Introduction to Matlab

Lesson 03 — Optimization

Rafael Serrano-Quintero

Department of Economics
University of Barcelona

Optimization Preliminaries

Optimization Preliminaries

- **Agents optimize** is the foundational assumption of economic theory.
  - Firms minimize costs / maximize profits.

  - Consumers maximize utility / minimize expenditures.

- Not only in theory ... Econometricians use
  - Maximum likelihood, least squares, method of moments ...

  - All optimization problems.

- We will learn the basics of how to state and solve this type of problems in Matlab.

Optimization Preliminaries — Definition of the Problem

The most general definition of an optimization problem

$$\min_{x \in \mathbb{R}^n} \quad f(x) \qquad \text{(Objective Function)}$$
$$\text{s.t. } g(x) = 0 \qquad \text{(Equality Constraints)}$$
$$h(x) \leq 0 \qquad \text{(Inequality Constraints)}$$

where

○ the Objective Function $f : \mathbb{R}^n \mapsto \mathbb{R}$

○ the m Equality Constraints $g : \mathbb{R}^n \mapsto \mathbb{R}^m$

○ the l Inequality Constraints $h : \mathbb{R}^n \mapsto \mathbb{R}^l$

Optimization Preliminaries — Definitions

Let $f : \mathcal{D} \subseteq \mathbb{R}^n \mapsto \mathbb{R}$.

Definition 1
A **critical point** $x^* \in \mathcal{D}$ of f satisfies $\nabla f(x^*) \equiv \left(\dfrac{\partial f}{\partial x_1}(x^*), \ldots, \dfrac{\partial f}{\partial x_n}(x^*) \right) = 0$.

Definition 2
A point $x^* \in \mathcal{D}$ is a **min** of f on $\mathcal{D}$ if $f(x^*) \leq f(x) \forall x \in \mathcal{D}$. It is a **strict min** if $f(x^*) < f(x) \forall x \neq x^* \in \mathcal{D}$.

Definition 3
A point $x^* \in \mathcal{D}$ is a **local (or relative or weak) min** of f on $\mathcal{D}$ if there is a ball $B_r(x^*)$ such that $f(x^*) \leq f(x) \forall x \in B_r(x^*) \cap \mathcal{D}$. It is a **strict local (or relative or weak) min** if $f(x^*) < f(x) \forall x \neq x^* \in B_r(x^*) \cap \mathcal{D}$

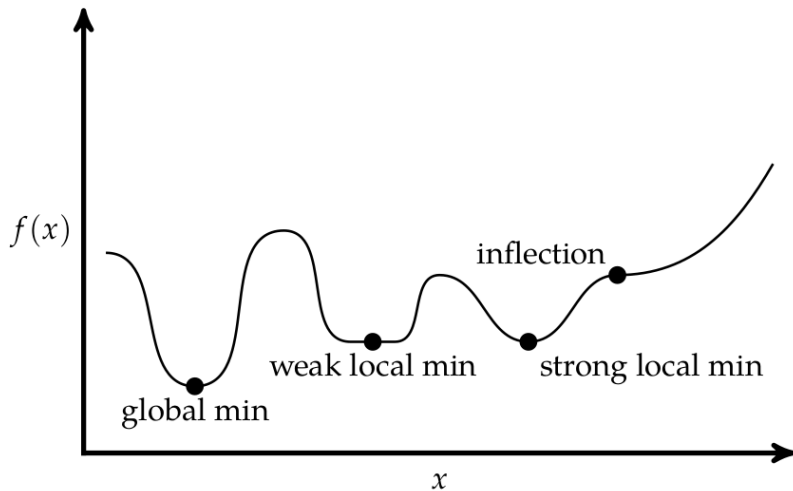# Optimization Preliminaries — Definitions

Figure 1: Classification of Critical Points. See Kochenderfer and Wheeler, 2019

Optimization Preliminaries — First Order Conditions

○ All points **need** that $f'(x^*) = 0$. Notice this is a **necessary condition** but not sufficient.

○ The inflection point also has $f'(x^*) = 0$ (An inflection point is where the sign of the second derivative changes).

Theorem 1

Let $f : \mathcal{D} \subseteq \mathbb{R}^n \mapsto \mathbb{R}$ be a $\mathcal{C}^1$ function. If x^* is a local min or max of f in $\mathcal{D}$ and x^* is an interior point of $\mathcal{D}$, then

$$\nabla f(x^*) = 0 \ \text{ for } i = 1, \ldots, n$$

Optimization Preliminaries — Second Order Conditions

Theorem 2
Let $f : \mathcal{D} \subseteq \mathbb{R}^n \mapsto \mathbb{R}$ be a $\mathcal{C}^2$ function. Suppose x^* is a critical point of f.

