Exercises in Optimization
(ACM 40990 / ACM41030)

Dr Lennon Ó Náraigh

Exercises #1

1. Program the steepest-decent and Newton algorithms using the backtracking line search algorithm. Use them to minimize the Rosenbrock function:

$$f(x, y) = 100(y - x^2)^2 + (1 - x)^2. \tag{1}$$

Set the initial step length $\alpha_0 = 1$ and print the step length used by each method at each iteration. First try the initial point $\boldsymbol{x}_0 = (1.2, 1.2)^T$ and then true the more difficult starting point $\boldsymbol{x}_0 = (-1.2, 1)^T$.

Matlab codes for this exercise can be found in the folder OP_Ros_BT, the main code is shown in the following listings. The Hessian is computable analytically, so to switch from an SD method to a Newton method, it is a simple matter of replacing $\boldsymbol{p}_k = -\alpha_k \nabla f_k$ with $\boldsymbol{p}_k = -\alpha_k B^{-1} \nabla f_k$.

```
function [x_k,f,k,xx,yy] = sd_ros()

% tol: stopping criterion on the norm of gradient
tol = 1e-5;

% maxit: maximum number of iterations
maxit=20000;

xx=0*(1:maxit);
yy=0*(1:maxit);

% BT LS parameters:
c1=0.1;
rho=0.8;

% x0: initial guess for the SD method:
x0=[1.2;1.2];
% x0=[-1.2;1];
x_k=x0;

k=1;
```

```
while 1

    xx(k)=x_k(1);
    yy(k)=x_k(2);
    % Calculation of the cost function.  Here, fun is the cost
    % function, this is defined in a separate Matlab routine and is
    % called here.  The Hessian is known for this problem, that is
    % why it is returned here.

    [f_k,g_k,Hessian] = fun(x_k);

    % Descent Direction:  The Hessian is known for this problem
    % so I can use the full Newton method:

    p_k = -(Hessian^(-1))*g_k;
    % p_k=-g_k/norm(g_k);

    % Optimum Step length, initial guess for Step Length is 1:
    f=f_k;
    a_k=1;
    while(f>f_k+c1*a_k*dot(p_k,g_k))
        a_k=a_k*rho;
        f=fun(x_k+a_k*p_k);
    end

    x_k_new=x_k+a_k*p_k;
    x_k=x_k_new;

    if(mod(k,10)==0)
        display(strcat('Cost Function:',num2str(f)))
        display(strcat('a=',num2str(a_k)))
    end

    if norm(g_k) < tol
        display(strcat('Convergence Reached: |\nabla f|=',...
            num2str(norm(g_k))))
        break;
    end

    if(k == maxit)
        disp('Maximum number of iteration reached');
        break;
    end

    k=k+1;
end

end
```

The code finds the minimizer $\boldsymbol{x}_* = (1,1)^T$ in each case. The code performance for the different starting-points is shown in Table 1. The effect of the choice of $\boldsymbol{x}_0$ on the performance of the algorithms is mixed. However, a clear finding is that the Newton method is faster, i.e. requires fewer iterations to achieve convergence.

Starting Value	Method	$f(\boldsymbol{x}_*)$	$\lvert\nabla f(\boldsymbol{x}_*)\rvert$	Number iterations
$\boldsymbol{x}_0 = (1.2, 1.2)^T$	SD	2.7488×10^{-11}	9.0021×10^{-6}	13,037
$\boldsymbol{x}_0 = (-1.2, 1)^T$	SD	2.6785×10^{-11}	9.0013×10^{-6}	13,065
$\boldsymbol{x}_0 = (1.2, 1.2)^T$	Newton	2.2680×10^{-11}	2.9955×10^{-6}	15
$\boldsymbol{x}_0 = (-1.2, 1)^T$	Newton	2.1218×10^{-11}	3.4709×10^{-6}	20

Table 1: Code performance for the line-search solver (with SWCs) for solving the Rosenbrock problem. Tolerance: 10^{-5}.

