Introduction to MATLAB

Session 5: Introduction to Numerical Optimization and MATLAB Solvers

Maria Ptashkina Damian Romero

Barcelona Graduate School of Economics

September 2020

Introduction

We want to find the solution of a system of equations or the max/min of a function

- Find the roots or zeroes of a function: $f(x^*) = 0$
- Find the maximize/minimize a function: $f'(x^*) = 0$

Examples

- Find the equilibrium price of an economy
- Maximize a log-likelihood function

Numerical methods will help us to find a numerical solution for the problem at hand

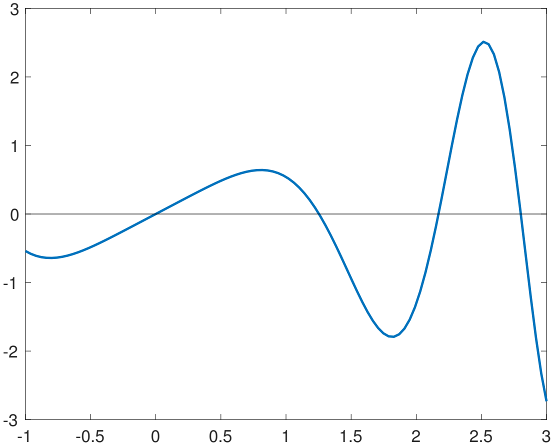
Key

- Because we are numerically solving a problem, we have to accept some *tolerance* for the solution
- We will rely on *iterative* algorithms: the starting point matters
- *Local* solutions

Example

Where are the zeroes? Where is the function maximized/minimized?

$$f(x) = x\cos(x^2)$$



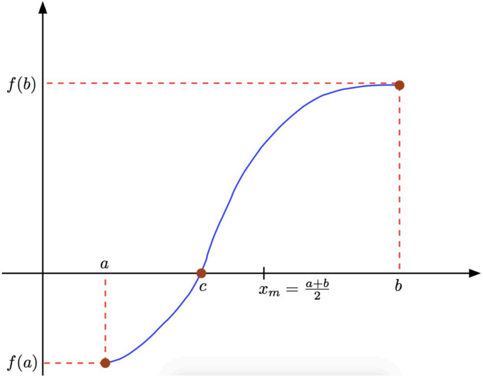
Rootfinding problems

Bisection

- Algorithm to find the root of a continuous real-valued function
- Based on the Intermediate Value Theorem: if f is continuous, and $f(a)$ and $f(b)$ have different signs, then f must have at least one root x in $[a, b]$
- Pros: very robust algorithm
- Cons: only applicable to one-dimensional problems

Rootfinding problems

Bisection: Graphical example



Source: `https://orionquest.github.io/Numacom/bisection.html`

Rootfinding problems I

Bisection

The pseudo-code for bisection is as follows

1. Determine the function f and the interval in which you want to find the zero (say $[a, b]$). **Important!** check that the sign of $f(a)$ is different to the sign of $f(b)$
   - Save them as `s_min` and `s_max`
2. Set an initial guess for the solution inside $[a, b]$. For example, $x^0 = \frac{(a+b)}{2}$
3. Set a tolerance and an initial error term (corresponding to the length of the interval)
4. Initiate the iterative step
   4.1 Evaluate the sign of the function at the candidate solution $f(x^0)$
      - If sign $f(x^0) =$ `s_min`, increase the initial guess
      - Else, decrease the initial guess

   4.2 Iterate until the length of the interval is smaller than the tolerance

go to MATLAB now! (example 1)

Rootfinding problems

Newton's Methods

Iterative method based on successive linearization

$$f(x) \approx f(x^k) + f'(x^k)(x - x^k) = 0 \tag{1}$$

Basic algorithm

1. Guess $x^{(0)}$ for the root of f. The super-script 0 denotes the iteration number
2. Using (1), update the guess with

$$x^{(k+1)} = x^{(k)} - [f'(x^{(k)})]^{-1} f(x^{(k)}) \tag{2}$$

3. Iterate until $x^{(k+1)}$ and $x^{(k)}$ are "close"

Any "problem" with this algorithm?

