Handout 5 – Optimization using MATLAB

MATLAB has several (more than 10) built-in functions for optimization. There is also a separate Optimization Toolbox and an `optimtool` function that opens a GUI application that has all the functionalities available in this toolbox. In this handout, we will study 5 of MATLAB's optimization functions.

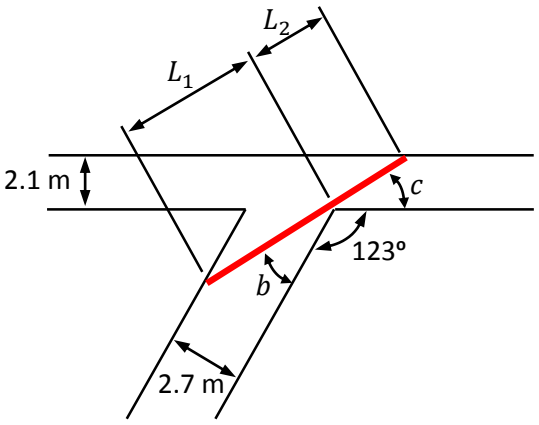
fminbnd

Performs 1D, unconstrained minimization in a given interval by using Brent's hybrid method that combines golden section search and quadratic (parabolic) interpolation. If you want to find the maximum (not minimum) of a function, you can use `fminbnd` by changing the sign of the function, i.e. use it with $-f(x)$.

Let's use it to solve the "ladder in a mine" problem, which is Q2 of Study Set 1. In order to find the longest ladder that can turn at the intersection of two shafts, we need to minimize the following function

$$L = L_1 + L_2 = \frac{2.7}{\sin\left(\pi - 123\frac{\pi}{180} - c\right)} + \frac{2.1}{\sin(c)}$$

where c is an angle that needs to be less than $180 - 123 = 57$ degrees (~ 1 rad), due to the geometry of the shafts. The problem can be solved in MATLAB as follows.




```
func = @(x) 2.7/sin(pi-123*pi/180-x) + 2.1/sin(x);

fminbnd(func, 0, 1)
```

```
% Objective function. x is the
% design parameter, i.e. the angle c

% Search in the interval [0, 1]
```

The result is 0.4677 radians (26.8 degrees), which is the c value that minimizes the objective function. We can also call `fminbnd` using two output arguments as follows

```
[x fval] = fminbnd(func, 0, 1)
```

```
% Two output arguments
```

which will give both the value of the design parameter c, and the minimum value of the objective function, which is 10.0252.

We can get wrong results if we use an improper interval.

```
[x fval] = fminbnd(func, 0, 2)
```

```
% This will give x = 0.9949 and fval = -8892. [0,2] is
% not a meaningful interval for this problem.
```

Below, we provide an interval that does not contain the correct minimum. In this case the minimum is at the right boundary of the interval, i.e. at 0.4, and that is what the function returns. But this is not the correct solution of the problem.

```
[x fval] = fminbnd(func, 0, 0.4)
```

```
% This will give x = 0.4, fval = 10.2111. The minimum
% is detected at the right end of the interval.
% Although correct for the specified interval, this is
% not the correct answer of the problem.
```

We can use the `optimset` command to activate different options and to change the default behavior of `fminbnd`.

```
options = optimset('Display', 'iter');          % Display option is activated
[c fval] = fminbnd(func, 0, 1, options)         % options parameter is used
```

