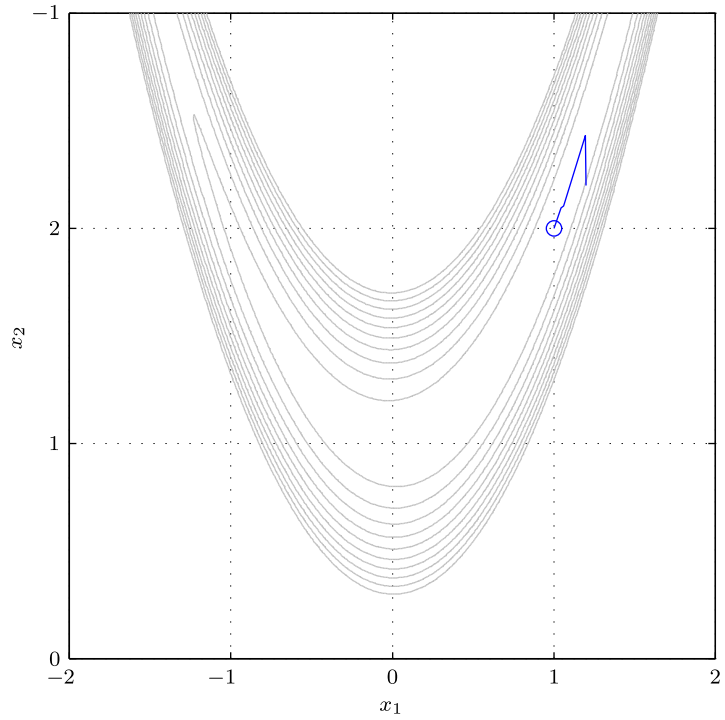

Figure 5: Contour plot showing the progression of the Newton algorithm starting from $x = [1.2,\ 1.2]^\top$.

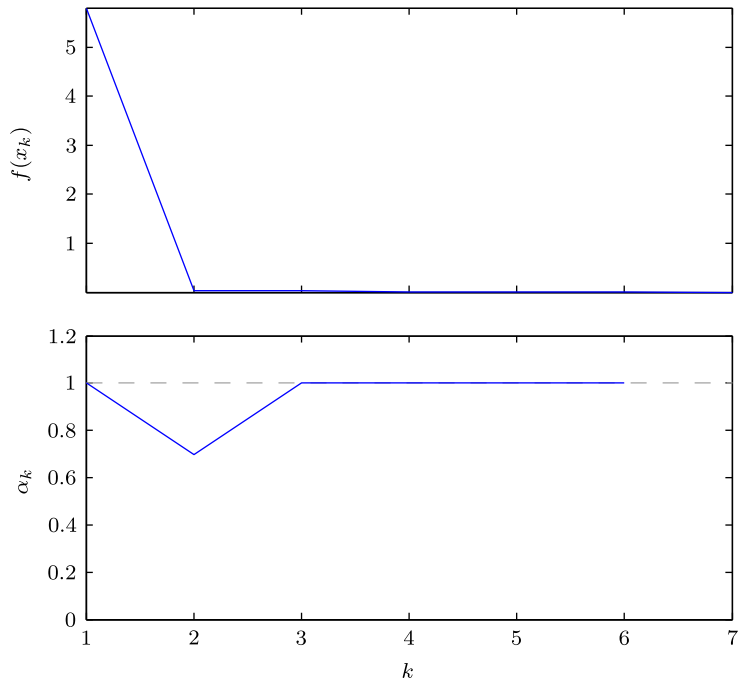

Figure 6: Plots of $f(x_k)$ and α_k for the Newton algorithm starting from $x = [1.2,\ 1.2]^\top$.