1. *If $H_f(x)$ is positive (negative) definite, then x^* is a strict local min (max) of f.*

2. *If $H_f(x)$ is indefinite, then x^* is neither a local min or max of f.*

Definition 4
Let $f : \mathcal{D} \subseteq \mathbb{R}^n \mapsto \mathbb{R}$ be a $\mathcal{C}^2$ function. The Hessian matrix H_f is a square $n \times n$ matrix whose $(i, j)-$th entry is defined by $(H_f)_{i,j} = \dfrac{\partial^2 f}{\partial x_i \partial x_j}$.

Optimization Preliminaries — Recap

- This has been a very brief recap on basic optimization.

- For a refresher, you can take a look at Simon and Blume, 1994, Chapters 17-19.

- We will cover the very basics of optimization and implementation in Matlab.

- All numerical optimization methods:
  - Search for **feasible choices**

  - Generate a sequence of **guesses**

  - Try to make the sequence **converge** to the true solution.

  Pretty similar to root finding algorithms…right?

Optimization in One Dimension

The Simplest Optimization Problem

The simplest optimization problem is **unconstrained optimization** in one dimension

$$\min_{x \in \mathbb{R}} \quad f(x)$$

where $f : \mathbb{R} \mapsto \mathbb{R}$. Why focusing on one dimension?

1. Illustrate techniques in a clear way.

2. Many multivariate methods boil down to solving a sequence of one-dimensional problems.

Optimization Methods — Categories

Four general categories for optimization methods:

1. Use derivatives.

2. Do not use derivatives.

3. Mixed methods.

4. Simulation-based methods.

We will look at methods in the first two.

Bracketing Method

Bracketing Method — The Intuition

○ Suppose $f : \mathbb{R} \mapsto \mathbb{R}$ is unimodal.

○ Let $a < b < c \in \mathbb{R}$ be three points such that

$$f(a), f(c) > f(b) \qquad (1)$$

○ Then, we know a minimum exists in $[a, c]$.
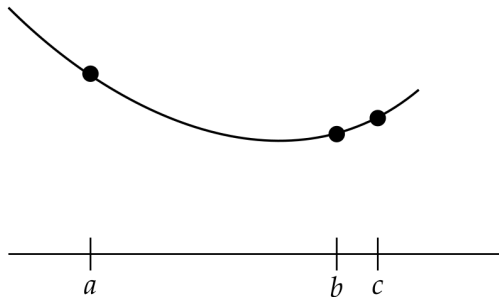
○ How to find the optimum?

Figure 2: See Kochenderfer and Wheeler, 2019

Bracketing Method — The Algorithm

1. Define h as the step size, a constant $\alpha > 1$, and a given initial x_0 and compute

$$f(x_0), f(x_0 \pm \alpha h), f(x_0 \pm \alpha^2 h), \ldots$$

until we find a triplet satisfying (1). Choose a stopping criterion ε.

2. If $b - a < c - b$, set $d = (b + c)/2$, otherwise, $d = (a + b)/2$. Compute $f(d)$.

3. If $d < b$ and $f(d) > f(b)$, replace (a, b, c) with (d, b, c). If $d < b$ and $f(d) < f(b)$, replace (a, b, c) with (a, d, b). If $d > b$ and $f(d) < f(b)$, replace (a, b, c) with (b, d, c). Otherwise, replace (a, b, c) with (a, b, d).

4. If $c - a < \varepsilon$, stop. Otherwise, go to step 2.

Bracketing Method — Remarks

- Slow. Similar to bisection.

- The stopping criterion is clear. If the length of the interval $[a, c]$ is sufficiently small, for practical terms, we have found the optimum.

- The method finds a **local minimum**, depending on the starting triplet (a, b, c) the method will converge to one minimum or another. <small>This is a fairly common problem in many methods.</small>

- If we know there is only one solution, the method always converges.

- Note we need three points in each iteration, depending on how costly it is to compute f this might be a problem.

Bracketing Method — An Example

Let's minimize

$$f(x) = \frac{x^2}{2} - x$$

○ The function has a global minimum in $x = 1$.

○ Let us divide the code into two blocks:
   1. Initial bracketing.

