

# Session 10: Numerical Optimization

Fabrice Collard and Philipp Wegmueller

In today's session we want to learn how Matlab can be used to perform numerical optimization.<sup>1</sup> One of the most basic numerical problems encountered in computational economics is to find the solution of a system of (potentially nonlinear) equations. Optimization is an important tool in decision science and in the analysis of physical systems. To use it, we must first identify some *objective*, a quantitative measure of the performance of the system under study. This objective could be utility, profit, time, or any quantity or combination of quantities that can be represented by a single number. The objective depends on certain characteristics of the system, called *variables* or *unknowns*. Our goal is to find values of the variables that optimize the objective. Often the variables are restricted, or *constrained*, in some way. Once the model has been formulated, an optimization algorithm can be used to find its solution. Usually, the algorithm and model are complicated enough that a computer is needed to implement this process.

The aims of today are:

- a) Set up successfully a grid search problem.
- b) Apply minimization functions to perform numerical optimization.

---

<sup>1</sup>See Numerical optimization (1999) by Wright, SJ; or Numerical methods in economics (1998) by Judd, K.

## 1 Optimization

The most general optimization problem *minimizes* an objective function,  $f$ , subject to equality ( $g$ ) and inequality constraints ( $h$ ). We examine minimization problems, since maximizing  $f$  is the same as minimizing  $-f$ . Most optimization software is geared for minimization, so we need to get used to transforming maximization problems into minimization problems.

$$\begin{array}{ll} \min_x & f(x) \\ \text{s.t.} & g(x) = 0 \\ & h(x) \leq 0, \end{array}$$

There exists a number of optimization methods. All of them search through the space of feasible choices, generating a sequence of guesses that should converge to the true solution. Methods differ in the kind of information they use. The simplest methods compute the objective and constraint functions at several points and pick the feasible point yielding the largest value.

## 1.1 Grid search

Consider as a first example that you are given the following polynomial function:

$$f(x) = x^2 + 3x - 10,$$

which gives rise to the following parabola:

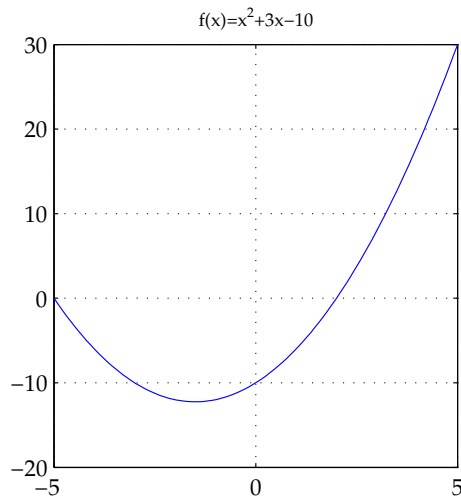


Figure 1: Finding the minimum of the function

Your task is to find **the minimum** of this function. The most primitive procedure to minimize a function is to specify a grid of points, say, between 100 and 1000 points, evaluate  $f(x)$  at these points, and pick the minimum. The grid search method is usually the first step in numerical optimization and serves as an indicator of the results.

In Matlab we can use the command `y = linspace(a,b,n)` to generate a linearly spaced vector of  $n$  points between  $a$  and  $b$ .

## 1.2 Numerical optimizers

In a second step you shall numerically find **the roots** of the function, that is you want to solve

$$\begin{array}{ll} \min_x & f(x) = x^2 + 3x - 10 \\ \text{s.t.} & f(x) = 0. \end{array}$$

All optimization problems can be stated in such a way that they have a zero on the left hand side of the equation. Then, the problem of solving the equation amounts to finding the values of  $x$  which yield  $f(x) = 0$ . This will be useful for steady state calculations (Session 12 of this course) or transformation of time series, etc. Matlab provides several solution algorithms to do this task.

**fzero** Can be used to solve a single equation numerically for a single variable. The solution can be found *iteratively* with the single statement:

```
[x_opt1, fval, exitflag]=fzero(@(x) (x.^2 +3.*x-10),-3);
```

Where -3 is the initial guess and the function returns the solution -5. The other solution is found using an initial guess of +3. The `@`-syntax is used to pass an anonymous function to the **fzero** function (it is a so-called **function handle**, which indicates, for which variable in the following equation it should be solved). In this case, it is the complete objective function, and the `(x)` preceding the function tells matlab that **x** is the variable to be adjusted. In this case, **x** is the only variable in the function. To constrain the search interval, a vector with bounds can be used for the second argument, such as `[3, 1]`. The response of

the objective function value is stored in `fval`, and the exit condition `exitflag` is 1 on a successful solution.

Alternatively you could write also

```
myfun = @(x) (x.^2 +3.*x-10)
[x_opt1, fval, exitflag]=fzero(myfun,-3);
```

**fsolve** (Only comes with the Optimization Toolbox – Check on your workstation) Should be used to solve systems of nonlinear equations. However, it does not allow you to include any constraints, even bound constraints. Suppose you have two equations:

$$0 = 2x - y - e^{-x}$$

$$0 = -x + 2y - e^{-y}$$

To solve this problem specify a function `myfun` with a handle, that reads like

```
myfun = @(x) [2*x(1)-x(2)-exp(-x(1));
-x(1)+2*x(2)-exp(-x(2))];
```

Then you make a starting guess, for instance `x0 = [-5 -5]` and use the function

```
[x_opt1, fval, exitflag]=fsolve(myfun,x0,[options])
```

where in `[options]` you could specify different options for the solution algorithm.