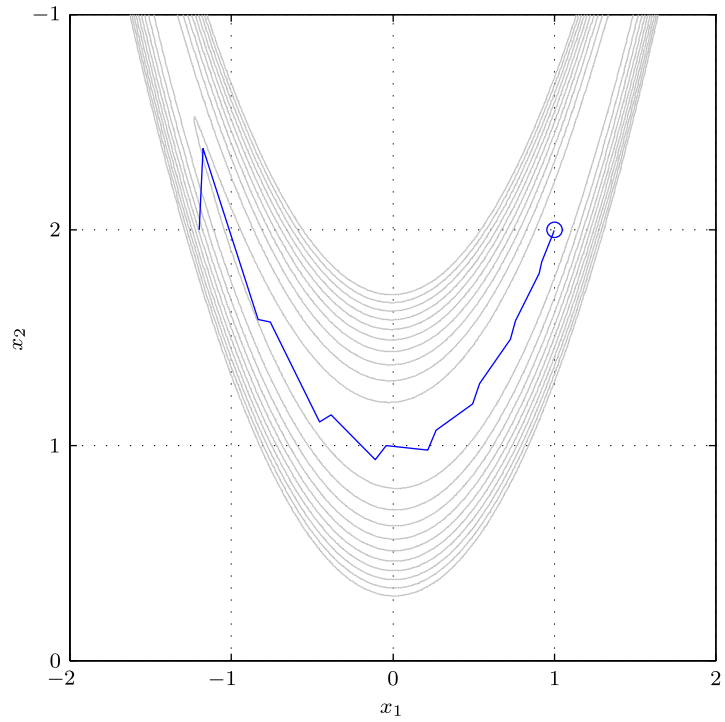

Figure 7: Contour plot showing the progression of the Newton algorithm starting from $x = [-1.2,\ 1]^\top$.

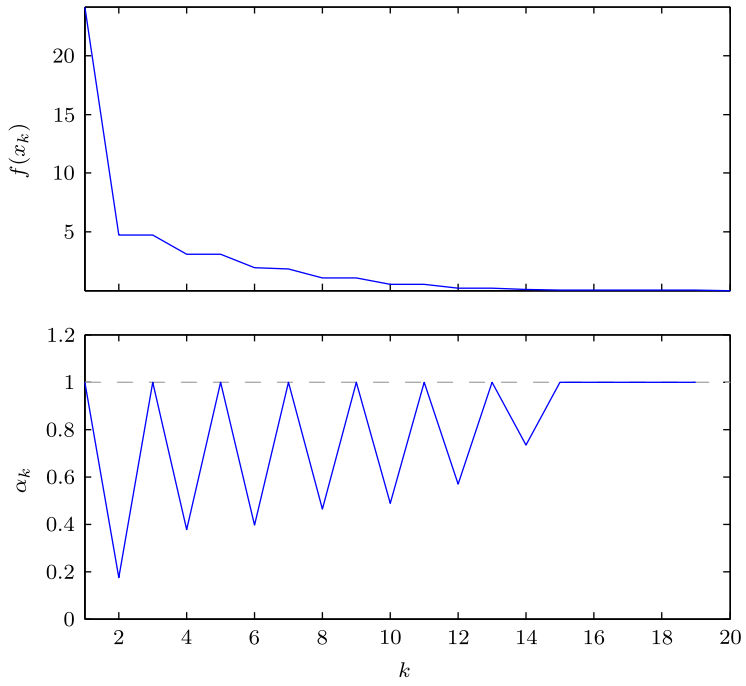

Figure 8: Plots of $f(x_k)$ and α_k for the Newton algorithm starting from $x = [-1.2,\ 1]^\top$.

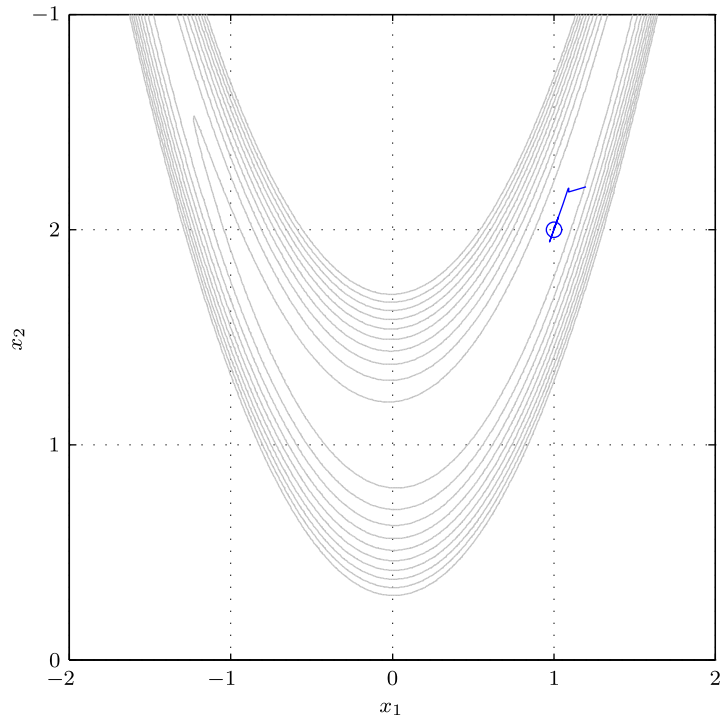

Figure 9: Contour plot showing the progression of the BFGS algorithm starting from $x = [1.2,\ 1.2]^\top$.

