Foundations of Data Science lectures
Continuous Optimization: Computational Exercises

Dr Lennon Ó Náraigh

8th December 2023

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=10000;

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}	9.87×10^{-6}	368
$\boldsymbol{x}_0 = (-1.2, 1)^T$	SD	2.6785×10^{-11}	9.7104×10^{-6}	383
$\boldsymbol{x}_0 = (1.2, 1.2)^T$	Newton	2.2680×10^{-11}	9.7938×10^{-6}	147
$\boldsymbol{x}_0 = (-1.2, 1)^T$	Newton	2.1218×10^{-11}	9.4728×10^{-6}	150

Table 1: Code performance for the line-search solver for solving the Rosenbrock problem

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.

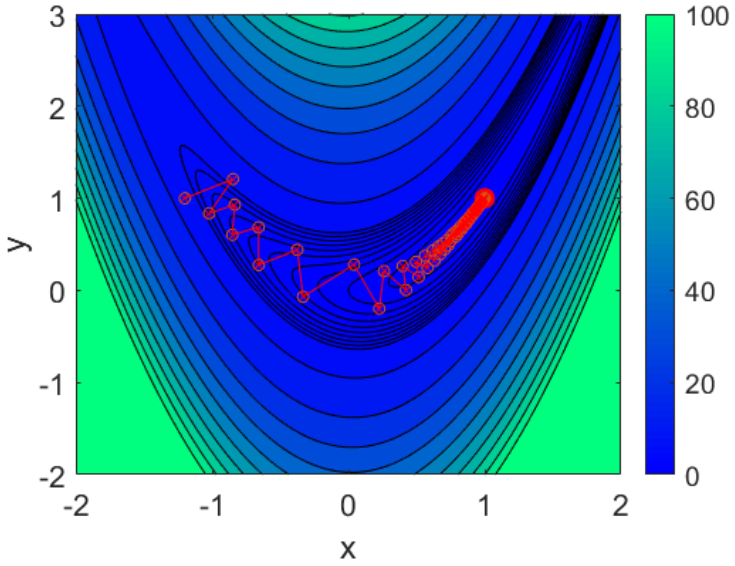

Figure 1: Contour plot of the Rosenbrock function showing the SD path to the minimum

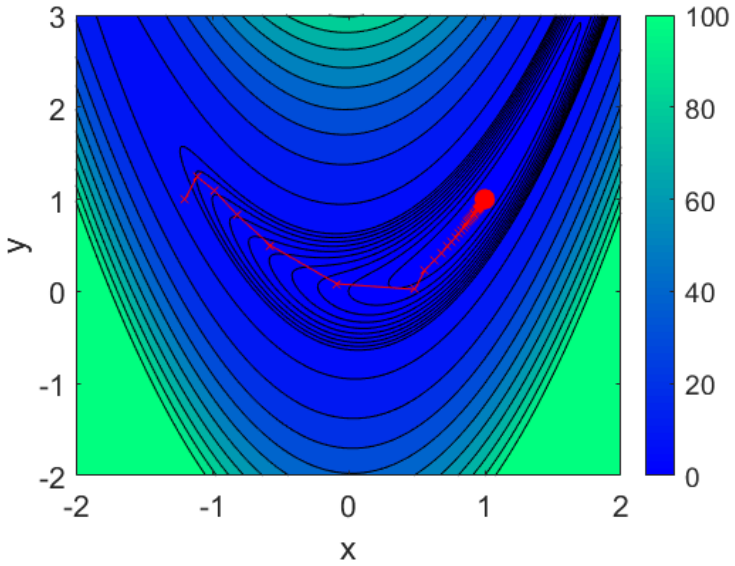

Figure 2: Contour plot of the Rosenbrock function showing the Newton path to the minimum

2. Consider the optimization problem,

$$\min f(\boldsymbol{x}), \qquad f(\boldsymbol{x}) = \langle \boldsymbol{a}, \boldsymbol{x} \rangle + \tfrac{1}{2}\langle \boldsymbol{x}, B\boldsymbol{x} \rangle,$$

where now B is a specific 10×10 matrix and $\boldsymbol{a}$ is a specific 10×1 column vector. The numerical values of these arrays can be found in the spreadsheet `OP_10x10.csv`:

- The spreadsheet contains a 10×1 array which corresponds to the vector $\boldsymbol{a}$;
- The spreadsheet contains a 10×10 array B_0.

The array B is obtained from B_0 by the following sequence of steps:

(i) Symmetrize B_0:

$$B_0 \to (B_0 + B_0^T)/2;$$

(ii) Scale B_0:

$$B_0 \to B_0 / \max(|B_0|)$$

(iii) Generate a positive-definite matrix:

$$B_0 \to (B_0^T) B_0.$$

The end result of this sequence of operations is the matrix B.

Hence,

(a) Find the minimizer $\boldsymbol{x}_*$ numerically, using the steepest-descent and Newton algorithms.

(b) Why is the convergence so poor in the case of the steepest-descent algorithm?

For part (a), the code is similar to Question 1 and is not shown here. The convergence is extremely slow in case of the SD method (for the given numerical parameters); results are not presented here. Instead, we go straight over to the Newton method, where we obtain the following results:

```
>> [x,f,i] = sdd_w()
Cost Function:17.9407
a=1
Cost Function:-225.0464
a=1
Convergence Reached: |\nabla f|=9.1863e-12

x =

   1.0e+03 *

   0.248481416364386
   1.318605978725834
   0.379081869087112
  -0.390457129647163
  -0.820952833950023
  -0.210071124291726
  -0.201216895062331
  -0.738932520228526
   0.847612993079193
  -0.261164500199232


f =

    -2.250464354514670e+02


i =

     2
```

Results using Newton Method

```
>> temp=load('problem.mat');
B=temp.ans.B;
a=temp.ans.a;

% B=B(1:5,1:5);
% a=a(1:5);

% B is initially a random matrix generated from a file, now I turn it into
% a symmetric positive definite matrix.

B=(B+B.')/2;
B=B/max(max(abs(B)));
B=(B.')*B;
>> x_star=-(B^(-1))*a

x_star =

   1.0e+03 *

   0.248481416365052
   1.318605978729388
   0.379081869088129
  -0.390457129648205
  -0.820952833952220
  -0.210071124292290
  -0.201216895062868
  -0.738932520230494
   0.847612993081450
  -0.261164500199958
```

Known Analytical Result

This can be checked, as we know the analytical value of the minimizer in this case, $\boldsymbol{x}_* = -B^{-1}\boldsymbol{a}$, this is also shown in the figure. The numerical and analytical solutions agree, confirming the correct implementation of the Newton method in this case.

For part (b), we look at the condition number of the matrix B:

```
>> cond(B)

ans =

     2.746503264292211e+05
```

This is greater than 10^5 meaning the maximum and minimum eigenvalues are orders of magnitude apart. From the theory for the SD method, we know that:

$$\|\boldsymbol{x}_{k+1} - \boldsymbol{x}_*\|_B^2 \leq \left(1 - \frac{1}{\kappa(B)}\right) \|\boldsymbol{x}_k - \boldsymbol{x}_*\|_B^2,$$

and with $\kappa(B) > 10^5$, we have:

$$\|\boldsymbol{x}_{k+1} - \boldsymbol{x}_*\|_B^2 \lesssim \|\boldsymbol{x}_k - \boldsymbol{x}_*\|_B^2,$$

leading to poor convergence of the SD method in this particular example.