University of Essex

Department of Mathematical Sciences

MA305-7: Nonlinear programming

Lab report 2: Multidimensional search

Registration number: 1900962

Registration number: 1900898

Lecturer:

Dr. Xinan Yang

9 December 2019

1. Functions of several variables

1. Check the results of `z, g` and `H` to see if they have been calculated correctly.

To check the results calculated, we can start by using a formula for the analytic values of the function value, gradient and Hessian matrix (respectively):

$$f(x_1, x_2) =$$

$$\vec{\nabla}\mathbf{f}(x_1, x_2) = \vec{\mathbf{g}}(x_1, x_2) =$$

$$\vec{\nabla} \otimes \vec{\nabla}\mathbf{f}(x_1, x_2) = \mathbf{H}(x_1, x_2) =$$

Where $\otimes$ is the outer product. 2. Try using `x=rand(2,1)`, which produces a random 2-by-1 matrix whose entries are drawn from a uniform distribution on $[0, 1]$, to evaluate `func_quartic` at some other points.

2. Surface and contour plots

1. Experiment by plotting the contours (with and without labels) of Rosenbrock's function, Beale's function, and the Quadratic function

We can start by capturing all the 8 combinations for the outputs:

2. Also look at the gradient vector and Hessian matrix for each of these functions, and check that they are correct.

3. Quadratic Taylor approximations

1. Using the same limits, plot the quadratic Taylor approximation to Rosenbrock's function around different xvector points.

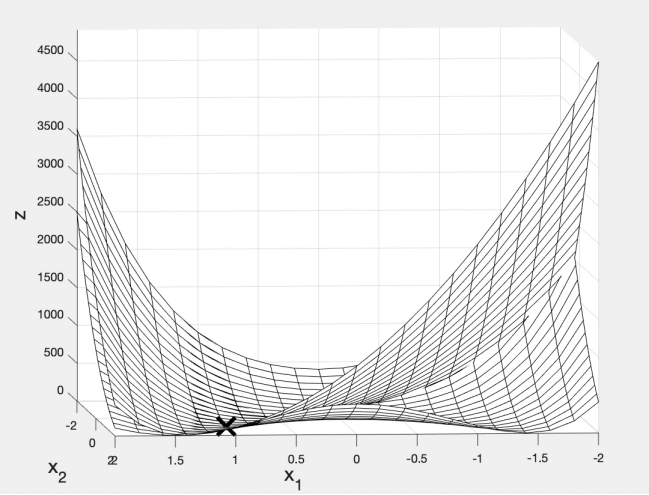


Figure 1: Rosenbrock´s function second order apprx.

2. Try quadratic Taylor approximations for Beale's function using `lims` $= [04 - 12]$.

The plot is the result of the experiment. It shows that the approximation is only good locally.

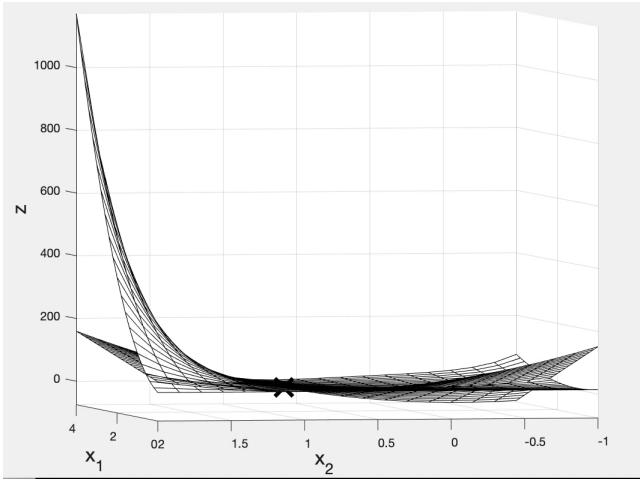


Figure 2: Beale function second order apprx.

3. Using `lims` $= [-22 - 22]$, what happens when you plot the quadratic Taylor approximation to the `func_quad.m` function? Why?

The approximation is the same as the objective function because of Taylor´s theorem.

$$A = \begin{pmatrix} 3 & 0 \\ 0 & 1) \end{pmatrix}$$

$$f(x) = x^T Ax$$

$$\nabla f(x) = 2Ax$$

$$H_f(x) = 2A$$

Thus the taylor approximation is such that,

$$\begin{aligned} T_{f_2}(x,h) &= f(x) + \nabla f(x)^T h + \frac{1}{2} h^T H_f h \\ &= x^T Ax + 2h^T Ax + h^T Ah \\ &= (x+h)^T A(x+h) \\ &= f(x+h) \end{aligned}$$

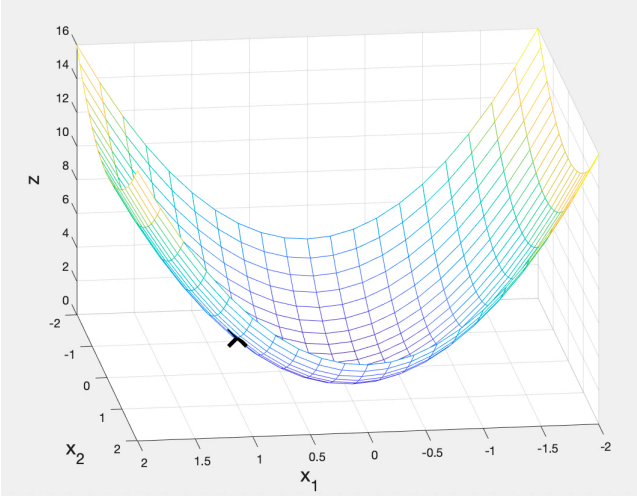
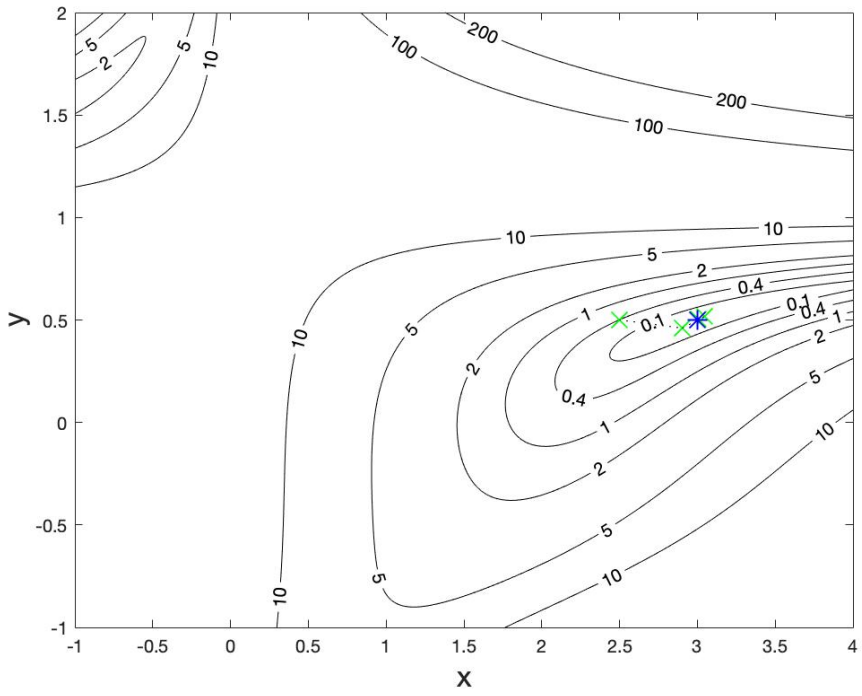


Figure 3: Quadratic function second order apprx.