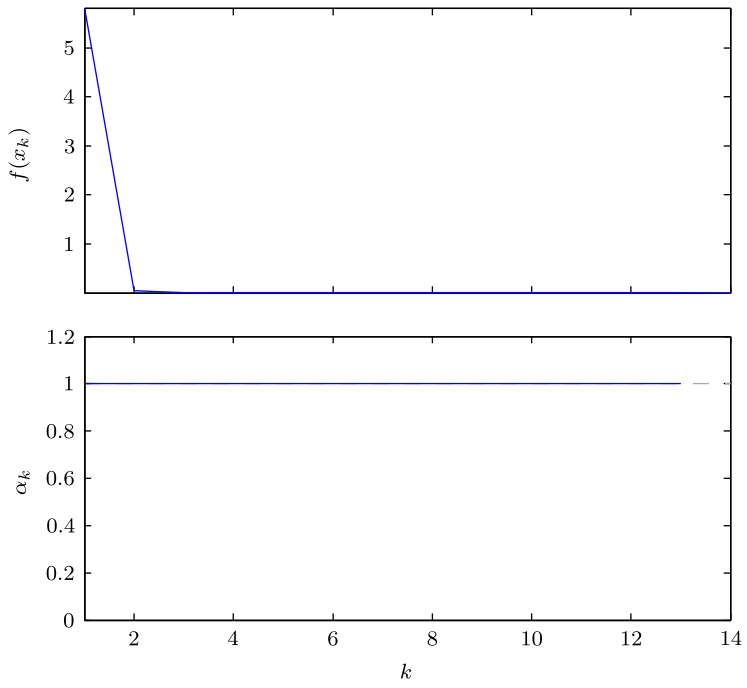

Figure 10: Plots of $f(x_k)$ and α_k for the BFGS algorithm starting from $x = [1.2,\ 1.2]^\top$.

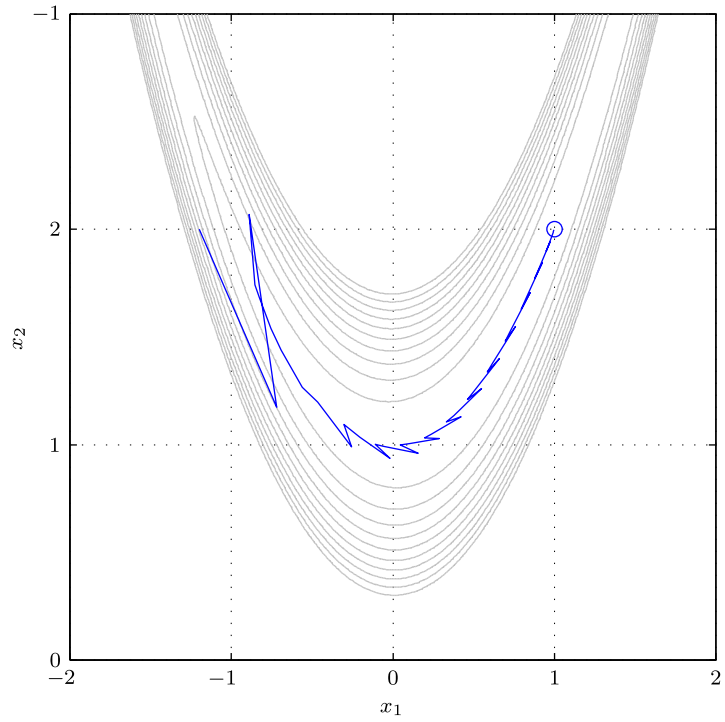

Figure 11: Contour plot showing the progression of the BFGS algorithm starting from $x = [-1.2,\ 1]^\top$.

