ECON 815 - Computational Methods for Economists

Fall 2017

Notes on Numerical Optimization

Numerical Optimization

- We'll consider methods of numerical *minimization*. But note that if you want to maximize a function (e.g., a likelihood function), you'll use the same methods because $maxf(x) = \min -f(x)$
- Also, we'll focus on method s suitable for nonlinear problems (linear programming methods can be used to find the minimum of linear problems)
- Optimization methods can generally be divided into two types:
  1. Gradient-based methods
     - Faster convergence to minimum
  2. Non-gradient based methods
     - More robust convergence to minimum
- Gradient-based methods are preferred if the function is smooth and you have a good initial guess
- When your function isn't smooth, optimization becomes much more difficult

Numerical Optimization in One dimension

1. Golden Ratio search (a non-gradient method)
   - The Golden Ratio search method finds the minimum of a function by narrowing down the state space in a specific way
   - In particular, it will break up the state space into intervals of equal length
   - Algorithm to find the minimum of $f(x)$:
     (a) Choose two points, a_1 and b_1 in the interval over which the problem is defined, $[a_0, b_0]$
        - The key to the Golden Ratio search method is in how these are chosen.
        - a_1 and b_1 are chosen so that the "outer intervals" are the same size. i.e., $a_1 - a_0 = b_0 - b_1 = \rho(b_0 - a_0)$
        - $\implies \rho = \frac{b_0 - a_0}{b_0 - b_1} = \frac{3-\sqrt{5}}{2} \simeq 0.381 = 1 + \frac{1}{\phi}$ where $\phi =$ the Golden Ratio

     (b) Evaluate $f(a_1)$ and $f(b_1)$

     (c) If $f(a_1) < f(b_1)$, then we know the minimum is in the interval $[a_0, b_1]$ (and if $f(a_1) > f(b_1)$, we'd know the min is in the interval) $[a_1, b_0]$)

     (d) Choose another point in (a_0, b_1), call this point b_2
        - As before, we choose this point so that outer intervals are the same length
        - i.e., $a_1 - a_0 = b_2 - b_1$

     (e) Evaluate $f(b_2)$

     (f) Compare $f(b_2)$ to $f(a_1)$ to decide which size of b_2 the next point will be on.

     (g) Repeat from Step (d)
   - Convergence:
     - Convergence is linear
     - The intervals of uncertainty shrink at a rate of $1 - \rho$. i.e., the size of the interval at iteration k is $(1 - \rho)^k(b_0 - a_0)$

- This is a robust method - it will always converge
- Golden Ratio search is similar to the Bisection method
  - Both converge linearly
  - Golden Ratio is more efficient in that it requires fewer function evaluations (one per iteration rather than two – see Step (e) above)

2. Brent's method (a combination of gradient and non-gradient methods)

- This method combines inverse quadratic interpolation (IQI) with a search method (e.g., the Golden Ratio search method)
  - At it's best, it converges super-linearly (by using IQI)
  - At worst, it converges linearly (by using search method)
  - By being a hybrid, it retains robustness
- Algorithm:
  (a) Approximate the function to be minimized with a quadratic function
  (b) This function can be evaluated at 3 points, a, b, and c, where a and c bracket the minimum
  (c) The minimum of this parabola is given by:

$$x = b - \frac{1}{2}\frac{(b-a)^2[f(b)-f(c)] - (b-c)^2[f(b)-f(a)]}{(b-a)[f(b)-f(c)] - (b-c)[f(b)-f(a)]} \tag{1}$$

  (d) If $a < x < b$, do (b) and (c) on a new interval with 3 points $= a, x, b$
  (e) If $b < x < c$, do (b) and (c) on a new interval with 3 points $= b, x, c$
  (f) (c) could send x away from the minimum
  (g) Thus check that x in the interval (b, c)
  (h) If x outside this interval, switch to Golden Ratio method
- Convergence:
  - Converges at a rate of about 1.83 ($> 1, < 2 \implies$ super linear convergence)

3. Newton method (a gradient-based method)

- Algorithm:
  (a) For an initial guess, x_0, let $q(x)$ be a Taylor expansion of degree 2 of $f(x)$ around x_0.