4. Pure Newton's method

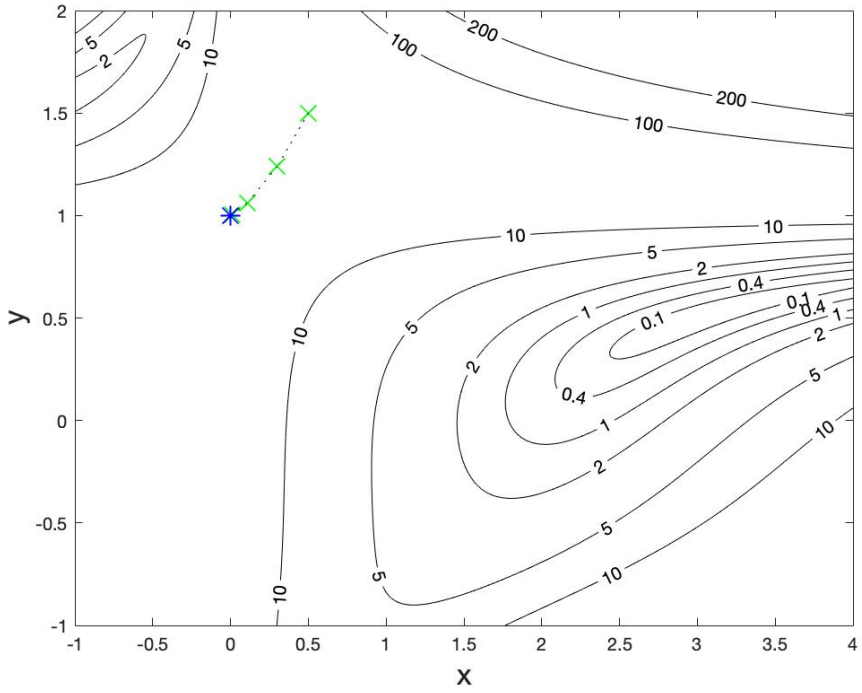
Task 1. For Beale's function, try another initial point, say `xinitial` $= [0.5; 1.5]$. Compare the result with `xinitial` = $[2.5; 0.5]$ and comment on your findings.

The first thing we can do is try the suggested point $[2.5; 0.5]$, just to see the behavior of the function and iterations:



We notice that there are indeed 6 iterations that the function goes through, the 5 green points and the final blue star.

Next, we can try using the point $[0.5; 1.5]$ to see how many iterations it goes through. With this, we get the following plot:



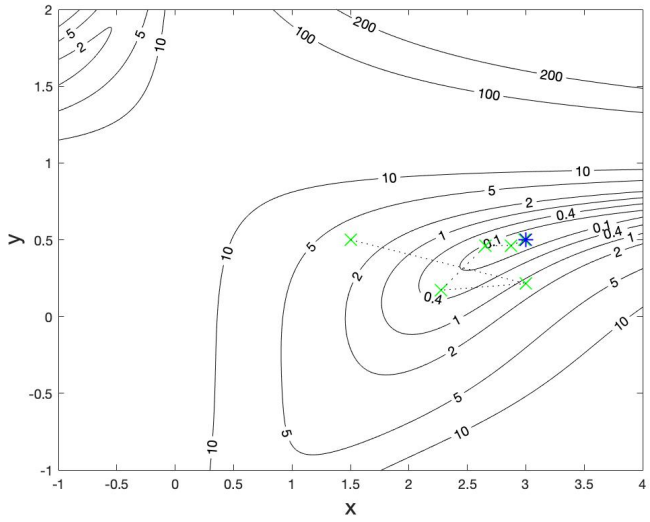
From this plot, it looks like there are 5 iterations, compared to 6 from the previous point. But if we actually count the number of iterations, there are 6 of them just like in the previous case. This means that there are more iterations near the optimum than on the previous case.

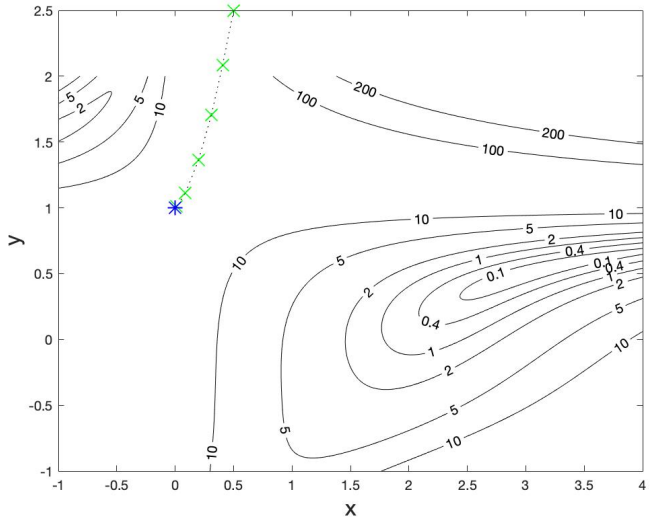
However, there is one more thing we can notice. From the contour lines only, it is not clear whether there is another optimum in the picture besides the one shown in the first example. But from the concavity of the contour lines, it looks like this point is actually a saddle point. Usually around a local maximum/minimum, the contour lines are concave towards the optimum. Here we can see that the equipotential lines are facing away from the point. This is not conclusive because a higher density of lines might reveal that lines closer to the point are facing it.

Fortunately, we can easily evaluate the Hessian matrix exactly and check using the second derivative test. If the eigenvalues of the Hessian have different signs, it means that this really is a saddle point. To evaluate the Hessian we can use the function `func_beale`, and then get the eigenvalues with `eig()`. The result is that the eigenvalues of the Hessian at the point $[0; 1]$ are approximately $[-27.74; 27.75]$. This confirms our observation that the algorithm converged to a

saddle point.