   2. Refining given the bracketing.

Bracketing Method — Initial Bracketing

1. Start from guess x_0 and compute $x_1 = x_0 + \alpha h$.

2. Evaluate $f(x_0)$ and $f(x_1)$. If $f(x_1) < f(x_0)$ keep increasing until $f(x_2) > f(x_1)$

3. Otherwise, change direction and increase interval until $f(x_2) > f(x_0)$.

4. Increase α in each step to make the interval larger.

Bracketing Method — Initial Bracketing

○ Start from guess $x_0 = -5$ (why not?) and compute increment.

```
1   % Parameters of bracketing method
2   h = 1e-2;
3   alp = 1.1;
4
5   % Step 1 - Initial bracketing
6   x0 = -5;
7   fx0 = fun(x0);
8   x1 = x0 + alp*h;
9   fx1 = fun(x1);
10  % If function is increasing in this direction, change
        direction
11  if fx1 > fx0
12      h = -h;
13  end
```

Bracketing Method — Initial Bracketing

○ Stablish the outer loop.

```
1  condition = true;
2  it = 1;
3  while condition
4  % Outer loop
5  end
```

○ Suppose $h < 0$

```
1  x2 = x0 + alp*h;
2  fx2 = fun(x2);
3  % We found the function increases!
4  if fx2 > fx0
5     a = x2;
6     b = x0;
7     c = x1;
8     condition = false;
9  end
```

Bracketing Method — Initial Bracketing

○ Suppose $h > 0$

```
1   x2 = x1 + alp*h;
2   fx2 = fun(x2);
3   % We found the function increases!
4   if fx2 > fx1
5      a = x0;
6      b = x1;
7      c = x2;
8      condition = false;
9   end
```

○ If we have not found the function increases, update α

```
1   alp = alp*2;
```

○ Then, simply put all together in nested `if-else` statements.

Bracketing Method — Refining Bracketing

○ Initiallize loop

```matlab
1    while (difference > tol) && (it < maxit)
2        % Stuff goes here
3    end
```

○ Define d and compute $f(d)$

```matlab
1    % Step 2: define d and compute f(d)
2    if (b - a) > (c - b)
3        d = (a + b)/2;
4    else
5        d = (b + c)/2;
6    end
7    fd = fun(d);
```

Bracketing Method — Refining Bracketing

○ Refine the interval if $d < b$

```
1 % Step 3: refine the
      interval
2 if d < b
3     if fd > fb
4         a1 = d;
5         b1 = b;
6         c1 = c;
7     else
8         a1 = a;
9         b1 = d;
10        c1 = b;
11    end
12 else
```

○ Refine the interval if $d > b$

```
1 % Step 3: refine the
      interval
2 else
3     if fb < fd
4         a1 = a;
5         b1 = b;
6         c1 = d;
7     else
8         a1 = b;
9         b1 = d;
10        c1 = c;
11    end
12 end
```

Rename a1 by a, compute $(c - a)$, and update iteration counter.

Newton-Raphson Method

Newton-Raphson Method

- Familiar? Yes! It is very closely related to the root finding algorithm!

- Given an initial x_0, compute a second order Taylor expansion around x_0:

$$f(x) \approx f(x_0) + f'(x_0)(x - x_0) + \frac{f''(x_0)}{2}(x - x_0)^2$$

- Minimizing this approximation, the FOC we get is

$$f'(x_0) + f''(x_0)(x^* - x_0) = 0$$

solving for x^*

$$x^* = x_0 - \frac{f'(x_0)}{f''(x_0)}$$

- Which is the **same iteration scheme** that we saw previously!

Newton-Raphson Method — Remarks

○ Newton-Raphson's method tries to finds **critical points**.

○ We **must check** $f''(x^*)$ to check what we have found.

○ Problems:

  ○ Convergence is not ensured.

  ○ $f''(x)$ might be difficult to compute. If we rely on finite difference methods, we will be adding errors.

  ○ Very sensitive to initial conditions.

○ From Fernández-Villaverde's slides: *If you do not know where you're going, at least go slowly.*

Newton-Raphson Method — The Algorithm

1. Choose initial guess x_0 and stopping parameters $\delta, \varepsilon > 0$.

2. Use the iteration scheme
$$x_{k+1} = x_k - \frac{f'(x_k)}{f''(x_k)}$$

3. If
$$\frac{|x_k - x_{k+1}|}{1 + |x_k|} < \varepsilon \text{ and } |f'(x_k)| < \delta$$
stop. Otherwise, go to step 1.

Newton-Raphson Method — An Example

○ Apply the Newton-Raphson method to solve

$$\min_{x \in \mathbb{R}^n} \ f(x) = \frac{x^2}{2} - \log(x^2)$$

○ The true solution is $x^* = \pm\sqrt{2} \approx \pm 1.4142\ldots$

○ Your choice of starting point x_0 will determine to which minimum you converge.

Optimization in Matlab

Unconstrained Optimization

Unconstrained Optimization in Matlab

- We have covered two basic methods of unconstrained optimization.

- We are going to see now how to solve **(un)constrained optimization** problems using Matlab's routines.

- We start with the unconstrained optimization routine `fminunc`.

- This solves unconstrained multivariate optimization problems.

- It can use two types of algorithms. Both based on Newton-Raphson methods.
  - BFGS which is a quasi-Newton method.

  - Trust-region methods. Approximates the objective function in a subset, if this approximates well the function, it extends the region, otherwise, it contracts the region.