Rootfinding problems

Quasi-Newton's Methods

The previous algorithm requires the Jacobian (f') of the function

Quasi-Newton methods replace the Jacobian by variants of a numerical derivative

$$f'(x^{(k)}) \approx \frac{f\left(x^{(k)}\right) - f\left(x^{(k-1)}\right)}{x^{(k)} - x^{(k-1)}}$$

The iteration rule now reads as

$$x^{(k+1)} = x^{(k)} - \frac{x^{(k)} - x^{(k-1)}}{f\left(x^{(k)}\right) - f\left(x^{(k-1)}\right)} f\left(x^{(k)}\right) \tag{3}$$

Any "problem" with this algorithm?

Rootfinding problems I

MATLAB functions

Root of non-linear function

```
[x,fval,exitflag,output] = fzero('fun',x0,options)
```

Root of a system of non-linear equations

```
[x,fval,exitflag,output] = fsolve('fun',x0,options)
```

Input

- `'fun'`: function we want to find a zero
- `x0`: initial guess for the solution.
- Depending on the problem, you can set different `options`

Output

- `x`: zero of equation `'fun'`
- `fval`: value of the function at the optimal point
- `exitflag`: integer encoding the exit condition
- `output`: information about the root-finding process

Rootfinding problems II

MATLAB functions

Important!

When using `fzero`, you a single equation and must provide a single initial starting point. When using `fsolve`, you have a system of N equations and have to provide a N-dimensional vector as starting point.

Exercise 1

Write a function `f1` defining $f(x) = x\cos(x^2)$

1. Find the zero of this function using `fzero` and initial guess $x^0 = 1$
2. Repeat using $x^0 = 2$ as initial guess
3. Repeat using `fsolve`
4. How does the solution changes with the algorithm used? What about the initial guess?

Rootfinding problems

Setting options

For `fzero` use `optimset`:

- `Display`: level of display. For example `'off'` displays no output, while `'iter'` displays output at each iteration
- `TolX`: termination tolerance on `x`.

Example: `options = optimset('Display','off','TolX',1e-8)`

For `fsolve` use `optimoptions` and assign to `fsolve`

- `Display`: same as before
- `MaxIter`: maximum number of iterations allowed (positive integer)
- `TolFun`: termination tolerance on the function value (positive scalar)

Example:

```
options = optimoptions('fsolve','Display','off','TolFun',1e-8,'MaxIter',1000)
```

Note! you can set more options. See `help fzero` and `help fsolve` for details

Rootfinding problems

MATLAB functions: Passing arguments

Suppose you have a function

```
function y = myfunction(x,a,b)
```

Where `a` and `b` are arguments/parameters of the function.

You can use `fzero`/`fsolve` by calling `myfunction` as an *anonymous* function (recall Session 3)

```
fzero(@(x) myfunction(x,a,b),x0,options)
```

Numerical optimization

Two commonly used methods

Grid search

- Generate a fine grid of points and evaluate the function at each one. Choose x^* as the value that generates the highest/lowest $f(x^*)$

Derivative based

- At the optimum, the first derivative of the function is zero. As in root-finding methods, this is an iterative procedure that uses numerical derivatives

Numerical optimization

Grid search

Basic steps

1. Generate a first rough grid with n points and evaluate the function
2. Select the point that maximizes/minimizes the function on this grid, $x^{(k)}$
3. Refine the initial grid around the optimal point $x^{(k)}$
4. Select the new point that optimizes the function
5. Repeat steps 3-4 until the optimized values of consecutive iterations are "close"

Even though this method is clear and intuitive, it computationally expensive

go to MATLAB now! (example 2)

Numerical optimization

Newton-Raphson

Same spirit as for rootfinding algorithm: linear approximation

$$g(x) \approx g(x^k) + g'(x^k)(x - x^k)$$

Solving the first-order condition, $g(x) = f'(x)$

$$f'(x^k) + f''(x^k)(x - x^k) = 0$$

Updating rule

$$x^{k+1} = x^k - [f''(x^k)]^{-1} f'(x^k)$$

Numerical optimization

MATLAB functions

Minimum of unconstrained multivariable function using derivative-free method
`[x,fval,exitflag,output] = fimnsearch(fun,x0,options)`

Minimum of unconstrained optimization
`[x,fval,exitflag,output] = fminunc(fun,x0,options)`

Input/output and passing additional arguments is similar to `fzero` and `fsolve`

Note! for `fminsearch` you can set similar options as with `fzero` (using `optimset`). See `help fminsearch`.