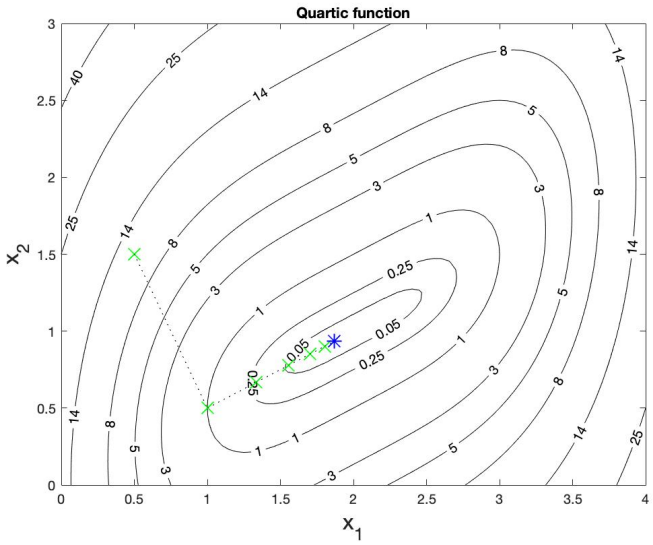
Finally, we also tried other points to try to get a feel of which points converge to the saddle and which to the actual optimum. We tried the two points $[0.5; 2.5]$ and $[1.5; 0]$. These are the two resulting sequences:



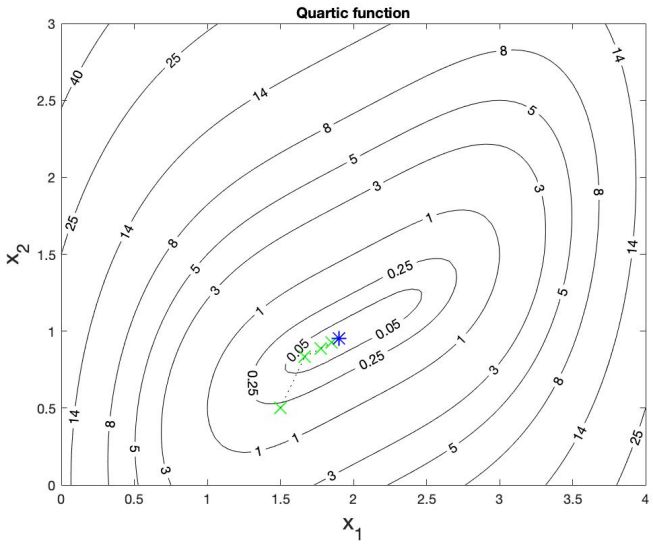


2. Try minimizing the quartic function defined above and experiment with different initial points, some close to the optimal solution, and some farther away. Compare with the experiments of Beale's function and comment on your observations.

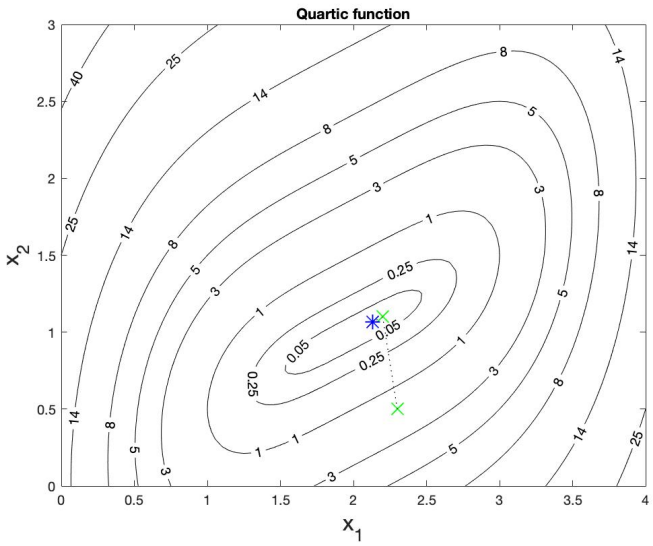
To get an idea of how convergence goes, we can try the points $[0.5; 2.5]$, $[1.5; 0.5]$, $[2.3; 0.5]$:



In this first sequence, there were 8 iterations.



For this one, there were 6 iterations



In this last one there were 4 iterations.

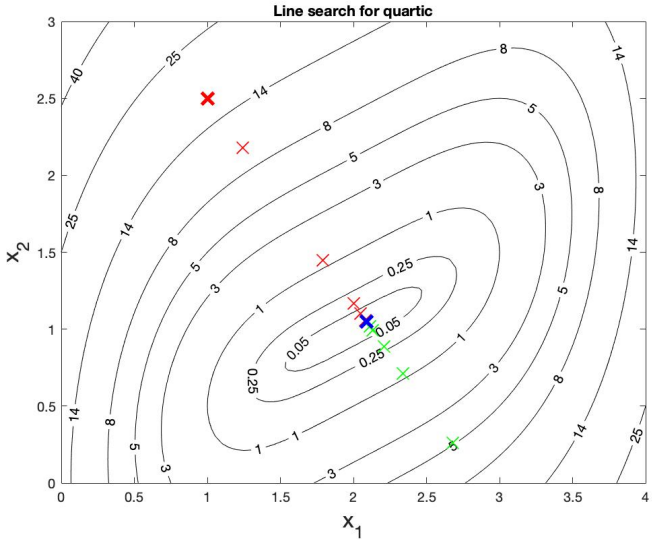
In general, we observe that the points that are furthest from the optimum have the most iterations, which makes sense. Even though we tried placing the points in places where the gradient does not point towards the maximum, it still takes a better step than gradient descent would.

Another property that the quartic function has is that all of its level curves are closed and concave towards the optimum. For example, in Beale's function some choices of the initial point converged to the optimum, while other choices converged to a saddle point. In contrast, every choice for the initial point in the quartic function converges to the optimum. Although quartic functions can have saddle points, from the contour lines we can tell that there are no visible saddle points in this region.

