Optimization

Tax Policy Research Unit and World Bank
Jason DeBacker

August 2019

Numerical Optimization

- We will cover two related problems of numerical optimization
  1. Minimization
     - This is where we find the value of an argument that yields the minimum value of a function
     - This will cover maximization as if we want to find $\max f(x)$ we can just do $\min -f(x)$
  2. Root finding
     - This is where we find the value of the arguments that yield the root (or zero) of a function
- Our focus will be on non-linear functions
  - Linear programming methods can be used to find the minimum of a linear function
- After this brief introduction to methods of optimization, we will learn how to implement them in Python

Optimzation methods

Optimization methods can generally be divided into 2 types:

1. Gradient-based convergence methods
    - Faster convergence to the minimum
2. Non-gradient-based convergence methods
    - More robust convergence to the minimum

- Gradient-based methods are preferred if the function is smooth and you have a good initial guess
- If the function is not smooth, or your initial guess if far from the true minimum, gradient-based methods may not converge

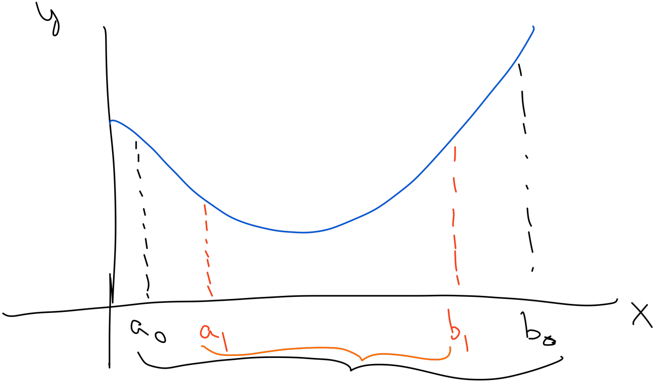
Some basics of numerical optimization

- We use numerical methods because it is often difficult (or impossible) to compute the solution to the optimization problem analyticallly
- When using numerical methods, we are *approximating* the true minimum
  - We'll therefore want to think carefully about the tolerance we use in this approximation
- There is no guarantee we will be able to find the globabl minimum of a non-linear problem

Numerical optimization in 1 dimension

- With optimization in a single dimension, we can use methods that guarantee we converge to the true solution (within the tolerance of our approximation)
- Non-gradient based methods:
  - Bisection method
  - Golden Ratio search
- Gradient-based methods
  - Newton's method
  - Method of steepest descent
- Hybrid methods
  - Brent's method

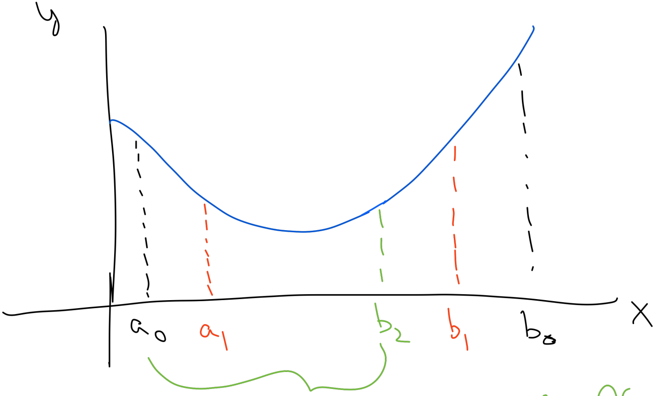
Illustration of Golden Ratio Search



$f(a_1) < f(b_1)$

$\Rightarrow$ use $[a_0, b_1]$ as next interval

Illustration of Golden Ratio Search

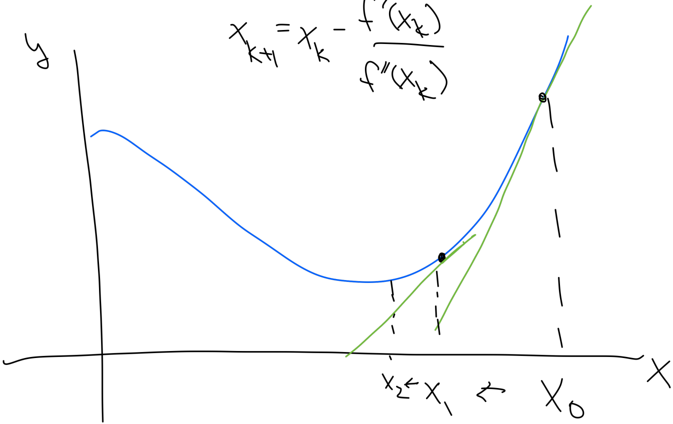


now compare $f(b_2)$ to $f(a_1)$

$f(b_2) < f(a_1) \Rightarrow$ next interval (a_1, b_1)

Illustration of Newton's method

$$x_{k+1} = x_k - \frac{f'(x_k)}{f''(x_k)}$$



Numerical optimization in multiple dimensions

- No methods will guarantee a solution
- Non-gradient based methods:
  - Nelder-Meade method
- Gradient-based methods
  - Newton's method
  - Gradient descent
  - Conjugate gradient method
  - BFGS
  - Gauss-Newton
- Hybrid methods
  - Powell's method
- Global solution methods
  - Simulated annealing (also called "basin-hopping")
  - Differential evolution

Root-finding

- When minimizing a problem, we solved:

$$\min_x f(x) \tag{1}$$

- The problem of finding a root is given by:

$$\text{find } x^* \text{ such that } f(x^*) = 0 \tag{2}$$

Root-finding

- The root-finding and minizer are related.
- In some cases, the root-finding problem can be transformed into a minimization problem.
  - e.g., $\min_x ||f(x) - 0||^2$
- Because these problems are similar, root-finding algorithms will use some of the same methods as minimzer algorithms
  - This includes both gradient and non-gradient methods

Summary

- In complex (multi-dimensional) problems, we have no guarantee of finding a solution
- Gradient-based methods are fastest, but less robust
- Starting values can be very important, especially for gradient-based methods

A rule of thumb

Some advice from Humphreys and Jarvis (2017):

1. If the dimension of the problem is not too big
    1. If x_0 is close to x^*
        1. If computing $(D^2 f(x))^{-1} Df(x)$ is cheap and accurate, use Newton method as it has fastest convergence
        2. If computing $(D^2 f(x))^{-1} Df(x)$ is expensive or error-prone, then use Gauss-Newton (possibly with Levenberg-Marquart Modification) if $f(x)$ of the form $f = r^T r$, else try BFGS
    2. If x_0 is not close to x^*, use gradient descent method (not necessarily steepest decent) for several steps to get closer to x^* then switch back to 1(a)
    3. If other methods are not converging rapidly, try conjugate gradient
2. If the dimension of the problem is large and the Hessian sparse, use conjugate gradient methods.