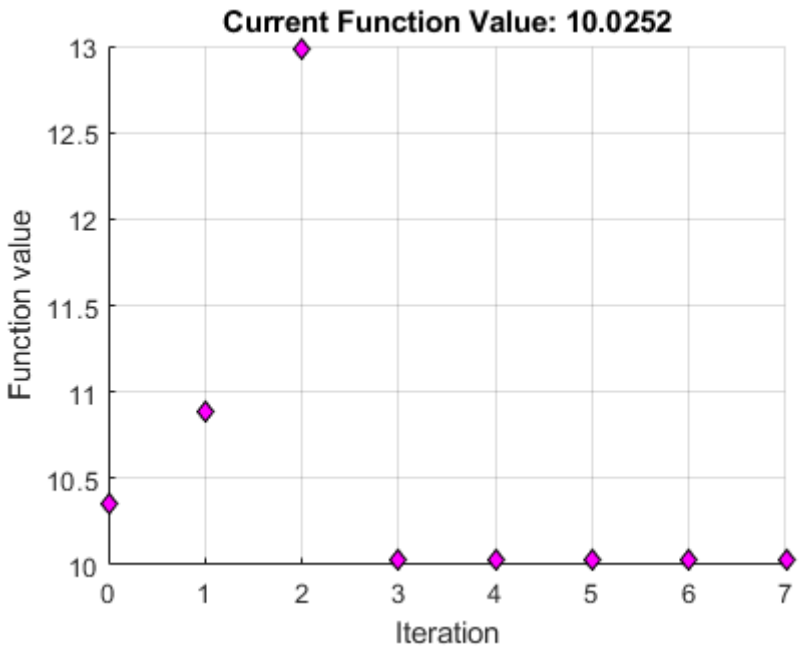
Following details will be displayed. 5 quadratic interpolation iterations (seen as “parabolic”) are used after two golden section iterations.

```
 Func-count     x          f(x)         Procedure
     1       0.381966      10.3277       initial
     2       0.618034      10.9622       golden
     3       0.236068      12.9032       golden
     4       0.474765      10.0271       parabolic
     5       0.467519      10.0252       parabolic
     6       0.467884      10.0252       parabolic
     7       0.467723      10.0252       parabolic
     8       0.467690      10.0252       parabolic
     9       0.467756      10.0252       parabolic
```

We can use the `'PlotFcns'` option to see how the function value changes with iterations.

```
options = optimset('PlotFcns', @optimplotfval);      % PlotFcns option is activated
[x fval] = fminbnd(func, 0, 1, options)
```

This will generate the following plot. The function value actually increases in the first 2 iterations, but then drops and settles down to its minimum around 10.



fminsearch

Performs multi-dimensional, nonlinear, unconstrained minimization without using the derivatives of the function. It uses a direct method known as the Nelder-Mead simplex algorithm, which we've not studied in our lectures, but you will study it in one of the study set questions. You can find two examples of how the iterations of this method progress in the following links.

https://www.youtube.com/watch?v=KEGSLQ6TlBM & https://www.youtube.com/watch?v=j2gcuRVbwR0

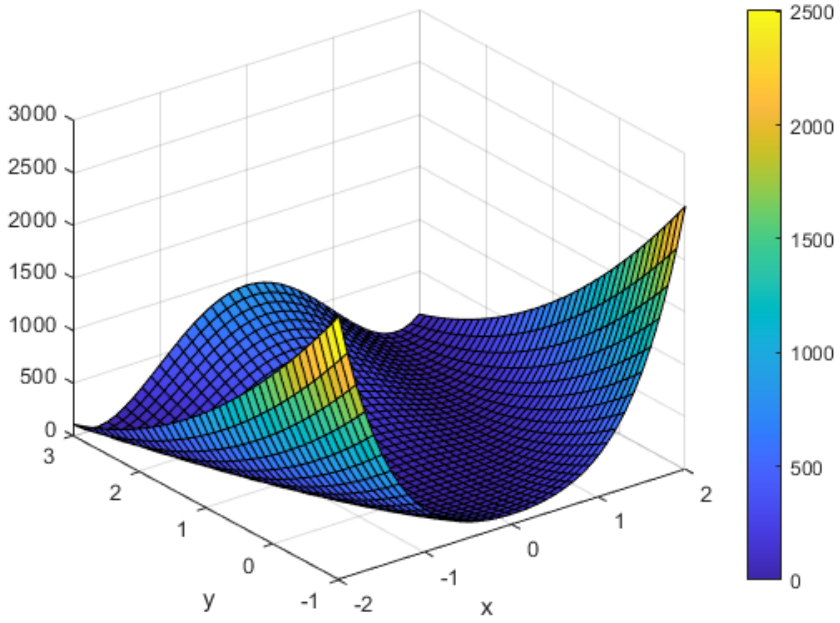
To find the maximum (not minimum) of a function, you can use `fminsearch` by changing the sign of the function, i.e. use it with $-f(x)$.