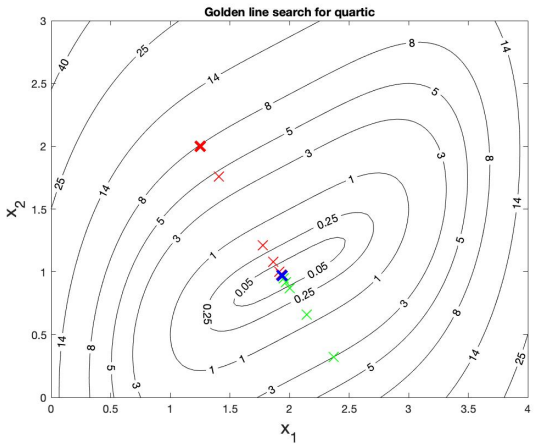
5. Line search

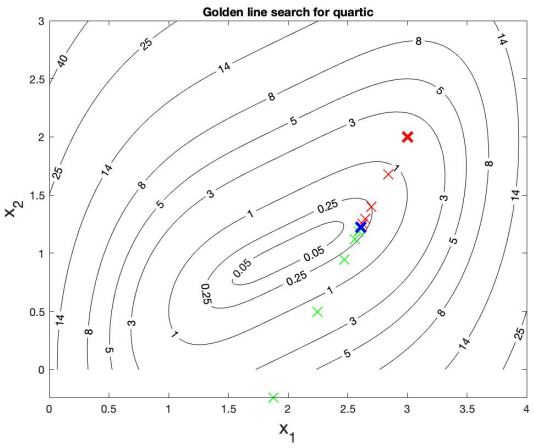
1. Perform a line search from various start points in different directions. Take care that your direction is an improving direction, i.e., that f value decreases along that line.

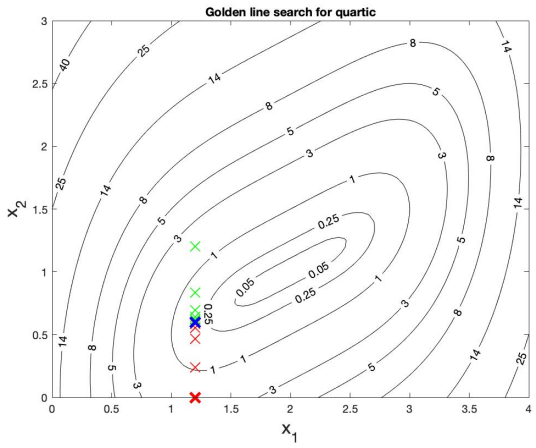
First, we can try out the example to see how this works:

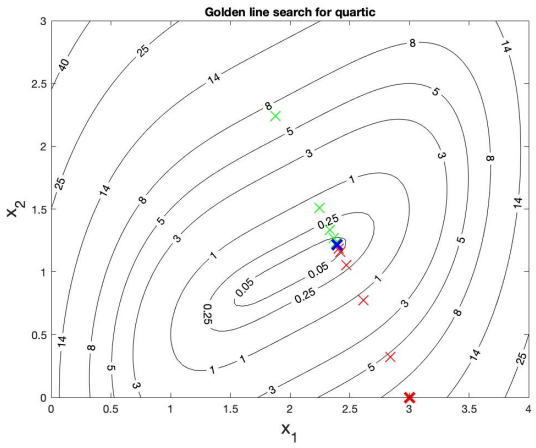


Now that we see how the example works, we can try using other points, and choosing a direction that is roughly looks like an improving direction. We tried the points $[1.25; 2]$, $[1.25; 2]$, $[3; 2]$ and $[1.2; 0]$ in the directions $[2; -3]$, $[-2; -4]$, $[0; 1]$ and $dvector = [-2; 4]$, respectively. These are the resulting plots in that order (left to right, top to bottom):









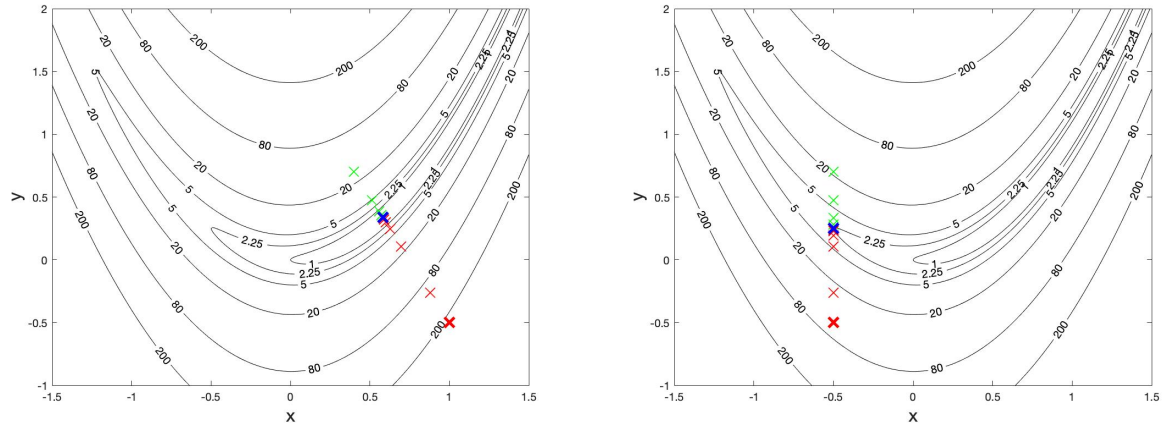
2. You may also like to try some different functions

Naturally, we can try the other 3 functions provided for this lab, given that we already have a nice function to plot the contour lines. We now tried 2 points for each of the 3 functions, each one with their own direction. This makes a total of 6 relevant experiments.

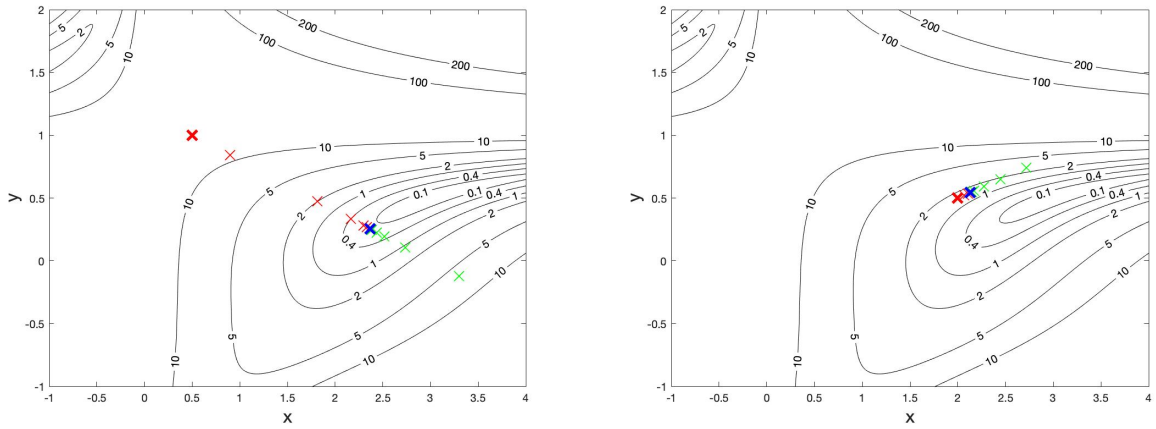
We experimented using all of the points in the table or Rosenbrock's function. We found it interesting that none of them caused infinite iterations. This is a property of the function and

The points and directions that we tried for each function are:

