julia> Lecture Three: Numeric Computation

Course: Computational Bootcamp

Name: John Higgins

Date: June 24, 2024

[~]$ _

julia> Today's goals

1. Survey different numerical optimization techniques
2. Understand how to apply them in Julia

julia> Optimization

> Suppose you want to find the minimum or maximum of a function, possibly subject to some constraints
> Your objective might be the following:

$$\min_x f(x)$$

$$\text{s.t.} \quad g(x) \geq 0$$

> Note: maximizing a function f is equivalent to minimizing the function $-f$, meaning that minimization techniques can be applied to maximization problems too

julia> When do we need to do this?

- Agent's choice problem:
  - Consumer maximizes utility subject to some constraints
- Estimation:
  - Ordinary Least Squares (OLS)
  - Simulated Method of Moments (SMM): choose parameters to minimize the distance between observed moments and simulated moments
  - Maximum Likelihood Estimation (MLE): choose parameters which maximize likelihood of the observed data

julia> How do we do this??

- Pencil and paper: Take derivatives, use Lagrangian, ...
  - This is what you did in first year
  - Not always possible
- Grid search: Guess and check
  - Evaluate the function at many different parameter values and find the point with the lowest value
  - Accurate solution requires a very fine grid
  - This can be very slow, especially in many dimensions (10 parameters, 10 possible values $\implies$ 10 *Billion* combinations to check)
- Optimization algorithms ("Smart" guess and check):
  - Use limited information about function (such as derivatives) and previous guesses to decide next guess
  - Lots of choices here
  - Sometimes does not work; no perfect algorithm

julia> Boxed constrained univariate optimization method

- We know solutions exists in $[a, b]$:
  - e.g. Search intensity normalized between 0 and 1
  - e.g. Hours of work must be between 0 and 80
  - e.g. Savings is between 0 and net worth
- Common algorithm: Brent's method
  - Does not require derivatives
  - Uses bisections, secants, and inverse quadratic interpolation...

julia> Multivariate Optimization

- Optimization in multiple dimension can be challenging
  - Curse of dimensionality
- If function is twice differentiable and has analytic gradient/Hessian, you can use Newton's method
  - Knowing the derivative gives you information about the function's local behavior, meaning the algorithm can find parameter values quicker
- In Economics, you will rarely be dealing with such a nice function in your optimization

julia> Quasi-Newtonian Methods

- Numerically approximates gradient and Hessian and applies similar updating rule to Newton's method
  - Finite differences: for a small enough δ,

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

  - Automatic differentiation: Computer attempts to apply chain run to components of the function (not super applicable in structural estimation, to my knowledge)
- Most common algorithm of this type is BFGS or (L-BFGS)
- Problems:
  - Approximation of derivative might not be well-behaved, and choice of δ is non-trivial
  - Can be computationally expensive, especially if function takes a long time and there are many parameters

julia> Derivative-free Methods

- Most common method: Nelder-Mead (link)
  - Easy to implement, but unfortunately does not always work well
- Intuition: tries to use the intuition of "rolling down a hill" by updating the set of candidate solutions until convergence
- Constructs a simplex with the three best candidates so far, searches for new candidates, replaces the worst candidate in the simplex
- Simplex eventually shrinks (hopefully); algorithm stops when difference between function values is sufficiently small
- Drawbacks:
  - Doesn't directly use local information about function behavior
  - Can be easily confused by local minima
  - Can be easily stuck in flat areas or go around in circles
  - Convergence is slow

julia> Randomization

- Randomization can improve the performance of our algorithms, especially when we have:
  - Many local minima
  - Non-smooth objective functions
- Basin hopping:
  - Guess initial point and run algorithm (such as NM)
  - From candidate solution, randomly "hop" to new initial point and run algorithm again
- Laplace-type estimator:
  - Randomly jump around parameter space
  - Accept (with some randomization) better guesses
  - Size of jump updates based on fraction of accepted guesses

julia> Tolerance

- Many (but not all) optimization algorithms end after reaching some preset tolerance level
- Lower tolerance improves quality of answer but increase time algorithm will run
- Important to keep nested optimization in mind
- Common procedure in economics:
  - Inner loop solves model given a parameter guess: e.g. make choices to maximize utility
  - Outer loop chooses parameters to minimize difference between model and data (maximum likelihood/SMM/Nested Fixed Point/BLP Contraction)
- Loose tolerance in inner loop may lead to bad parameter guesses, failed convergence
- However, tight tolerance in inner/outer loop can slow down convergence (when is it good enough?)

julia> Advice

- Try to reduce dimensions, especially when getting started
- You can plot in 2D and 3D, but 4D becomes quite challenging
- Understand the shape of the function you are trying to optimize:
  - Is it smooth? Does it have kink points?
- There is no perfect algorithm
- Transformation of variables and scale may be helpful. $\log()$, $\exp()$, . . .
  - Quick reference on transformations can be found here: https://jblevins.org/notes/bijections