MATLAB Tutorial and Week 7 Hand-in Exercise

1 Introduction to the optimization toolbox

Exercises to be posted to CANVAS are at the end of the assignment sheet. The first part in preparation for the assessment.

The parts of this tutorial that are flagged in the section header for submission are to be used for assessment purposes. The code used to generate the input for that exercise must be included in the submitted work along with the output and handed in to be marked. Please do this by publishing your work in Matlab as a pdf file and then upload onto CANVAS.

Any script or function code that *you have modified or written* must also be so included using the `type filename.m` command.

Add a pfd file or a photo of your handwriting for the questions that are not related to Matlab.

2 MatLab Laboratory Session Week 7

There are two ways of interacting with the toolbox. Via an interactive GUI or via a command line input. This typical of Matlab philosophy. As is the case for many Matlab toolboxes, one needs to understand the way to input basic parameters and code objective and constraint functions. Once one has done then there is a graphical interface for the Optimization Toolbox that enables one to run many algorithms with little further knowledge. On the other hand if one wants to use these tools in a larger program it is essential to know how to enter command line code to perform the same tasks.

1. In our exercises so far we have learnt to define our functions using a particular format. For instance for the Rosenbrock function `Nf3.m` . To test these codes for larger number of variable we will code the Rosenbrock's valley (or the De Jong's function 2). This is given by

$$f(x) = \sum_{i=1}^{n-1} 100\left(x_{i+1} - x_i^2\right)^2 + (1 - x_i)^2$$

   which is given by in Matlab (assuming `x` is a given vector of length n)

```
        k= [1 : n − 1]
   value  = sum(100*(x(i+1)-x(i).^2).^2+(1-x(i)).^2);
```

(a) For $n = 4$ the function looks like

$$\begin{aligned} f(x) = &\ 100\left(x_2 - x_1^2\right)^2 + (1 - x_1)^2 \\ &+\ 100\left(x_3 - x_2^2\right)^2 + (1 - x_2)^2 +\ 100\left(x_4 - x_3^2\right)^2 + (1 - x_3)^2 \end{aligned}$$

and has derivatives

$$\begin{aligned} \frac{\partial f}{\partial x_1} &= 200\left(x_2 - x_1^2\right)(-2x_1) + 2(x_1 - 1) \\ \frac{\partial f}{\partial x_2} &= 200\left(x_2 - x_1^2\right) +\ 200\left(x_3 - x_2^2\right)(-2x_2) + 2(x_2 - 1) \\ \frac{\partial f}{\partial x_3} &= 200\left(x_3 - x_2^2\right) + 200\left(x_4 - x_3^2\right)(-2x_3) + 2(x_3 - 1) \\ \frac{\partial f}{\partial x_4} &= 200\left(x_4 - x_3^2\right). \end{aligned}$$

Can you guess what the general case for an arbitrary n would be.

(b) From CANVAS week 7 module you may download the m-file to return the value of the De Jong's function 2 and its gradient depending on how many out put arguments are requested (using the switch construct based on `nargout` taking the values 1 or 2).

2. To run these routines from the command line we must first set the options for the solver. These can be set using the

   `options=optimset('Name',value,..)` (A.1)

   command. For instance true using:

```
>>options=optimset('LargeScale','off');
>>x = fminunc(@DeJong2,3*ones(10,1),options);
>> x
```

   The last command simply discplays the value of x which should contain ones. One can slect to input gradients and hessians by using the following options:

```
options=optimset('GradObj','on','Hessian','on');
```

   Of course your function must out put these (our `DeJon2.m` only outputs the gradient at this stage which `Nf3.m` can out put the Hessian as well).

3. For as given solver one can find a full list of options (and their current values) that can be varied by typing using (A.1) by typing the following in the command window:

```
>>optimset fminunc
```

Also

```
>>help fminunc
```

will list the syntax and usage of the command line version of this optimization routine.

Let us now consider the case of constraint optimization. Constraints need to be coded into m-files just like the objective functions. There are two classes of constraints used, inequality and equality i.e.

$$\begin{aligned} g(x) &\le 0 \qquad \text{inequality} \\ h(x) &= 0 \qquad \text{equality.} \end{aligned}$$

Suppose we wish to impose the following constraint:

$$\begin{aligned} g_1(x_1,x_2) &= x_1^2+x_2^2-1 \le 0 \qquad \text{inside the unit circle} \\ g_2(x_1,x_2) &= (x_1-1)^2+x_2-2 \le 0 \qquad \text{below the parabola} \end{aligned}$$