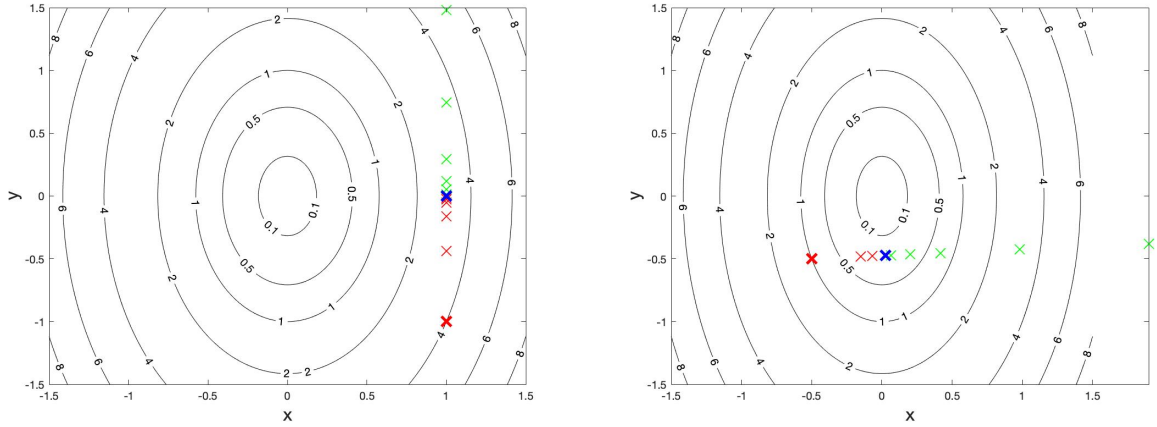
The plots resulting from the experiments are the following:



Rosenbrock's function



Beale's function



Quadratic function

We played around with the direction mostly, and we noticed that for example the Rosenbrock and Beale functions were the ones that have more interesting behavior.

6. Gradient search

1. Above experiment takes 35 iterations to say there is a minimum at approximately (0.8564, 0.7329). Is it close to the true optimum of Rosenbrock's function?

The minima of Rosenbrock´s function is attained at (a, a^2) in this case $a = 1$, so $x^* = (1, 1)$ and has a function value of zero.

$$minarg_x f(x^*) = 0$$

When performing the algorithm and looking at the distance between optima and its approximation, we get:

$$\|x^* - x_{fin}\| = 0.3033$$

Even though the final vector appears to be close, it has a relative error of approx 21.21% and falls out of a 'proper' convergence radius; nevertheless, reducing the size of epsilon will increase the number of iterations by a lot.

2. Show the 'zigzagging' of the above search (you may want to take a closer look by clicking on the "zoom in" button on the figure window). Remember that the alternative move directions of Gradient Search should be orthogonal to each other. Can you see why successive steps are not (exactly) orthogonal to each other in this experiment?

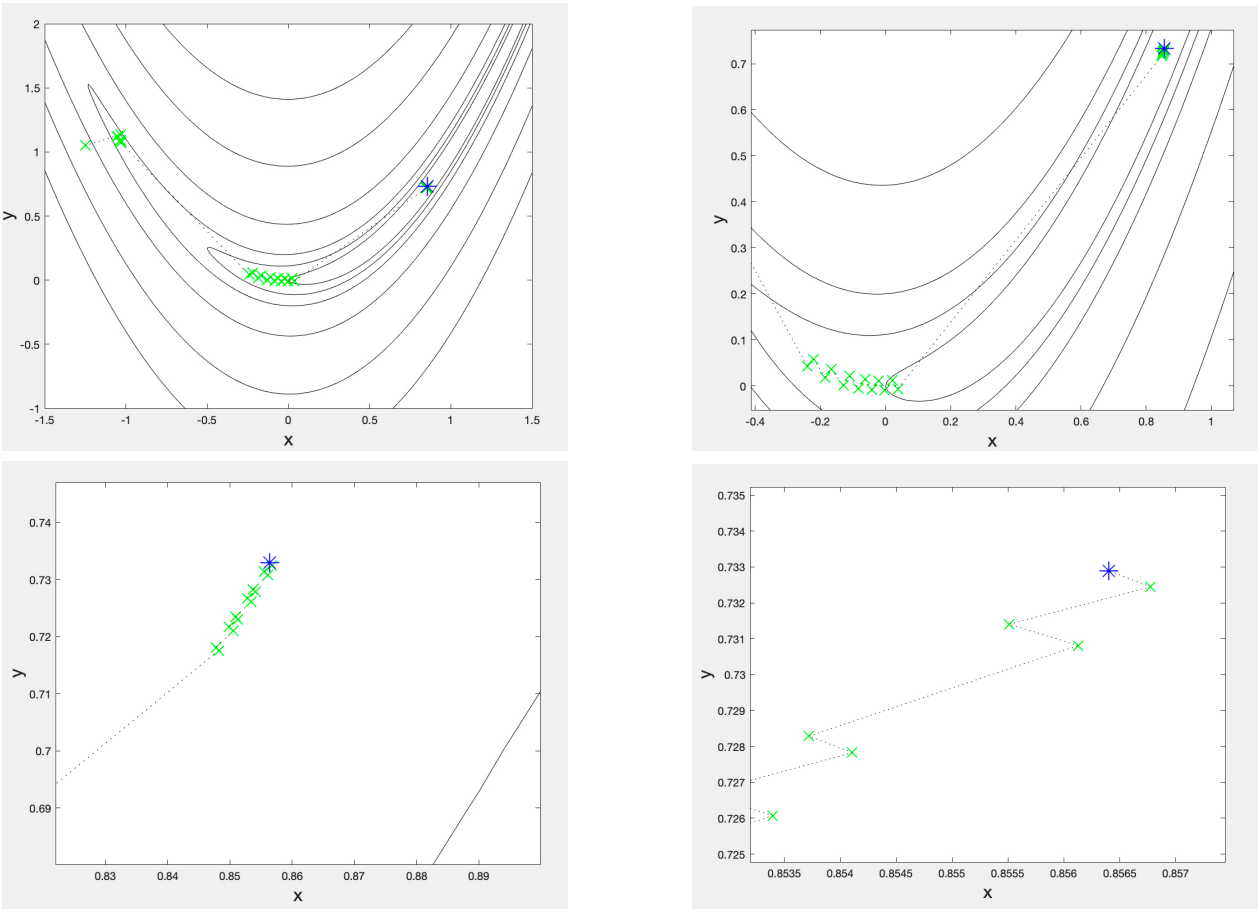


Figure 4: Gradient search, increasing zoom, contour levels plots for Banana function.