To verify our code implementation, we have also calculated the minimum of the Rosenbrock function using the built-in Matlab optimization functions, the listings for which are shown here:

```
function x_star=sd_ros_matlab()

x0=rand(2,1);

fval=@myfun;
x_star = fmincon(fval,x0,[],[],[],[],[],[],[]);

    function y=myfun(x)
        [y,~,~]=fun(x);
    end

end
```

Execution of this code confirms that the minimizer is indeed at $\boldsymbol{x}_* = (1,1)^T$. Further, because this is a simple 2D optimization, the cost function can be studied graphically and the landscape around the minimum can be inspected. We do this in Figure 1–2, where we further show the SD path to the minimum and the Newton path to the minimum. The SD path shows the familiar 'zig-zag' pattern whereas the Newton path is characteristically straight. These different path shapes show intuitively why the Newton method is faster to converge to the minimizer.

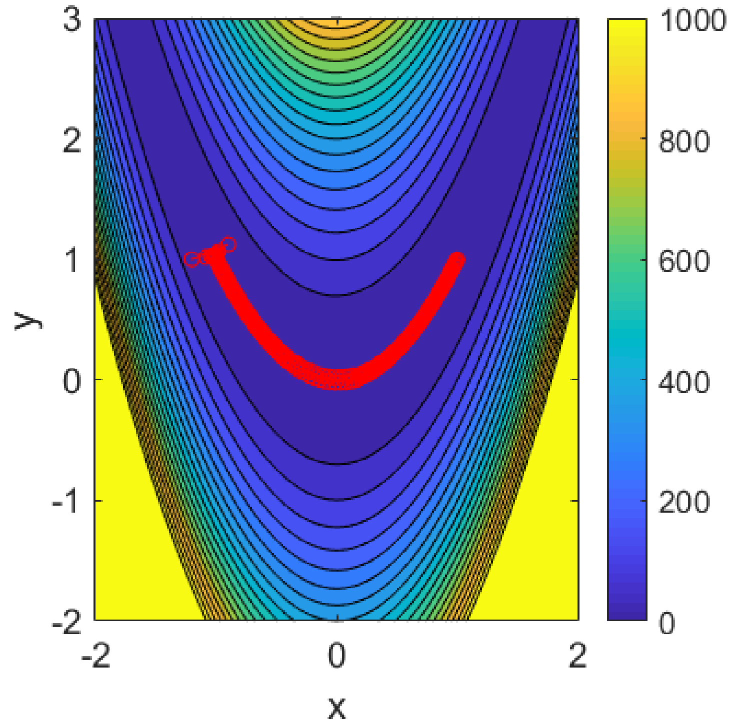


Figure 1: Contour plot of the Rosenbrock function showing the SD path to the minimum

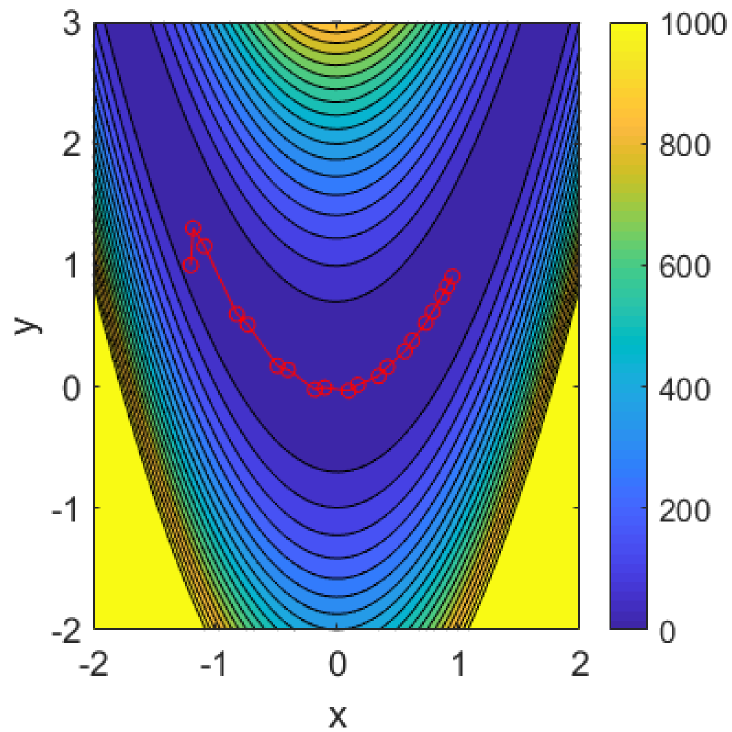


Figure 2: Contour plot of the Rosenbrock function showing the Newton path to the minimum

2. Program the steepest-descent and Newton algorithms with the stepsize determined by the SWCs. Use them to minimize the Rosenbrock function in Equation (1).