In this case we have to return a vector on objective values $[g_1(x), g_2(x)]$ and a transposed Jacobian for the gradients

$$J^T = (\nabla g_1, \nabla g_2) = \left(\begin{pmatrix} \frac{\partial g_1}{\partial x_1} \\ \frac{\partial g_1}{\partial x_2} \end{pmatrix}, \begin{pmatrix} \frac{\partial g_2}{\partial x_1} \\ \frac{\partial g_2}{\partial x_2} \end{pmatrix} \right) = \begin{pmatrix} \frac{\partial g_1}{\partial x_1} & \frac{\partial g_2}{\partial x_1} \\ \frac{\partial g_1}{\partial x_2} & \frac{\partial g_2}{\partial x_2} \end{pmatrix}$$

and we have no equality constraint. We can provide this information in a function file as follows:

1.
   ```
   function [c ceq gc gceq] = unitdiskb(x)
   switch nargout
    case 1
        c(1) = x(1)^2 + x(2)^2 - 1;
       c(2) = (x(1) -2).^2 + x(2) -2;
    case 2
        c(1) = x(1)^2 + x(2)^2 - 1;
       c(2) = (x(1) -2).^2 + x(2) -2;
        ceq = [ ];
    case 3
        c(1) = x(1)^2 + x(2)^2 - 1;
       c(2) = (x(1) -2).^2 + x(2) -2;
        ceq = [ ];
   ```

```
     gc(1:2,1) = [2*x(1);2*x(2)];
    gc(1:2,2) = [2*(x(1)-2); 1];
 case 4
     c(1) = x(1)^2 + x(2)^2 - 1;
    c(2) = (x(1) -2).^2 + x(2) -2
     ceq = [ ];
     gc(1:2,1) = [2*x(1);2*x(2)];
    gc(1:2,2) = [2*(x(1)-2); 1];
     gceq = [];
end
```

Notice we have supplied and empty place holder for the equality constraint function. An alternative is to use (which demands at least 2 outputs at every call)

```
function [c, ceq, gc, gceq] = unitdiskb(x)
    c(1) = x(1)^2 + x(2)^2 - 1;
    c(2) = (x(1) -2).^2 + x(2) -2;
    ceq = [ ];
if nargout > 2
        gc(1:2,1) = [2*x(1);2*x(2)];
        gc(1:2,2) = [2*(x(1)-2); 1];
        gceq = [];
end
```

2. We can solve the problem of minimizing the Rosenbrock function with $w = 100$ (edit this value in the file `Nf3.m`) subject to the unit disc constraint either using a command line input as follows. Try the following which invoke a version of an active set strategy method and plots the value on the objective as we solve the problem:

```
>> options = optimset('Algorithm','active-set',...
        'GradObj','on','GradConstr','on','PlotFcns',{@optimplotx,...
                @optimplotfval});
>> x = fmincon(@Nf3,[0 0],[],[],[],[],[],[],...
                        @unitdiskb,options)
```

Note the use of many empty place holders.

Note also that $(1, 1)$ is not feasible (not inside the unit circle) so the solution must be on the boundary of the circle. Check that this is true.

3. You can find out the syntax of this command using

   ```
   >> help fmincon
   ```

   This procedure solves the following problem

   $$\begin{gathered} \min \quad F(x) \\ \text{Subject to} \\ Ax \leq b, \quad A_{eq}x = b_{eq} \\ LB \leq x \leq UB \\ C(x) \leq 0, \quad C_{equ}(x) = 0. \end{gathered}$$

   using the command structure

   ```
   [x,Fval]=FMINCON(@F,x0,A,b,Aeq,beq,LB,UB,@NONLCON)
   ```

   where `@NONLCON` is structured as in the function file `unitdiskb`.

4. The objective of this exercise is to design an aquarium. The aquarium is a cuboid defined by its length, width and height. There is no top and the bottom and the sides are all built in the same glass. Our objective is to maximise the volume for a fixed budget that is proportional to the surface of the bottom and the sides. We denote the variable as a 3-dimensional vector x, where $x(1)$ is the length, $x(2)$ is the width and $x(3)$ is the height. The problem to consider (as a minimization problem) is then:

   $$\begin{aligned} P_1 : \quad & \min \quad -x(1) \times x(2) \times x(3) \\ & \text{Subject to} \\ & x(1) \times x(2) + 2x(3) \times (x(1) + x(2)) = b = 1 \\ & x \geq 0 \end{aligned}$$

   where b represents the budget, seen as a fixed surface of glass.

   (a) Write a function `min_volume` with the same structure as the objective functions seen until now. The output is either the value of the function or its value and its gradient (we do not need the Hessian part). The input is a 3-dimensional vector x.