It is not that the successive steps are not orthogonal to each other, but that the banana function has a curved flat valley in the region where the optima is located; thus, the gradient search method moves through contour plot in really small step,s and takes a lot of steps to make progress. This is because the function decays sluggishly in the valley and forces the gradient to take tiny steps. The zoomed figures show how the method goes from step to step connecting the green dots and how slowly it moves at the end.

3. Compare with the experiment of using Newton's Method. State your observations and comment on the advantages and drawbacks of both methods.

The following table shows the result of doing the experiment.

	Xintial	Xopt	Zopt	Eps	Iterations	norm_g
Gradient	(-1.25, 1.05)	(0.8564, 0.7329)	0.0206	0.15	35	0.1491
Newton	(-1.25, 1.05)	(1, 1)	3.31E-12	0.15	6	3.64E-06

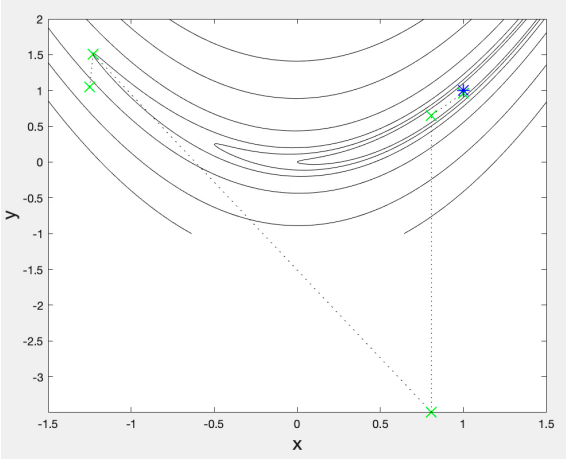


Figure 5: Newton method for Banana function.

Newton´s method was more effective as it was closer the optima and had a smaller objective function value; nevertheless, this was possible because the Hessian is known and behaves properly within the method. Gradient search took more steps but does not depends on computing the Hessian, thus, avoids risk of catastrophic differences or a Hessian that blows up the algorithm. This is evidenced when the method exists the region as seen in the plot and distances itself too much from the optima. Gradient search does not have this problem.

7. And Finally...

The function to minimize is:

$$f(\underline{x}) = \frac{1}{2}x^T Ax - d^T x + d^T \underline{1}$$

Given our registration numbers 1900962, 1900898. $a = 8$, $b = 2$, $c = 1$.

$$A = \begin{pmatrix} a+2 & c+1 \\ c+1 & b+2 \end{pmatrix}$$

$$a > b > c > 0$$

$$d = (a + c + 3, b + c + 3)^T$$

$$\underline{1} = (1, 1)^T$$

The conditions on a, b, c guarantee that A is, by deign, positive definite; thus, f is a convex function and the following happens.

$$\nabla f(x) = Ax - d$$

$$H_f = A$$

$$\tilde{x} = A^{-1}d$$

$$\nabla f(\tilde{x}) = \underline{0}$$

The point $\tilde{x}$ is the optima as long as A is not singular, but it is positive definite so no worries.

For Gradient search (Steepest descent) it can be shown that the given function guarantees convergence to the optima.

$$x_{k+1} = x_k - \frac{\nabla f_k^T \nabla f_k}{\nabla f_k^T A \nabla f_k} \nabla f_k$$

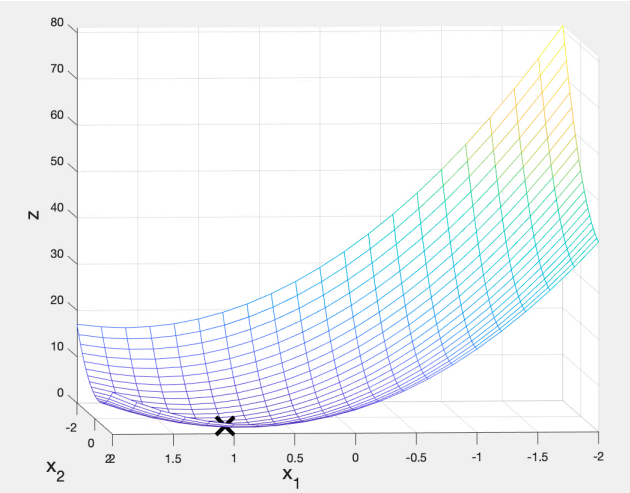


Figure 6: Plot of teh function to minimize.

For newton´s method:

$$\begin{aligned} x_{k+1} &= x_k - H_{f_k}^{-1} \nabla f_k \\ &= x_k - A^{-1}(Ax_k - d) \\ &= x_k - x_k + A^{-1}d \\ &= \tilde{x} \end{aligned}$$

1. Use the method of Gradient Search to minimize the function f(x), starting from three different initial guesses and with an appropriate choice of epsilon.

The following table shows the result of using gradient search.

Xintial	Xopt	Zopt	Eps	Iterations	norm_g
(0,0)	(1.0072, 0.9806)	7.33E-04	0.1	4	0.0715
(-1,-1)	(0.9962, 0.9956)	1.47E-04	0.1	5	0.0536
(1,0)	(0.9999, 0.9815)	6.91E-04	0.1	7	0.0836