**fminbnd** Finds a minimum of single-variable function on a fixed interval, and should be used when there are *bounded constraints*:

$$\min_x f(x) \quad \text{s.t.} \quad x_1 < a < x_2$$

Then the syntax is as follows:

`[x_opt1, fval, exitflag]=fminbnd(myfun,x1,x2)` returns a value `x_opt1` that is a local minimizer of the scalar valued function that is described in `myfun` in the interval  $x_1 \leq x \leq x_2$ .

`[x_opt1,...]=fminbnd(myfun,x1,x2,options,P1,P2,...)` provides for additional arguments, `P1`, `P2`, which are passed to the objective function `myfun`. Use `options=[]` as a placeholder if no options are set.

**fcsolve** Is an extension of `fsolve`, as it is possible to add constraints to the optimization. The function reads similarly to the previous ones:

```
[x,rc] = fcsolve(myfun,x0,option,P1,P2,.....)
```

Note that generally these optimizers allow for different types of **inputs** for the function which shall be evaluated. For instance take the `fminbnd` function: Function arguments contains general descriptions of arguments passed in to `fminbnd`. Suppose `myfun` is the function to be minimized, it accepts a scalar `x` and returns a scalar `f`, that is the objective function  $f(x)$  evaluated at `x`. The function `myfun` can be specified in two ways:

**As handle:** `x = fminbnd(@myfun,x1,x2)`, where we have function `f = myfun(x)` and inside the function a line which says `f = ...`, where  $f(x)$  is evaluated at `x`.

**As object:** `x = fminbnd('myfun',x1,x2);`, where `'myfun'` is a string containing the name of the function. For example, `fminbnd('cos',0,4)` returns the value 3.14.

We can only touch upon the issues of numerical optimization and there are many intricacies involved. A good starting point is often the **Matlab help**, which provides good guidance on which algorithm/solver to use for the problem at hand. In general, functions differ whether your problem i) is nonlinear, ii) differentiable, iii) is univariate or multivariate, iv) involves solving or minimizing a function, and/or iv) features constraints.

### Exercise 1: Function optimization \*

1. Plot the following polynomial function in a range of  $x \in [-2.5; 2.5]$ .

$$f(x) = x^3 - x^2 - 3x.$$

2. Use the `fsolve` function to find the roots of this polynomial.
3. Find the local *maxima* and *minima* (if any) of this polynomial function (within the range specified) using the function the `fminbnd`. Use the roots to define appropriate bounds.
4. Consider the following function  $f(x)$ :

$$f(x) = \left( 2x - \frac{x^2}{2} - \frac{x^2}{0.02 + x^2} \right).$$

Use the `fzero` function to find the roots of  $f(x)$  Compare your results by plotting the function in the range  $0 \leq x \leq 5$ .

5. Consider the following equation system:

$$f(1) = x_1^2 + x_2^2 - 2$$

$$f(2) = x_1 x_2 - 1$$

Use the `fsolve` function to solve for  $x_1$  and  $x_2$ . Take as initial guess  $\mathbf{x}_0 = [10 \ 10]$ .

### Exercise 2: Utility Maximization \*\*

Consider (standard) the problem of a household who decides on her consumption plans so as to maximize her utility function subject to the budget constraint

$$\begin{aligned} \max_{c_1, c_2} c_1^\alpha c_2^{1-\alpha} \\ p_1 c_1 + p_2 c_2 = I \\ c_1, c_2 \geq 0 \end{aligned}$$

where  $I$  is the income of the household,  $p_1$  and  $p_2$  are the prices of each goods. In the first place, we set  $I = 10$ ,  $p_1 = 1$ ,  $p_2 = 2$  and  $\alpha = 0.5$ .

1. Set a grid of values for  $c_1$  (you will have to think about the relevant bounds for the grid) using 10'000 points in the grid, and evaluate the utility function at each point of the grid using Matlab. Find the maximal value of the utility function and the optimal consumption plan. (Hint: Use the `max` function to find the maximum value in a vector.)
2. Setup a function that evaluates the utility function, the same way you did it in the first question, and which passes as argument  $c_1$ , and the vector of parameters  $(I, p_1, p_2, \alpha)$ . Repeat Question 1 using this function.
3. Use the previous function and the function `fminbnd` to obtain the optimal consumption plan. (Hint: Be careful, you may have to modify a bit the function you obtain in Question 2, think of the minimization

problem.)

4. Write the first order conditions to the problem on a piece of paper. Then set up a Matlab file that evaluates these first order conditions. This function will have as arguments  $c_1$  and  $c_2$ , and as vector of parameters  $(I, p_1, p_2, \alpha)$ .  
Hint: Use the function `fcsolve` to solve this optimization problem. This function solves for  $(@)FUN(x,P1,P2,\dots)=0$ . Maybe you need to use a function handle  $(@)$ . Search the Matlab documentation on how to use a handle if you need more help.
5. Assume now that the income of the agents varies from 1 to 20 (take 20 values), use the approach of Question 4, to obtain the optimal consumption decisions for each level of income. Plot the consumption functions as a function of income.
6. Repeat the last question with  $p_1$ , for  $p_1$  between 0.5 and 1.5 (20 values)

### **Exercise 3: Utility Maximization - CES utility \*\*\***

Repeat Exercise 1 with the following utility function

$$u(c_1, c_2) = \left( \alpha c_1^{\frac{\rho-1}{\rho}} + (1-\alpha) c_2^{\frac{\rho-1}{\rho}} \right)^{\frac{\rho}{\rho-1}}$$

Also study the effect of variations in rho (between 0.5 and 1.5) on optimal consumption.