Unconstrained Optimization — `fminunc`

The basic syntax of `fminunc` takes as inputs a function `fun` and an initial point `x0`

```
1  [x, fval] = fminunc(fun,x0)
```

The basic output is x^* and $f(x^*)$

- This computes the derivatives numerically

- You can also supply the gradient and the Hessian.

- To supply gradient and Hessian, you need to write it in the function script.

Unconstrained Optimization — `fminunc`

Let's move to multivariate optimization and minimize the Rosenbrock function

$$f(x_1, x_2) = 100 \left(x_2 - x_1^2 \right)^2 + \left(1 - x_1 \right)^2$$

The gradient is

$$\nabla f(x) = \begin{pmatrix} -400(x_2 - x_1^2)x_1 - 2(1 - x_1) \\ 200(x_2 - x_1^2) \end{pmatrix}$$

We will write $f(x_1, x_2)$ as a function script that will give as outputs the value of f and the value of the gradient. Check out `nargout` and `nargin`

Unconstrained Optimization — `fminunc`

- The function script

```
function [f, fgrad] = rosenbrock(x)
  % Not necessary, but for clarity we unpack the two inputs
  x1 = x(1);
  x2 = x(2);

  % Compute f
  f = 100.*(x2 - x1.^2).^2 + (1 - x_1).^2;

  % Compute gradient (if necessary)
  if nargout > 1
      % Notice this is a vector, and the order MATTERS!
      fgrad = [-400*(x2 - x1.^2).*x1 - 2*(1 - x1);
      200*(x2 - x1.^2)];
  end
end
```

Unconstrained Optimization — `fminunc`

- The optimization call

```matlab
% Initial point
x0 = [14, 4];
% Optimization call
[x, fval] = fminunc(@(x)rosenbrock(x),x0);
```

- This **does not** tell `fminunc` that we must use the gradient.

- We need to use an options parser. Check out `optimoptions`.

```matlab
% Add options
opts = optimoptions('fminunc','Algorithm','trust-region'...
                              'SpecifyObjectiveGradient', true);
[xg, fvalg] = fminunc(@(x)rosenbrock(x),x0,opts);
```

Unconstrained Optimization — `fminunc`

○ The optimization without gradient yields $f(x_1, x_2) = 1.4045$

○ The optimization **with** gradient yields $f(x_1, x_2) = 1.459 \times 10^{-11}$. Quite a change!

○ Note however the initial guess is pretty bad $x = (14, 4)$ when it should be close to $(1, 1)$.

○ Improving the guess reduces the differences. Numerical derivatives work well in this case.

○ A derivative free solver for unconstrained optimization is `fminsearch`.

Constrained Optimization

Constrained Optimization

- Matlab offers several options.

- `fminbnd` — Finds a minimum of a **single-variable** function $f(x)$ in a given interval. The constraints are of the type $a \leq x \leq b$.

- `fmincon` — It is a multivariate constrained optimization command. It accepts constraints of the type $g(x) \leq 0$ and $h(x) = 0$.

- There are others that you can check out here

Constrained Optimization — `fmincon`

- Let's focus on `fmincon` which is quite general for the type of problems you will most likely encounter.

- The general declaration of the function is

```
x = fmincon(fun,x0,A,b,Aeq,beq,lb,ub,nonlcon,options)
```

- We know `options`, `fun`, and `x0` from before.

- `A` and `b` are a matrix and a vector respectively denoting linear constraints such as $Ax \leq b$.

- Similarly, `Aeq` and `beq` denote $Ax = b$.

- `lb` and `ub` are lower and upper bounds respectively for each variable.

Constrained Optimization — `fmincon`

○ The general declaration of the function is

```
x = fmincon(fun,x0,A,b,Aeq,beq,lb,ub,nonlcon,options)
```

○ `nonlcon` are the nonlinear constraints that are supplied in function scripts. These should take the form $h(x) = 0$.

○ Optimizing the Rosenbrock function in a unit circle would take as `nonlcon`

```
ucircle = @(x) c = (x(1)-1)^2 + (x(2)-1)^2 - 1;
[x, fval] = fmincon(@(x)rosenbrock(x),x0
    ,[],[],[],[],[],[],@ucircle);
```

○ The empty brackets [] denote empty arguments.

Optimization — General Tips

Optimization — General Tips

- Try to use inequality constraints, $g(x) \leq 0$ is easier to solve than $g(x) = 0$. Recall tolerances.

- A good initial guess is extremely important when optimizing nonlinear functions.

- Good approaches to solving complex problems:
    - Solve an easier version of the problem to get a good guess.

    - Change of variables.

    - Combine local and global solution methods.

- Try to normalize variables as much as you can, unit free problems are typically easier.

- Take advantage of the problem you are tackling by simplifying the computations as much as you can.