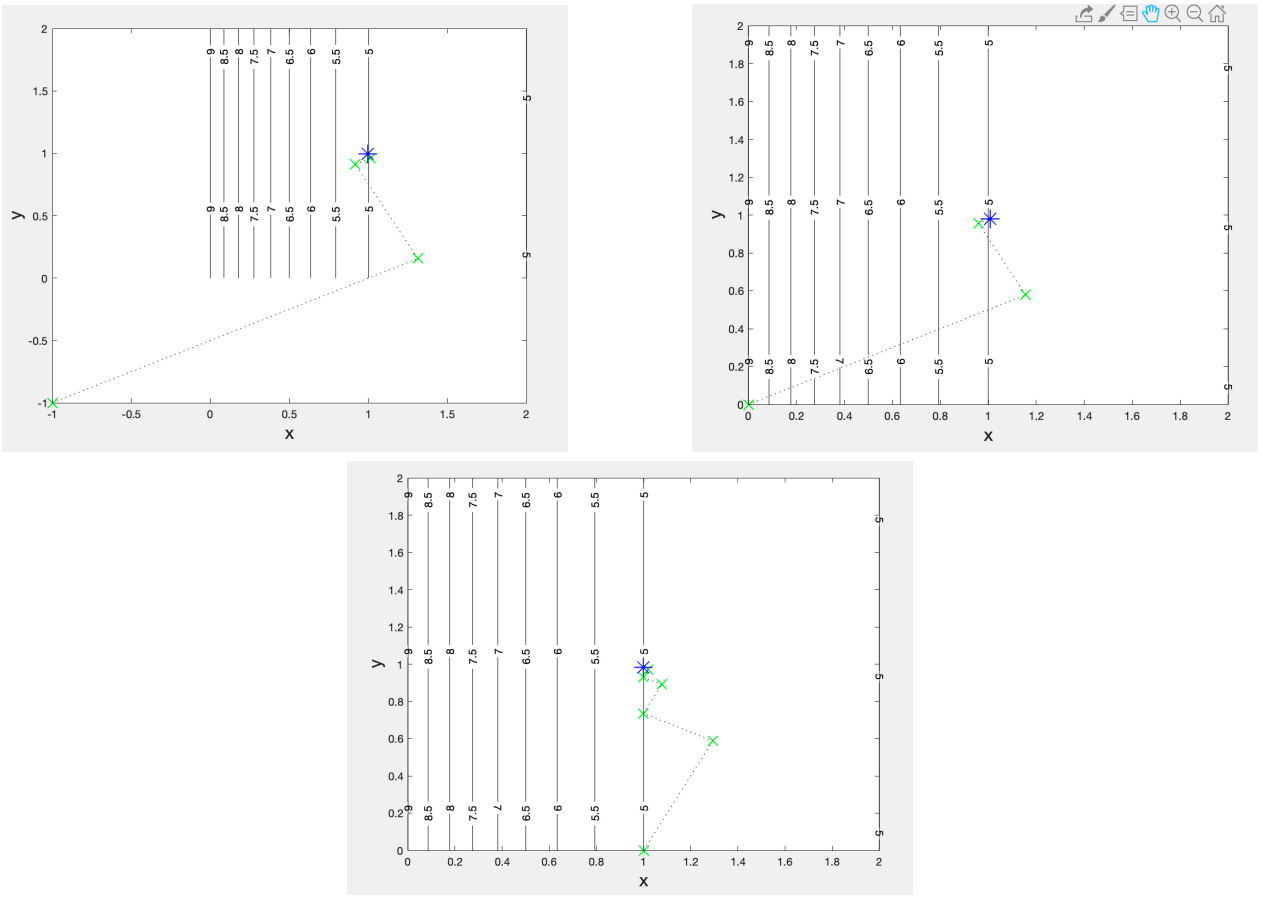


Figure 7: Gradient search contour levels plots.

Note that it solves the step size on each iteration using numerical procedures. Nevertheless the function is such that.

$$\tilde{x} = \underline{1}$$

Also,

$$f(x) = \frac{1}{2}x^T(Ax - d) + \frac{1}{2}d^T(\underline{1} - x)$$

Thus,

$$\begin{aligned} f(\underline{x}) &= \frac{1}{2}\tilde{x}^T(A\tilde{x} - d) + \frac{1}{2}d^T(\underline{1} - \underline{1}) \\ &= 0 \end{aligned}$$

2. Use Newton's Method to minimize the function f(x), starting from the same three different initial guesses and with the same choice of epsilon. Compare the results given by the two methods. Note that for any feasible choice of a, b, c it can be shown that for this given function,

The problem would be regarding catastrophic cancellation during iterations as the method will converge after 1 step for any initial guess that is not the optima. We see the blue dot is the optima, the green dot is the initial guess and there are no step dots.

Xintial	Xopt	Zopt	Eps	Iterations	norm_g
(0,0)	(1,1)	0	0.1	2	1.78E-15
(-1,-1)	(1,1)	-1.78E-15	0.1	2	3.66E-15
(1,0)	(1,1)	0	0.1	2	0

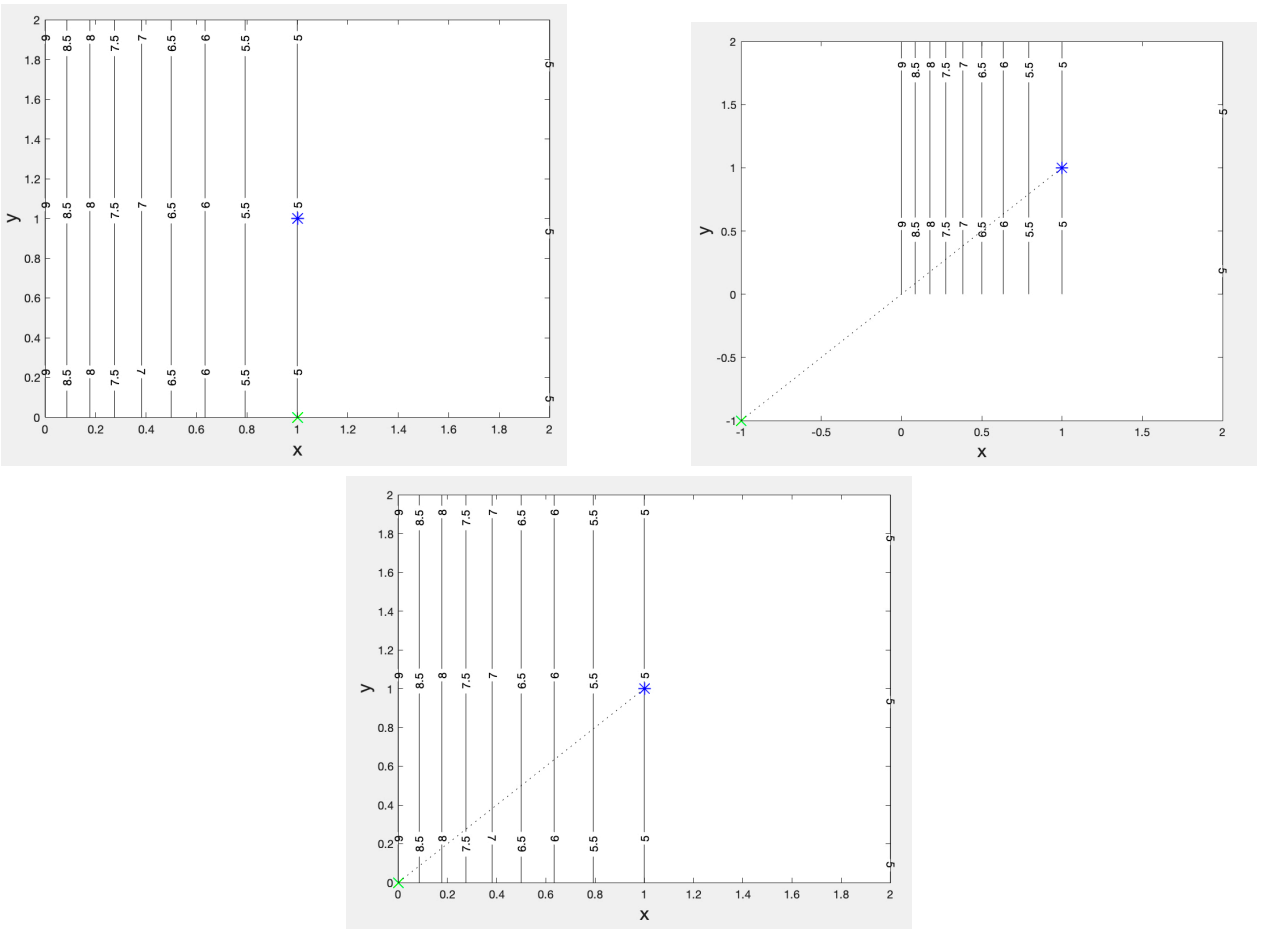


Figure 8: Newton´s method and contour levels

As the figures and table show, Newton´s method was more effective and efficient; nevertheless, both methods were able to converge rather quickly to the proper optima. The main advantage of Newton´s method is that convergence is quadratic by the form of using second order terms in the approximation, but its main drawback is that the Hessian might not be positive definite and the method will no longer have a descent direction or no way of solving for the next iteration or become inefficient calculating the inverse of the Hessian or solving the linear system.