Matlab codes for this exercise can be found in the folder `OP_Ros_SWC`. The code finds the minimizer $\boldsymbol{x}_* = (1,1)^T$ in each case. The code performance for the different starting-points is shown in Table 2.

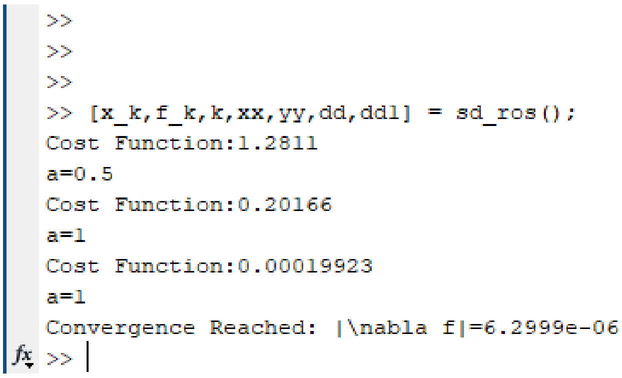
Starting Value	Method	$f(\boldsymbol{x}_*)$	$\lvert\nabla f(\boldsymbol{x}_*)\rvert$	Number iterations
$\boldsymbol{x}_0 = (1.2, 1.2)^T$	SD	3.5151×10^{-11}	1.766×10^{-4}	4964
$\boldsymbol{x}_0 = (-1.2, 1)^T$	SD	3.1317×10^{-11}	1.067×10^{-4}	2303
$\boldsymbol{x}_0 = (1.2, 1.2)^T$	Newton	1.7392×10^{-11}	1.4519×10^{-6}	8
$\boldsymbol{x}_0 = (-1.2, 1)^T$	Newton	2.5518×10^{-11}	1.7003×10^{-7}	21

Table 2: Code performance for the line-search solver (with SWCs) for solving the Rosenbrock problem. Tolerance: 2×10^{-4}.

As before, the effect of the choice of $\boldsymbol{x}_0$ on the performance of the algorithms is mixed. However, it is still the case that the Newton method is faster, i.e. requires fewer iterations to achieve convergence. Overall, the implementation with the SWC is fastest (i.e. faster than the BT Line-search method).

3. Program the BFGS algorithm using the SWCs for the stepsize. Have the code verify that $\langle \boldsymbol{y}_k, \boldsymbol{s}_k \rangle$ is always positive. Use the code to minimize the Rosenbrock function in Equation (1).

Matlab codes for this exercise can be found in the folder OP_Ros_BFGS. Although the Hessian is available analytically, we don't compute it so, the aim of this question (and of BFGS more generally) is to approximate the Hessian numerically. We need a starting-value for the (inverse) Hessian, we take this to be $H_0 = \mathbb{I}_{2\times 2}$. We use the SWCs to compute the stepsize α_k. A screenshot showing the execution of the code is shown in Figure 3, showing good convergence to the minimizer $\boldsymbol{x}_* = (1,1)^T$ from an initial value $\boldsymbol{x}_0 = (-1,2,1)^T$.



```
>>
>>
>>
>> [x_k,f_k,k,xx,yy,dd,ddl] = sd_ros();
Cost Function:1.2811
a=0.5
Cost Function:0.20166
a=1
Cost Function:0.00019923
a=1
Convergence Reached: |\nabla f|=6.2999e-06
>>
```

Figure 3: Screenshot showing the execution of the line-search solver (with BFGS/SWCs) for solving the Rosenbrock problem

The output variables xx and yy record the trajectory of the solution towards the minimum (x- and y-coordinates), these are plotted in Figure 4.