$$q(x) \equiv f(x_0) + f'(x_0)(x - x_0) + \frac{1}{2}f''(x_0)(x - x_0)^2 \tag{2}$$

  (b) Find the value of x that minimizes $q(x)$. Call this x_1
    - x_1 solves $q'(x) = 0$
    - $q'(x) = f'(x_0) + f''(x_1 - x_0)$
    - $\implies x_1$ solves $f'(x_0) + f''(x_1 - x_0)$
    - $\implies x_1 = x_0 - \frac{f'(x_0)}{f''(x_0)}$
  (c) Form a Taylor expansion around x_1 and repeat, from Step (b) to find $x_2, x_3, ...x_k$
  (d) Repeat until convergence: $|x_{k+1} - x_k| < \varepsilon$
- Convergence:
  - Rate of convergence is quadratic
- Can sensitive to initial guess
- Often the method of choice it feasible for your problem

4. The method of steepest descent (a gradient based method)

- Idea: The gradient of the function points in the direction of ascent in x
- To minimize, go in the opposite direction
- Question - how far to go?
  - Going further might get you to solution more quickly
  - But larger steps might have you jump over solution and have to backtrack
- Algorithm:
  (a) For an initial guess, x_0 find the gradient, $\bigtriangledown f(x_0)$
  (b) Find x_1 as $x_1 = x_0 - \alpha_0 \bigtriangledown f(x_0)$
  (c) But, what is α_0 (or what should it be)?
    - Find optimal step, α_0 by solving:

$$\min_{\alpha} f(x_0 - \alpha \bigtriangledown f(x_0)) \tag{3}$$

    - This problem can be solved by Brent's Method, Golden Ratio Search, or other algorithm.
  (d) Repeat from Step (a) to find $x_2, x_3, ..., x_k$ until convergence
- Convergence:
  - Can converge very quickly for some problems
  - Slower for others
- Gradient Descent Method
  - Solving for the optimal α_k at each iteration can be costly
  - Thus we might not want to solve for the optimal step size, but rather just use a rule of thumb to chose the step size
  - This method is the Gradient Descent Method (as opposed to the Method of Steepest Descent)

Numerical Optimization in Multiple Dimensions

- Solving non-linear problems in multiple dimensions is hard
- There is not guarantee of convergence to the true global min in any solution method
- It's all about trade offs - and thus it's important to understand how each method works

1. The Nelder-Meade Method
   - This function is particularly useful for multidimensional functions that aren't differentiable
   - Algorithm:
     (a) Choose and initial simplex of $n + 1$ points (where n is the number of dimensions of x)
     (b) Order according to the values at the vertices: $f(x_1) \leq f(x_2) \leq f(x_3) \leq ... \leq f(x_n) \leq f(x_{n+1})$
     (c) Calculate the centroid of all points except x_{n+1}. Call this x_o
     (d) Compute the reflected point, $x_r = x_o + \alpha(x_o - x_{n+1})$, where $\alpha > 0$
       - If x_r is better than the second worst, but note better than the best (i.e., $f(x_1) \leq f(x_r) < f(x_n)$), then:
         * Replace the worst point, x_{n+1} with x_r and go to Step (b)
       - If x_r is the best point (i.e., $f(x_r) < f(x_1)$) then:
         * Compute the expanded point: $x_e = x_o + \gamma(x_r - x_o)$ with $\gamma > 1$
         * If $f(x_e) < f(x_r)$ then replace x_{n+1} with x_e and go to Step (b)
         * Else, replace x_{n+1} with x_r and go to (b)
       - If x_r is at least as bad as the second worst point (i.e., $f(x_r) \geq f(x_n)$):

    - ∗ Compute the contraction point: $x_c = x_o + \rho(x_{n+1} - x_o)$ with $0 < \rho \leq 0.5$
    - ∗ If $f(x_c) < f(x_{n+1})$, replace x_{n+1} with x_c and go back to Step (b)
  - If $f(x_r) > f(x_{n+1})$, then shrink the space where searching by computing a new simplex, replacing each point except x_1 with $x_i = x_1 + \sigma(x_i - x_1)$ (where $0 < \sigma < 1$) and go to Step (b)
- Notes:
  - Standard values for the reflection, expanding, contraction, and shrink coefficients are $\alpha = 1, \gamma = 2, \rho = 0.5, \sigma = 0.5$
  - The initial simplex is important. If it's too small, the algorithm can get stuck there.

