Essential numerical tools and perturbation analysis (1.c)

Optimization

Pablo Winant

African Econometric Society Worskhop 14/12/23

Introduction

Optimization is *everywhere* in economics:

- model an agent's behaviour: what would a rational agent do?
  - consumer maximizes utility from consumption
  - firm maximizes profit
- an economist tries to solve a model:
  - find prices that clear the market

A Tale of Two Optimization Problems

minimization/maximization.

$$\min_{x\in X} f(x) \text{ or } \max_{x\in X} f(x)$$

root finding:

$$\text{find } x \in X \text{ such that} f(x) = 0$$

,

Often a minimization problem can be reformulated as a root-finding problem

$$x_0 = argmin_{x\in X} f(x) \overbrace{\iff}^{??} f'(x_0) = 0$$

Warning:

When using first order conditions to minimize a function one must pay attention to the two caveats:

- that the root does not correspond to a local *maximum*
- that the minimum is *global* (for $\forall x \in X$) not local

▶ Local/Global Illustration

Optimization tasks come in many flavours

▶ Continuous versus discrete optimization

▶ Constrained and Unconstrained optimization

▶ Stochastic vs Deterministic

▶ Local vs global Algorithms

Math vs practice

- The full mathematical treatment will typically assume that f is smooth ($\mathcal{C}_1$ or $\mathcal{C}_2$ depending on the algorithm).
- In practice we often don't know about these properties
  - we still try and check that we have a local optimal
- So: fingers crossed

▶ A complicated surface

What you need to know

- handcode simple algos (Newton, Gradient Descent)
  - very useful 👍
- understand the general principle of the various algorithms to compare them in terms of
  - robustness

- efficiency
- accuracy
- then you can just switch the various options, when you use a library...
  - root-finding: Roots.jl (1d) and NLsolve.jl
  - optimization: Optim.jl
  - more advanced modeling: JuMP.jl (comparable to GAMS)

Unconstrained Optimization

- Minimize $f(x)$ for $x \in \mathbf{R}^n$ given initial guess $x_0 \in \mathbf{R}^n$
- If you have intuitions from the 1d case, they still apply
  - replace derivatives by gradient, jacobian and hessian
  - recall that matrix multiplication is not commutative
- Some specific problems:
  - update speed can be specific to each dimension
  - saddle-point issues (for minimization)

Quick terminology

Function $f: R^p \rightarrow R^q$

- *Jacobian*: $J(x)$ or $f'\_x(x)$, $p \times q$ matrix such that: $J(x)_{ij} = \frac{\partial f(x)_i}{\partial x_j}$
- *Gradient*: $\nabla f(x) = J(x)$, jacobian when $q = 1$
- *Hessian*: denoted by $H(x)$ or $f''_{xx}(x)$ when $q = 1$:

$$H(x)_{jk} = \frac{\partial f(x)}{\partial x_j \partial x_k}$$

In the following explanations, $|x|$ denotes the supremum norm, but most of the following explanations also work with other norms.

Newton-Raphson Root-Finding

- Algorithm:
  - start with x_n
  - compute $x_{n+1} = x_n - J(x_n)^{-1} f(x_n) = f^{\text{newton}}(x_n)$
  - stop if $|x_{n+1} - x_n| < \eta$ or $|f(x_n)| < \epsilon$

- Convergence: **quadratic**

▶ Newton Raphson Illustration

Newton-Raphson Root Finding (2)

- what matters is the computation of the step $\Delta_n = \color{red}{J(x_n)^{-1}} f(x_n)$
- don't compute $J(x_n)^{-1}$
  - it takes less operations to compute X in $AX=Y$ than A^{-1} then $A^{-1}Y$
  - in Julia: `X = A \ Y`
- strategies to improve convergence:
  - *dampening*: $x_n = (1-\lambda)x_{n-1} - \lambda\Delta_n$
  - *backtracking*: choose k such that $|f(x_n - 2^{-k}\Delta_n)| < |f(x_{n-1})|$
  - *linesearch*: choose $\lambda \in [0,1]$ so that $|f(x_n - \lambda\Delta_n)|$ is minimal

Exercise 1: Simple Newton Algorithm

Choose a two dimensional function with a zero.

Write a method `zero_newton(f::Function, x0::Vector{Float64})` which computes the zero of a vector valued function `f` starting from initial point `x0`.

Change the signature of the function `zero_newton(f::Function, x0::Vector{Float64}, backtracking=true)` and implement backtracking in each iteration.