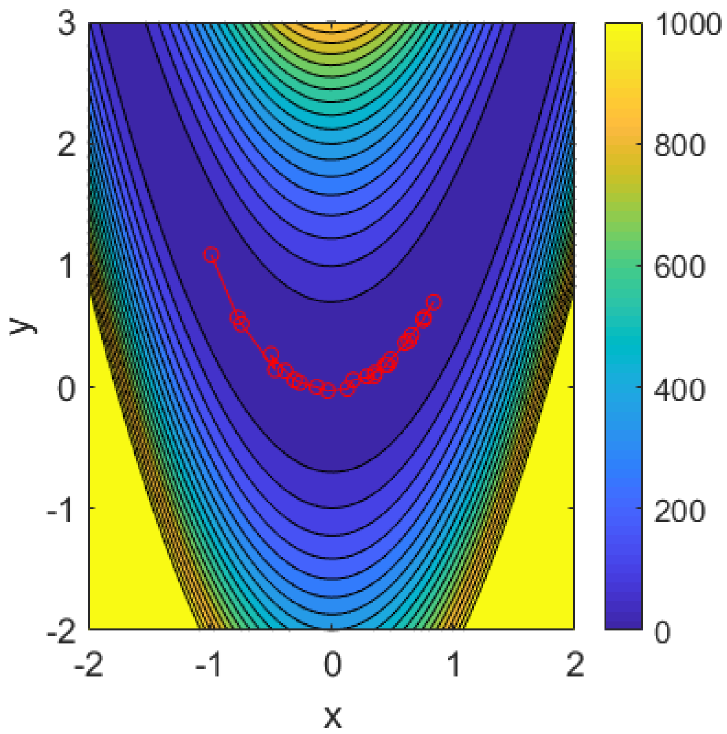


Figure 4: Contour plot of the Rosenbrock function showing the BFGS/SWC path to the minimum

We have further added an additional output variable `dd`, this records the value $\langle \boldsymbol{y}_k, \boldsymbol{s}_k \rangle$ at each iteration. The result is plotted in Figure 5.

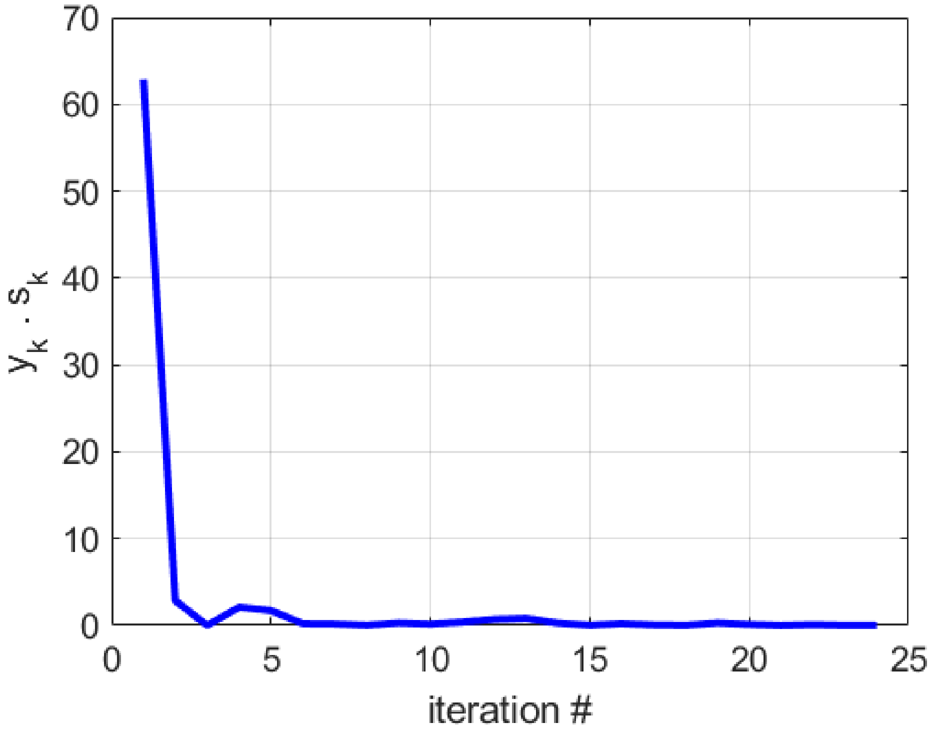


Figure 5: Monitoring the dot product $\langle \boldsymbol{y}_k, \boldsymbol{s}_k \rangle$ in the BFGS code