Let's use it to find the minimum of the following function, known as the Rosenbrock's "banana function". It is a popular optimization test function because of the slow convergence most methods exhibit in minimizing it.

$$f(x,y) = (1-x)^2 + 100(y-x^2)^2$$

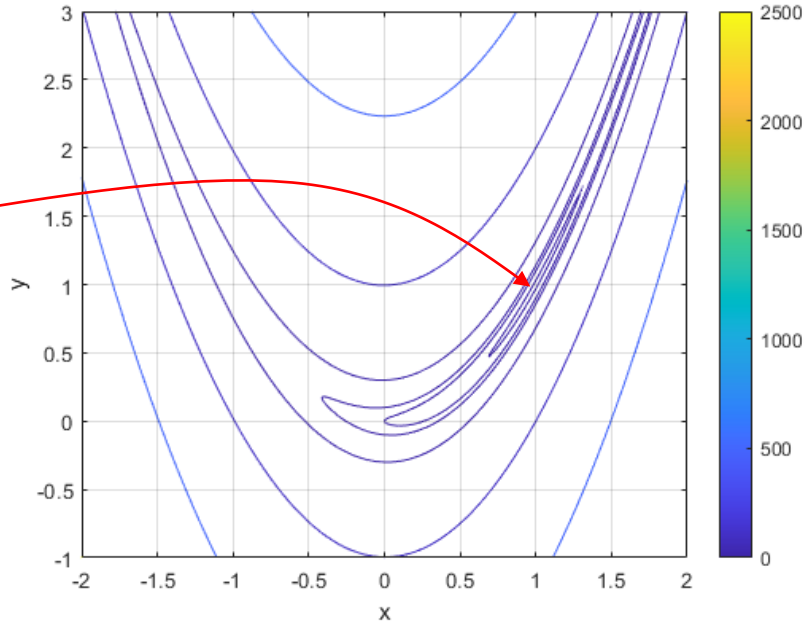
Here are two different plots of this function obtained by MATLAB's `surf` and `contour` commands.

```
xlabel('x'); ylabel('y')
[X,Y] = meshgrid(-2:0.1:2, -1:0.1:3);
surf(X,Y,Z)
Z = (1 - X).^2 + 100*(Y - X.^2).^2;
surf(X,Y,Z)
```




```
[X,Y] = meshgrid(-2:0.01:2, -1:0.01:3);
Z = (1 - X).^2 + 100*(Y - X.^2).^2;
contour(X,Y,Z, [0 0.1 1 2 10 100 500 2500])
```

Minimum value is zero, located at (1,1). It is inside a narrow valley, which is hard to reach. This makes the problem challenging.

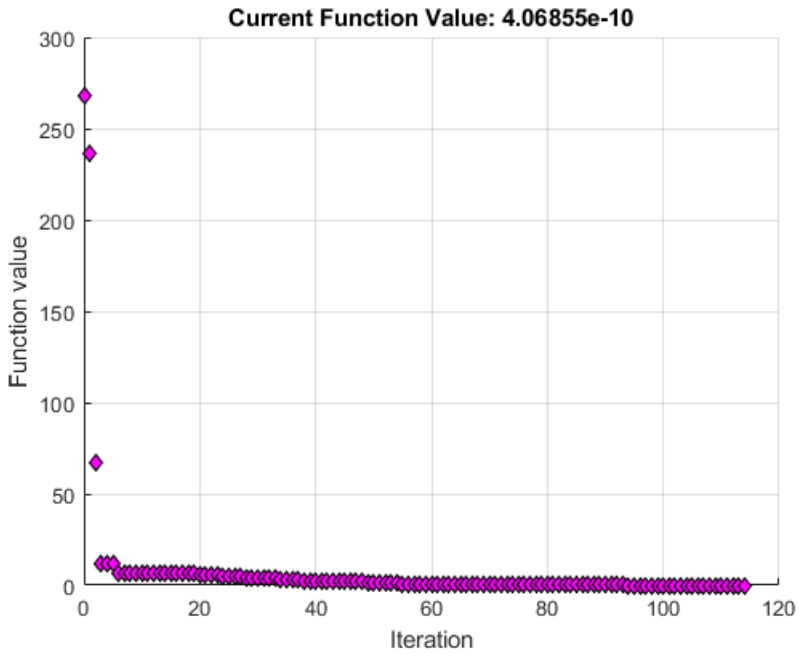


The minimum value and its location can be found as follows. The initial guess used here is (-1.9, 2). The objective function is defined not in terms of x and y, but x_1 and x_2, which are the two entries of the array $\{x\}$. Therefore, they are seen in the code as `x(1)` and `x(2)`. The initial guess is also given as an array.

```
fun = @(x) (1 - x(1))^2 + 100*(x(2) - x(1)^2)^2;    % Function of x(1) and x(2)
x0 = [-1.9, 2];                                     % Initial guesses
[x fval] = fminsearch(fun, x0)
```

The result is $x_{min} = 1.000$, $y_{min} = 1.000$, and $f_{min} = 4.0686 \times 10^{-10}$.