2. Newton Method

- Like Newton method in one dimension, but not deal with vectors/matrices of first/second order conditions.
- Still find x_{k+1} by finding minimum of a degree-two Taylor approximation of $f(x)$. Thus:

$$x_{k+1} = x_k - (D^2 f(x_k))^{-1} Df(x_k)^T \tag{4}$$

- Algorithm is the same as one dimension otherwise - iterate on the above until convergence starting from some initial guess, x_0
- Potential issues:
  - May not converge (the initial guess is important here)
  - $D^2 f(x)$ may not be positive definite, and thus inverse may not exist
  - $(D^2 f(x))^{-1} Df(x)^T$ may be expensive, unstable, or otherwise difficult to compute
    - ∗ The curse of dimensionality applies here.
    - ∗ If the problem has n dimensions, then the Hessian has n^2 elements.
    - ∗ Thus to compute the Hessian at each iteration requires computing $n^2/2$ second derivatives.

3. Conjugate Gradient Method

- This method is midway between Newton method and Gradient Descent
  - Newton method gives faster convergence
  - Gradient descent is less expensive to compute per step
- It works well to find local minima when the function is approximately quadratic near the minimum
- Similar to the Method of Steepest Descent
- Algorithm:
  - (a) Begin at point x_0 and find the gradient at this point, $\bigtriangledown f(x_0)$
  - (b) Move in the opposite direction the gradient points (since it points uphill and you want to find the min), $\Delta x_0 = - \bigtriangledown f(x_0)$
    - How far to move? Solve this as solve for in Steepest Descent Method
    - Find α_0 as $\min_\alpha f(x_0 + \alpha \Delta x_0)$
  - (c) Find point move to as $x_1 = x_0 + \alpha_0 \Delta x_0$
  - (d) Calculate the steepest direction at x_1: $\Delta x_1 = - \bigtriangledown f(x_1)$
  - (e) Compute β_1
    - There are several ways to do this.
    - Some examples:
    - i. Fletcher-Reeves:

$$\beta_n^{FR} = \frac{\Delta x_n^T \Delta x_n}{\Delta x_{n-1}^T \Delta x_{n-1}} \tag{5}$$

ii. Fletcher-Reeves:

$$\beta_n^{FR} = \frac{\Delta x_n^T \Delta x_n}{\Delta x_{n-1}^T \Delta x_{n-1}} \tag{6}$$

iii. Polak-Ribière:

$$\beta_n^{PR} = \frac{\Delta x_n^T (\Delta x_n - \Delta x_{n-1})}{\Delta x_{n-1}^T \Delta x_{n-1}} \tag{7}$$

iv. Hestenes-Stiefel:

$$\beta_n^{HS} = \frac{\Delta x_n^T (\Delta x_n - \Delta x_{n-1})}{s_{n-1}^T (\Delta x_n - \Delta x_{n-1})} \tag{8}$$

v. Dai-Yuan:

$$\beta_n^{DY} = \frac{\Delta x_n^T \Delta x_n}{s_{n-1}^T (\Delta x_n - \Delta x_{n-1})} \tag{9}$$

(f) Update conjugate direction: $s_1 = \Delta x_1 + \beta_1 s_0$

(g) Solve for optimal step size, $\alpha_1 = arg\min_\alpha f(x_1 + \alpha s_1)$

(h) Update the position: $x_2 = x_1 + \alpha_1 s_1$

(i) Repeat Steps (d)-(h) until convergence

4. Powell's Method

- Useful because it does not require that the function be differentiable
- Essentially breaks up the solution to an n-dimensional problem into n single dimensional minimization problems at each step
- Uses solution at each step to see the n directions in which minimums are found in the next step