φ (generic function with 1 method)

```
function φ(x::Vector{Float64})
    y = [x[1]^2 + x[2]^2 - 0.5 , x[1] + x[2] - 1  ]
    dy = [ (2*x[1]) 1.0 ; 1.0 (2*x[1]) ]
    return (y, dy)
end
```

```
([-0.45, -0.7], 2×2 Matrix{Float64}:
                0.4  1.0
```

```
φ([0.2, 0.1])
```

```
# what are the zeros of φ?
```

zero_newton (generic function with 1 method)

```
function zero_newton(f::Function, x0::Vector{Float64})
end
```

Multidimensional Gradient Descent

- Minimize $f(x) \in R$ for $x \in R^n$ given $x_0 \in R^n$
- Algorithm
  - start with x_n
  - set $x_{n+1} = (1-\lambda)x_n - \lambda \nabla f(x_n)$
  - stop if $|x_{n+1} - x_n| < \eta$ or $|f(x_n)| < \epsilon$
- Comments:
  - lots of variants
  - **automatic differentiation** software makes gradient easy to compute
  - convergence is typically **linear**

▶ Gradient Descent Illustration

Multidimensional Newton Minimization

- Algorithm:
  - start with x_n
  - compute $x_{n+1} = x_n - {\color{red}H(x_n)^{-1}}{\color{green}J(x_n)'}$
  - stop if $|x_{n+1} - x_n| < \eta$ or $|f(x_n)| < \epsilon$
- Convergence: **quadratic**

▶ Newton Illustration

Practical Problems

- hessian $H(x_n)$ is hard to compute efficiently
- rather unstable

There are ways to approximate the hessian without a full evaluation

- quasi-newton
- gauss-newton
- Levenberg-Marquardt

Exercise 2: Profit optimization by a monopolist

A monopolist produces quantity q of goods X at price p. Its cost function is $c(q) = 0.5 + q(1 - qe^{-q})$

The consumer's demand for price p is $x(p) = 2e^{-0.5p}$ (constant elasticity of demand to price).

Write down the profit function of the monopolist and find the optimal production (if any) numerically. You can use the `Optim.jl` library.

Hint

```
profits (generic function with 1 method)
```

```
begin
    p(q) = -2log(q/2)
    c(q) = 0.5 + q*(1-q*exp(-q))
    R(q) = q*p(q)
    profits(q) = R(q) - c(q)
end
```

```
@bind q_sl Slider(0.001:0.01:  0.5)
```

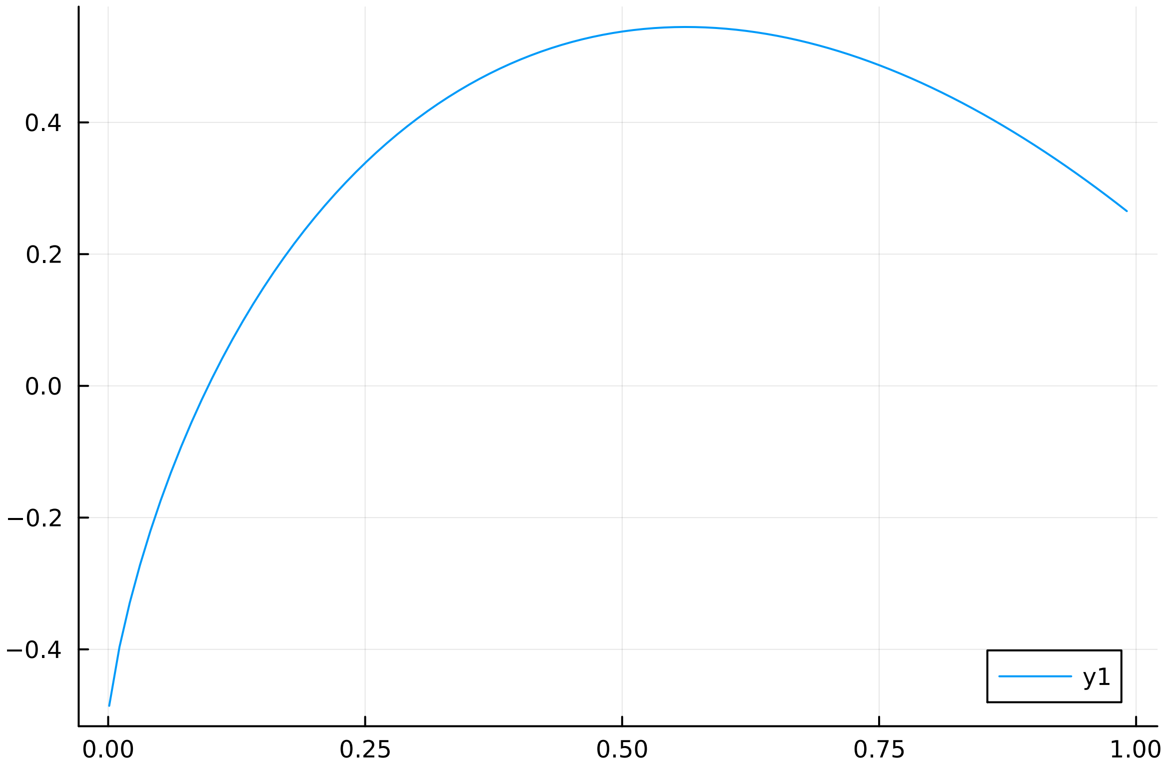
```
-0.48579719608041594
```

```
profits(q_sl)
```

```
[-0.485797, -0.396414, -0.3292, -0.27172, -0.220625, -0.174282, -0.131716, -0.0922769, -0.
```

```
begin
    qvec = 0.001:0.01:  1.0
    πvec = profits.(qvec)
end
```

```
using Plots
```




```julia
plot(qvec, πvec)
```

```julia
using Optim
```

```
res =  * Status: success

       * Candidate solution
         Final objective value:     -5.448588e-01

       * Found with
         Algorithm:     Nelder-Mead

       * Convergence measures
         √(Σ(yᵢ-ȳ)²)/n ≤ 1.0e-08

       * Work counters
         Seconds run:   0  (vs limit Inf)
         Iterations:    7
         f(x) calls:    17
```

```julia
res = optimize(u->-profits(u[1]) , [0.25])
```

```
min0 =  [0.561865]
```

```julia
min0 = res.minimizer
```

```julia
println("Profit is maximum for q=$min0")
```

```
Profit is maximum for q=[0.561865234375]
```