Important!
When using `fminsearch` or `fminunc` you can have a N-dimensional vector as starting point, but the output of function `fun` must be a scalar

Exercise 2

Recall the function $f(x) = x\cos(x^2)$

1. Find the maximum of this function using `fminsearch`. Use as initial guess $x^0 = 1$
2. Repeat the previous step but using as initial guess $x^0 = 3$. Any difference?
3. Find the maximum of this function using `fminunc`. Use as initial guess $x^0 = 1$

Numerical optimization

Tips

Suppose you want to obtain θ which must satisfy some constraint

Get an unconstrained value ψ and transform as follows

Example	Constraint	Transformation
Variance	$\theta > 0$	$\theta = \psi^2$
		$\theta = \exp(\psi)$
Probability	$\theta \in (0, 1)$	$\theta = \frac{1}{1+\exp(\psi^{-1})}$
Stationary autoregressive parameter	$\theta \in (-1, 1)$	$\theta = \frac{\psi}{1+\lvert\psi\rvert}$
Shares	$\theta_1, \theta_2 \geq 0$	$\theta_1 = \frac{1}{1+\exp(\psi)}$
	$\theta_1 + \theta_2 = 1$	$\theta_2 = \frac{\exp(\psi)}{1+\exp(\psi)}$

Exercise 3

Consider a Cournot duopoly model, in which the inverse demand is:
$P(q) = q^{1/\eta}$

Each firm $i = 1, 2$ have costs: $C(q_i) = \frac{1}{2} c_i q_i^2$

Profits of firm i are: $\pi_i(q_1, q_2) = P(q_1 + q_2)q_i - C_i(q_i)$

The first order condition reads as

$$\frac{\partial \pi_i}{\partial q_i} = P(q_1 + q_2) + P'(q_1 + q_2)q_i - C_i'(q_i) = 0$$

Thus the market equilibrium outputs, q_1 and q_2, are the roots of the two nonlinear equations

$$f_i(q) = (q_1 + q_2)^{-1/\eta} - (1/\eta)(q_1 + q_2)^{-1/\eta - 1} q_i - c_i q_i = 0, \qquad i = 1, 2$$

Exercise 3

Assume $\eta = 1.6$, $c_1 = 0.6$ and $c_2 = 0.8$ and find the equilibrium of this economy

- Create a function called `cournot_res` that receives as input
  1. The initial guess for the vector of equilibrium output
  2. The values of parameters (η, c_1, c_2)

  and takes as output the "residual" of the equilibrium conditions
- Use one the solvers covered previously (which one? why?)

How the equilibrium output of each firm changes with the elasticity η?

- Set a linear space of 100 values between 0.6 and 5 for η
- Solve the model for each value on the grid
- Plot the equilibrium output of each firm against the elasticity

Exercise 4

Suppose you want to estimate the following model

$$y_i = \alpha + \beta x_i + u_i$$

- x_i: regressor of interest
- y_i: dependent variable
- $u_i \sim N(0, \sigma^2)$: error term, independent of x_i and independent across observations

The log-likelihood reads as

$$\ell = -\frac{n}{2}\log(2\pi) - n\log(s) - \frac{1}{2s^2}\sum_{i=1}^{n}(y_i - \alpha - \beta x_i)^2 \tag{4}$$

s^2: estimator of the variance error

Exercise 4

Load the dataset `us_data.xlsx`

- *spread*: difference between the 10-year yield and the 2-year yield
- *growth*: growth rate of GDP of the US

Estimate the parameters (α, β, s^2) by Maximum Likelihood Estimation (MLE)

- Create a function `us_mle` that receives as input
  1. The initial guess for the vector of parameters
  2. The regressor and the independent variable

  and takes as output the log-likelihood (4)

Hint! Do you have to impose any constraint in the estimation?

Compare the solution with Ordinary Least Squares: $\widehat{\beta} = (X'X)^{-1}X'Y$