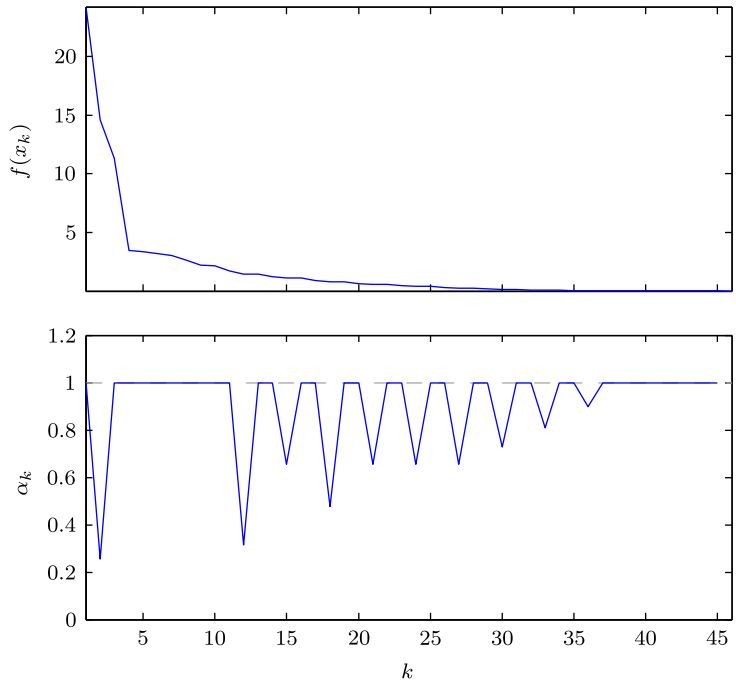

Figure 12: Plots of $f(x_k)$ and α_k for the BFGS algorithm starting from $x = [-1.2,\ 1]^\top$.

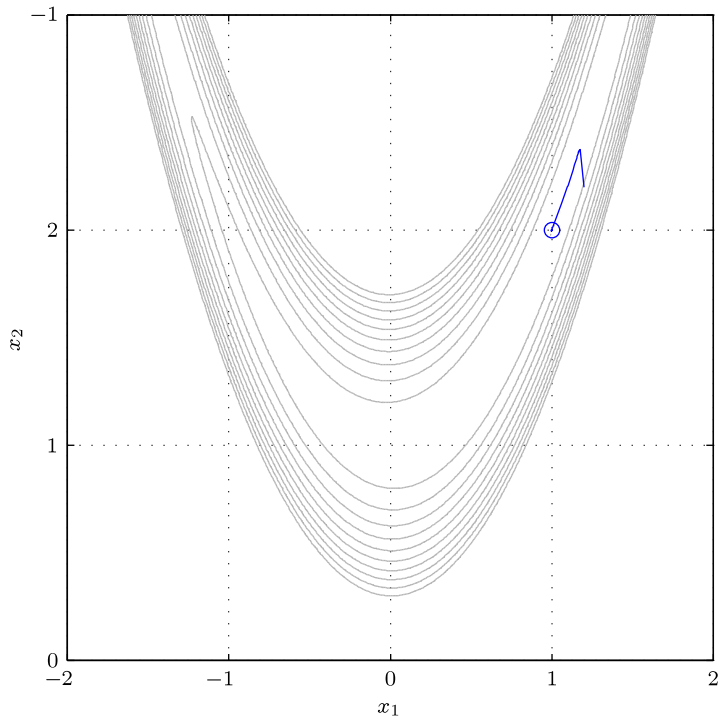

Figure 13: Contour plot showing the progression of the Nelder-Mead algorithm starting from $x = [1.2,\ 1.2]^\top$.