Here is the variation of the optimum value during the iterations. There is a sharp drop in the first 5 iterations, but then the convergence is rather slow, due to the challenging nature of this test problem.



fminunc

Performs multi-dimensional, nonlinear, unconstrained minimization by using the derivatives of the function, i.e. it makes use of gradient based methods. Two alternatives exist; Quasi-Newton (default option) and Trust-Region. To find the maximum (not minimum) of a function, you can use `fminunc` by changing the sign of the function, i.e. use it with $-f(x)$.

The same "banana function" problem can be solved as follows.

```
fun = @(x) (1 - x(1))^2 + 100*(x(2) - x(1)^2)^2;
x0 = [-1.9,2];
[x fval flag output] = fminunc(fun,x0)          % The function is called here with 4
                                                % output arguments
```

`fminunc` is called with 4 output arguments. The last argument, called output, is a structure shown below. It tells us that the default Quasi-Newton method is used, convergence took 34 iterations and the function is evaluated 150 times.

```
output =

  struct with fields:

      iterations: 34
       funcCount: 150
        stepsize: 0.003378819707224
    lssteplength: 1
   firstorderopt: 3.849663029020789e-04
       algorithm: 'quasi-newton'
         message: 'Local minimum found. Optimization completed ... '
```

It is possible to perform a Steepest Descent search by setting the `'HessUpdate'` option to `'steepdesc'`. This option controls how the search direction is determined in the quasi-Newton method.

```
options = optimset('HessUpdate', 'steepdesc');
[x fval flag output] = fminunc(fun,x0)
```

```
% To use steepest descent
```

But the result is not successful due to a premature termination. It happens because the default maximum number of function evaluations, which is 200, is reached before the minimum is detected. This shows the inefficiency of the steepest descent algorithm compared to the default one.

We can increase the number of function evaluations limit using the MaxFunEvals option as follows.

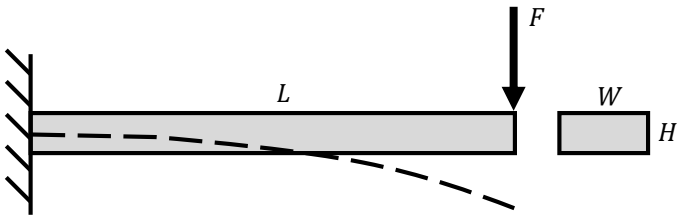
```
options = optimset('HessUpdate','steepdesc', 'MaxFunEvals', 500);
[x fval flag output] = fminunc(fun,x0)
```

But even 500 function evaluations is not enough for convergence with the default tolerance. According to the result of this last run, the minimum is located at (-1.1073, 1.2165), which is not correct.

fmincon

Performs multi-dimensional, nonlinear, constrained minimization by using a number of different algorithms. To find the maximum (not minimum) of a function, you can use `fmincon` by changing the sign of the function, i.e. use it with $-f(x)$.

Let's use this to solve the cantilever beam problem that we've discussed at the beginning of Chapter 4. It is about minimizing the weight of a beam. Design parameters are the width, W, and height, H. There are two constraints; one for the deflection of the tip and the other for the bending stress at the base. The necessary equations are



Objective function: $f(W,H) = \rho LWH$

Deflection constraint: $\frac{4FL^3}{EWH^3} \leq 0.025 \text{ m} \quad \rightarrow \quad \frac{4FL^3}{EWH^3} - 0.025 \leq 0$