This motivated techniques for approximating the Hessian and making it positive definite by modifying it´s eigenvalues. The most popular is Quasi-newton methods were the Hessian is approximated without second order derivatives which avoid indefiniteness in the Hessian and have super linear convergence. The gradient search method is effective but as the Rosenbrock function depicts has the drawback that it can excruciantingly slow in hard problems and might not even converge. For these reasons methods like gradient conjugate search have been developed.

To conclude, Newton´s method is very powerful but depends on the objective function having nice properties (i.e. a positive definite Hessian), while gradient search will eventually find the optima but might take too long or zig-zag in a crazy way in weird regions. Both methods share depend on the objective function being differentiable or at least being continuous and well behaved.

8. Appendix: Some of the MATLAB Code that we used

```
% ================================================
% LAB 2 NONLINEAR PROGRAMMING
% ================================================

%% ================= Question 1

f = 'func_quartic';
x = [1;2];
[zcalc, gcalc, Hcalc] = func_quartic(x);

% To check if they are correctly calculated, we can simply
   substitute in
% the analytic formula for the Hessian and the gradient.

% Define the formulas:
z = (x-2).^4 + (x-2*y).^2;
g = @(x) [4*(x-2).^3 + 2*(x-2*y) ;
    -4*(x-2*y)];
H = @(x) [12*(x-2).^2, -4;
          -4    ,       8];
% Evaluate and check


%% ================= Question 2

```

```
26
27
28
29
30 %% ================= Question 3
31 f = 'func_rosenbrock'
32 xvector = [1;1]
33 lims = [-2 2 -2 2]
34 plot_surface_taylor(f,xvector,lims)
35
36 f = 'func_beale'
37 xvector = [1;1]
38 lims = [-2 2 -2 2]
39 plot_surface_taylor(f,xvector,lims)
40
41 f = 'func_quad'
42 xvector = [1;1]
43 lims = [-2 2 -2 2]
44 plot_surface_taylor(f,xvector,lims)
45
46
47
48
49 %% ================= Question 4
50
51
52
```

```
53
54 %% ================= Question 5
55 % Try this ...
56 f = 'func_quartic';
57 func_quartic_contour(1);
58 xvector = [1;2.5];
59 dvector = [3;-4];
60 [tlo,tmid,thi] = nlp_line_threepoint_min(f,0,0.08,xvector,dvector
      ,1);
61
62 %% Try this ...
63 f = 'func_quartic';
64 func_quartic_contour(1);
65 xvector = [1;2.5];
66 dvector = [3;-4];
67 [tlo,tmid,thi] = nlp_line_threepoint_min(f,0,0.08,xvector,dvector
      ,0);
68 epsilon = 0.01;
69 [topt,zopt] = nlp_line_golden_min(f,tlo,thi,epsilon,xvector,
      dvector,1);
70
71 %% Task 1
72 f = 'func_quartic';
73 func_quartic_contour(1);
74
75 % Try various points for the starting point, in different
      directions.
```

```
% We can try 4 different points towards 4 different directions, for a total
%       of 4x4 = 16 tries.

% Starting points
xvector = [1;2.5];
% xvector = [1;2.5];
% xvector = [1;2.5];
% xvector = [1;2.5];

% Directions
dvector = [3;-4];
% dvector = [3;-4];
% dvector = [3;-4];
% dvector = [3;-4];

[tlo,tmid,thi] = nlp_line_threepoint_min(f,0,0.08,xvector,dvector,0);
epsilon = 0.01;
[topt,zopt] = nlp_line_golden_min(f,tlo,thi,epsilon,xvector,dvector,1);
%% Task 2
% Same idea, but now with 4 other functions and 2 points and
%       2 directions each, for a total of 4x2x2 = 16 new tries
f = 'func_quartic';
% f = 'func_quartic';
% f = 'func_quartic';
```

```
% f = 'func_quartic';
func_quartic_contour(1);
% func_quartic_contour(1);
% func_quartic_contour(1);
% func_quartic_contour(1);

% Try various points for the starting point, in different
   directions.
% We can try 4 different points towards 4 different directions,
   for a total
%       of 4x4 = 16 tries.

% ==== Starting points
% Rosenbrock points
xvector = [1;2.5];
% xvector = [1;2.5];
% Beale points
xvector = [1;2.5];
% xvector = [1;2.5];
% Quadratic points
xvector = [1;2.5];
% xvector = [1;2.5];
% ASJKFNASKFA points
xvector = [1;2.5];
% xvector = [1;2.5];

% ==== Directions
```

```
125 % Rosenbrock directions
126 dvector = [1;2.5];
127 % dvector = [1;2.5];
128 % Beale directions
129 dvector = [1;2.5];
130 % dvector = [1;2.5];
131 % Quadratic directions
132 dvector = [1;2.5];
133 % dvector = [1;2.5];
134 % ASNDJKASNDJKA directions
135 dvector = [1;2.5];
136 % dvector = [1;2.5];
137
138 [tlo,tmid,thi] = nlp_line_threepoint_min(f,0,0.08,xvector,dvector
       ,0);
139 epsilon = 0.01;
140 [topt,zopt] = nlp_line_golden_min(f,tlo,thi,epsilon,xvector,
       dvector,1);
141 %% ================= Question 6
142
143
144
145
146
147 %% ================= Question 7
148 f = 'My_fucntion'
149 My_function_contour(1)
```

```
xinitial = [-1;-1];
epsilon = 0.1;
hold on
[xopt,zopt] = nlp_solve_gradient(f,xinitial,epsilon);

f = 'My_fucntion'
My_function_contour(1)
xinitial = [-1;-1];
epsilon = 0.1;
hold on
[xopt,zopt] = nlp_solve_purenewton(f,xinitial,epsilon)
```