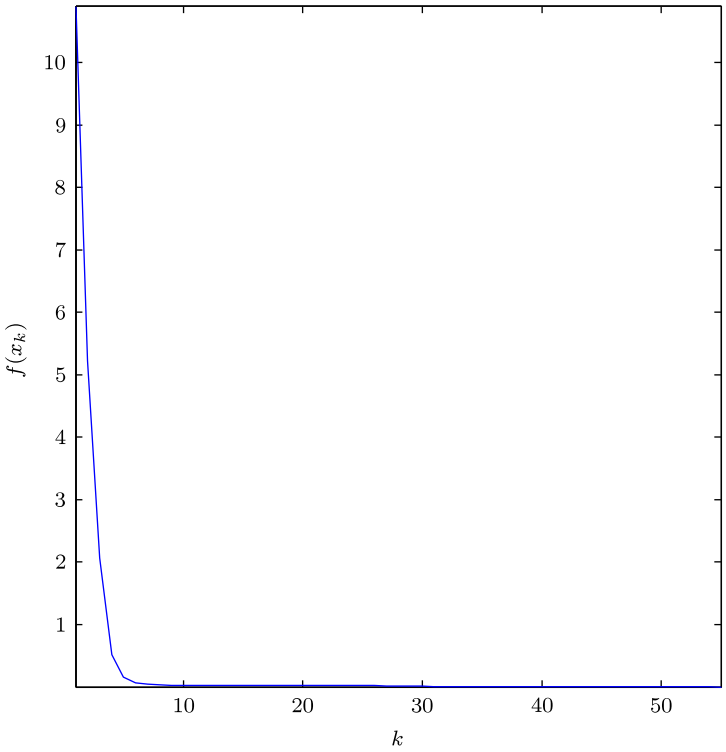

Figure 14: Plot of the average value of $f(x_k)$ for the Nelder-Mead algorithm starting from $x = [1.2,\ 1.2]^\top$.

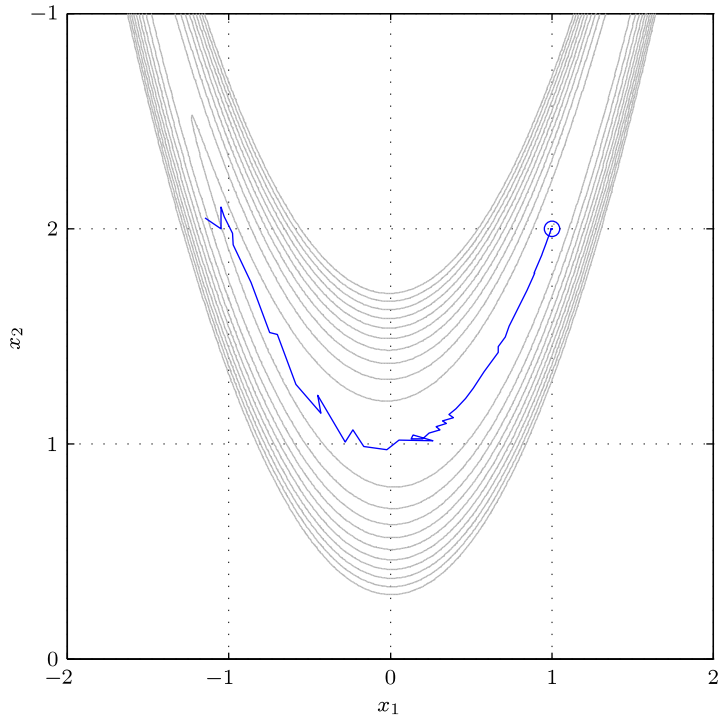

Figure 15: Contour plot showing the progression of the Nelder-Mead algorithm starting from $x = [-1.2,\ 1]^\top$.

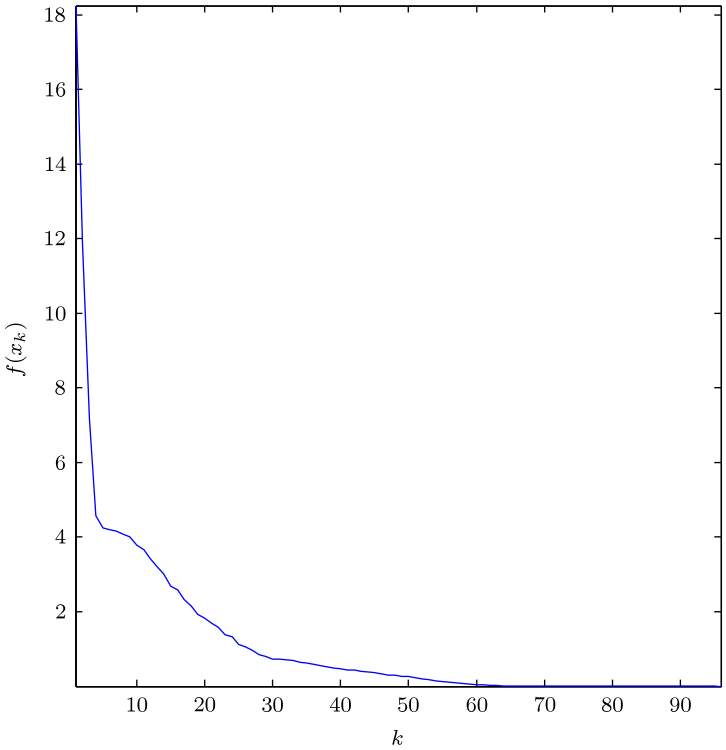

Figure 16: Plot of the average value of $f(x_k)$ for the Nelder-Mead algorithm starting from $x = [-1.2,\ 1]^\top$.