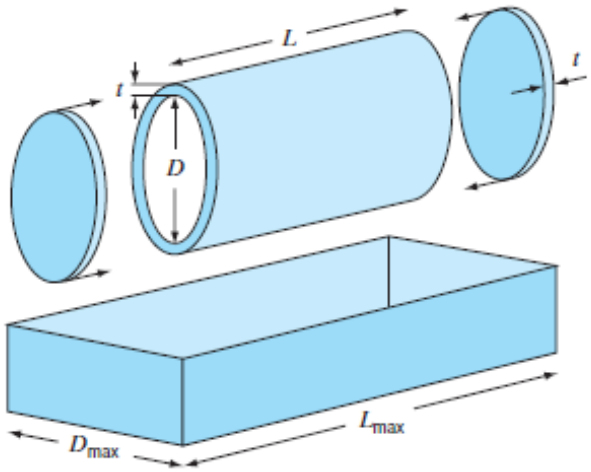
Stress constraint: $\frac{6FL}{WH^2} \leq 140 \times 10^6 \text{ Pa} \quad \rightarrow \quad \frac{6FL}{WH^2} - 140 \times 10^6 \leq 0$

where $F = 4000$ N is the point load applied at the tip of the beam, $L = 1$ m is the length of the beam, $\rho = 7600 \text{ kg/m}^3$ is the density of the beam, and $E = 200 \times 10^9$ Pa is the elastic modulus of the beam.

Beam_for_min_weight.m code solves this problem using the `fmincon` function. Please download it from ODTUClass and study it.

The result is $W_{min} = 0.4894$ m , $H_{min} = 0.0187$ m , $\text{Weight}_{min} = 69.62$ kg

As a second example, let's solve the cylindrical tank design problem, which is taken from our textbook. See page 412 of the 8th edition of our textbook for details. Design parameters are the diameter, D, and length, L, of the cylindrical tank. Objective function is the sum of material and welding costs, which needs to be minimized. There are two simple linear inequality constraints, $L \leq L_{max}$ and $D \leq D_{max}$, which can also be seen as upper bounds. There is also a third nonlinear equality constraint for the volume of the cylindrical tank. Equations that govern the problem are



Objective function: $f(L,D) = c_m m + c_w l_w$

Volume constraint: $\frac{\pi D^2 L}{4} = 0.8 \quad \rightarrow \quad \frac{\pi D^2 L}{4} - 0.8 = 0$

Upper bounds: $L \leq 2 \text{ m} \quad , \quad D \leq 1 \text{ m}$

where $c_m = 4.5$ \$/kg is the mass cost factor, $c_w = 20$ \$/m is the welding cost factor, m is the mass of the cylindrical tank given below

$$m = \rho\left\{L\pi\left[\left(\frac{D}{2}+t\right)^2-\left(\frac{D}{2}\right)^2\right]+2\pi t\left(\frac{D}{2}+t\right)^2\right\}$$

l_w is the weld length given below

$$l_w = 4\pi(D+t)$$

$\rho = 8000\ \text{kg/m}^3$ is the density of the tank material, and $t = 0.03\ \text{m}$ is the thickness of the tank wall.

Cylindrical_tank _for_min_cost.m code solves this problem using the `fmincon` function. Please download it from ODTUClass and study it.

The result is $L_{min} = 1.053\ \text{m}$, $D_{min} = 0.983\ \text{m}$, $\text{Cost}_{min} = 5723\ \$$

`linprog`

Performs multi-dimensional, linear, constrained minimization, which is known as linear programming. If you want to find the maximum (not minimum) of a function, you can use `fmincon` by changing the sign of the function, i.e. use it with $-f(x)$.

Let's solve the gas processing plant example defined at page 396 of the 8th ed. of our textbook. We want to determine how much two different grades of gases (regular and premium) should be produced such that the profit is maximized. There are material, time and storage constraints. Equations that we need are

Objective function:	$f(x_1,x_2) = 150x_1 + 175x_2$
Material constraint:	$7x_1 + 11x_2 \le 77$
Time constraint:	$10x_1 + 8x_2 \le 80$
Storage constraints:	$x_1 \le 9$ & $x_2 \le 6$
Positivity constraints:	$x_1 \ge 0$ & $x_2 \ge 0$

In linear programming, the inequality constraints are usually shown in the following matrix form

$$[A]\{x\} \le b \quad \rightarrow \quad \begin{bmatrix}7 & 11\\ 10 & 8\end{bmatrix}\begin{Bmatrix}x_2\\ x_2\end{Bmatrix} \le \begin{Bmatrix}77\\ 80\end{Bmatrix}$$

Gas_processing_plant.m code solves this problem using the `linprog` function. Please download it from ODTUClass and study it.

The result is $x_1 = 4.889$, $x_2 = 3.889$, $\text{Profit} = -1414$

Note that the profit is calculated as a negative value because we converted the original maximization problem into minimization by using the negative of the cost function. By default, all optimization functions of MATLAB perform minimization.