   (b) Write a function `sides` with the same structure as the constraint functions (like `unitdiskb` but in this case only assigns one gradient as output). The output has 1 to 4 components (see the structure of `unitdiskb`) and the input is a 3-dimensional vector x.

   (c) Define the options as previously except that we chose `'SQP'` as the algorithm. So, `'GradConstr'` and `'GradObj'` are `'on'` and `'PlotFcns'` is `{@optimplotx @optimplotfval}`. Try to solve the problem starting from $(0.5, 0., 0.5)$. i.e.

   ```
   >> options = optimset('Algorithm','SQP',...
   ```

```
        'GradObj','on','GradConstr','on','PlotFcns',{@optimplotx,...
                 @optimplotfval});
>> [x , fval] = fmincon(@min_volume,[0.5, 0.5, 0.5],[],[],[],[],...
                           zeros(3,1),[],@sides,options)
```

2.1 Example: Circle Packing Problem (to be handed in)

1. In this exercise we want to find out the centers and radi of he largest three circles we can inscribe inside a rectangle of length 5 units and heigth 3 units. See for example the figure below:

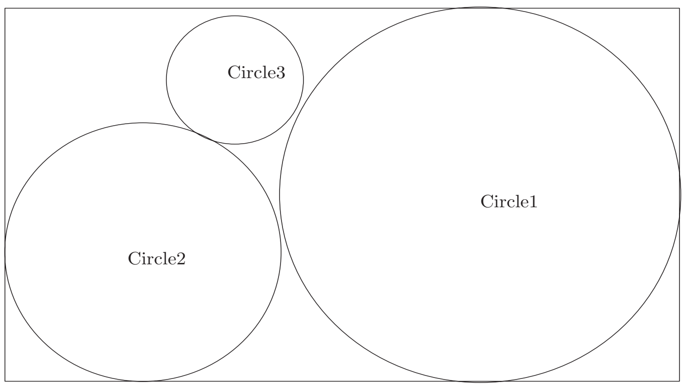

Figure 1: Three circles inscribed inside a rectangle.

   (a) Denoting the centers of circle i by (x_i, y_i) and its radius by r_i for $i = 1, 2, 3$. Then we require, to avoid overlap of the circles 1 and 2 (say), that we have
   $$d\left((x_1, y_1), (x_2, y_2)\right) \geq r_1 + r_2.$$
   This may be written as an inequality constraint (after squaring) to get
   $$g(r, x, y) = (r_1 + r_2)^2 - (x_1 - x_2)^2 - (y_1 - y_2)^2 \leq 0.$$
   There are three such inequalities, derive these and then write a MATLAB functions file to return the values and the gradients of the three functions. Call the file `Three_circles.m` and use a function call with following structure `[val, valeq, gval, gvaleq] = Three_circles(rx)` where the input `rx` is a vector of 9 components containing
   $$rx = [r(1), r(2), r(3), x(1), x(2), x(3), y(1), y(2), y(3)].$$

   (b) There are also 12 linear inequality constraints one needs to inlcude to avoid the radii from violatng the bounds of the rectangle. For instance if we tale as the origin the bottom left hand corner of the rectangle then for circle one we need

$$0 \leq x_1 - r_1,\ x_1 + r_1 \leq 5,\ 0 \leq y_1 - r_1 \text{ and } y_1 + r_1 \leq 3.$$

Write these down and construct a 12×9 matrix A and a 12×1 right hand side vector b to describe these via $A * rx \leq b$.

Hint: It is useful to concatinate identity matricies together i.e. `A1 = [[eye(3); eye(3)], eye(6)];` and `A2 = [[eye(3);eye(3)], -eye(6)];`

(c) In order to maximise the areas of the circles we need write an objective function, and since our solvers minimise it will deliver the negative the sums of the areas of the there circles and the gradient of this function of `rx`. Call the m-files `@Neg_Areas.m`

(d) Solve for the optimal centers and matrices that maximise the volumes of the three circles using the following options:

```
 options = optimoptions('fmincon','Display','iter','GradObj','on',...
              'GradConstr','on','PlotFcns',{@optimplotx,...
                                             @optimplotfval});
[rx, fval] = fmincon(@Neg_Areas, zeros(9,1), A, b,...
         [], [], zeros(9),[inf; inf; inf; 5; 5; 5; 3; 3; 3],...
                                     @Three_circles, options);
```

(e) What is the maximal area. Check that the answer look feasible (compare it to the area of the rectangle and compare the radii lengths etc).