Dealing with constraints

Consider the optimization problem:

$$\max U(x_1, x_2)$$

subject to to the constraint $p_1 x_1 + p_2 x_2 \leq B$

where $U(.)$ is concave, p_1, p_2 and B are given.

This is a constrained maximization problem. Some optimization algorithms are equipped to deal with the constraint.

One can also reformulate the maximization problem as a first order system.

Karush-Kuhn-Tucker conditions

- If $(x^\star, y^\star)$ is optimal there exists λ such that:
  - $$(x^\star, y^\star)$$
    maximizes lagrangian $\mathcal{L} = U(x_1, x_2) + \lambda(B - p_1 x_1 - p_2 x_2)$
  - $$\lambda \geq 0$$
  - $$B - p_1 x_1 - p_2 x_2 \geq 0$$
  - $$\lambda(B - p_1 x_1 - p_2 x_2) = 0$$
- The three latest conditions are called "complementarity" or "slackness" conditions
  - they are equivalent to $\min(\lambda, B - p_1 x_1 - p_2 x_2) = 0$
  - we denote $\lambda \geq 0 \perp B - p_1 x_1 + p_2 x_2 \geq 0$
- Lagrange multiplier λ can be interpreted as the welfare gain of relaxing the constraint.

Karush-Kuhn-Tucker conditions

- We can get first order conditions that factor in the constraints:
  - $$U'_x - \lambda p_1 = 0$$
  - $$U'_y - \lambda p_2 = 0$$

- $$\lambda \geq 0 \perp B - p_1 x_1 - p_2 x_2 \geq 0$$
- It is now a nonlinear system of three equations in three unknowns with complementarities
  - a.k.a. Nonlinear Complementarity Problem (NCP)
  - there are specific solution methods to deal with it
  - and commercial solvers (knitro, PATH)
  - but one can often use a simple, easy trick...

Smooth method

- Consider the *Fisher-Burmeister* function $\phi(a,b) = a + b - \sqrt{a^2 + b^2}$
  - It is infinitely differentiable, except at $(0,0)$
  - Show that $\phi(a,b) = 0 \iff \min(a,b) = 0 \iff a \geq 0 \perp b \geq 0$
- After substitution in the original system, we obtain:
  - $$U'_x - \lambda p_1 = 0$$
  - $$U'_y - \lambda p_2 = 0$$
  - $$\phi(\lambda, B - p_1 x_1 - p_2 x_2) = 0$$
- And we can use our preferred nonlinear solver

> Tip
>
> Fun fact: the formulation with a $\min$ is nonsmooth but also works quite often

Exercise 3: Constrained Optimization

Consider the function $f(x,y) = 1 - (x - 0.5)^2 - (y - 0.3)^2$.

Use Optim.jl to maximize f without constraint. Check you understand diagnostic informations returned by the optimizer.

```
Enter cell code...
```

Now, consider the constraint $x < 0.3$ and maximize f under this new constraint.

```
Enter cell code...
```

Reformulate the problem as a root finding problem with lagrangians. Write the complementarity conditions.

```
1 Enter cell code...
```

Solve using NLSolve.jl

```
1 Enter cell code...
```

Adapt the `zero_newton` function from before to incorporate bound informations, and use the Fisher-Burmeister transform to solve the system.

Table of Contents

Essential numerical tools and perturbation analysis (1.c)
Optimization
Introduction
A Tale of Two Optimization Problems
Optimization tasks come in many flavours
Math vs practice
What you need to know
Unconstrained Optimization
Quick terminology
Newton-Raphson Root-Finding
Newton-Raphson Root Finding (2)
Exercise 1: Simple Newton Algorithm
Multidimensional Gradient Descent
Multidimensional Newton Minimization
Exercise 2: Profit optimization by a monopolist
Dealing with constraints
Karush-Kuhn-Tucker conditions
Karush-Kuhn-Tucker conditions
Smooth method
Exercise 3: Constrained Optimization