5. Gradient Descent

- See above - that description is generalized for multidimensional problems

6. BFGS

- The longer name is the Broyden-Fletcher-Goldfarb-Shanno Method
- This method is similar to Newton method, but rather than computing the Hessian at each iteration (which can be expensive), it only computes the Hessian initially and then approximates it after that
- Given this approximation, it has a slower rate of convergence than Newton method, but the cost at each iteration is lower.
- Algorithm:
  - Recall that Newton method had:

$$x_{k+1} = x_k - (D^2 f(x_k))^{-1} Df(x_k)^T \tag{10}$$

  - With BFGS, we have this exactly for the first iteration. i.e.,

$$x_1 = x_0 - (D^2 f(x_0))^{-1} Df(x_0)^T \tag{11}$$

  - But for all following iterations we have:

$$x_{k+1} = x_k - A_k^{-1} Df(x_k)^T \tag{12}$$

    * where

$$A_{k+1} = A_k + \frac{y_k y_k^T}{y_k^T s_k} - \frac{A_k s_k s_k^T A_k}{s_k^T A_k s_k} \tag{13}$$

    * $y_k = Df(x_{k+1})^T - Df(x_k)^T$

    * $s_k = x_{k+1} - x_k$
  - A nice property of this is the A_k is guaranteed to be positive definite

7. Gauss-Newton Method

- This is a special methods for nonlinear least squares problems, a form of Newton method
- e.g., consider a problem where you have a set of residual
  - The function you are minimizing is of the form

$$f(x) = \sum_{i=1}^{M} r_i(x)^2 \tag{14}$$

  - where each r_i is a smooth function
  - Also, be careful with the notation - x are the parameters, i denotes observations (data)
  - A function of this form can be solved more easily than a more general nonlinear function
  - Express the problem in matrix notation:

$$f(x) = r(x)^T r(x) \tag{15}$$

  - The derivative of $f(x)$ is $Df(x) = 2r(x)^T J(x)$
  - where $J(x)$ is the Jacobian matrix for r (if x has n dimensions, then J is $M \times n$)
  - $J(x)_{i,j} = \frac{\partial r_i}{\partial x_j}$
  - The Hessian is given by

$$D^2 f(x) = 2\left(J(x)^T J(x) + \underbrace{\sum_{i=1}^{M} r_i(x) \overbrace{D^2 r_i(x)}^{\text{Deriv of Jacobian}}}_{Q(x)} \right) \tag{16}$$

  - Newton method implies:

$$x_{k+1} = x_k - (J(x_k)^T J(x_k) + Q(x_k))^{-1} J(x_k)^T r(x_k) \tag{17}$$

  - If $r(x)$ is close to 0, then $Q(x) \simeq 0$ and so we can drop it and find

$$x_{k+1} = x_k - (J(x_k)^T J(x_k))^{-1} J(x_k)^T r(x_k) \tag{18}$$

  - We can iterate on this equation until convergence, having to compute the Jacobian, but not the Hessian at each step
- Gauss-Newton is standard for nonlinear least squares problems
- If $Q(x) \simeq 0$ you get close to quadratic convergence
- If $Q(x)$ far from 0, you might not get convergence at all (initial guesses are important)
- Also, if $Q(x)$ is large, $J^T J$ may not be positive definite or even invertible. To fix this, one can use the Levenberg-Marquart method
  - To do this, replace $J^T J$ with $J^T J + \mu I$ where $\mu > 0$ is a dampening factor

8. Simulated annealing (a non-gradient method (though one could use gradients with this))

- Simulated annealing is a heuristic - a rule of thumb that will hopefully help you find the solution, but that is not mathematically proven to do so.
- It's a probabilistic method to find a global maximum (or minimum)

- The name derives from the process of annealing in metallurgy, which is the heating and cooling of metals to share and strengthen them
- Algorithm:
  (a) Start at an initial point, x_0 and draw a random perturbation around this point. Let the point that is this perturbation from x_0 be called x_1.
  (b) Determine the "energy" at points x_1 and x_0, call these e' and e, respectively.
    – How energy is defined is loose, but it will be something that relates to the value or slope of the functions are a given point
    – Since you'll compute this often, it's helpful is it's something that is easy to compute
    – To fix ideas, just think of the energy of point x being $f(x)$
  (c) Compare the energy at these two points:
    i. If $e' > e$, move to x_1
    ii. If $e' < e$, *maybe* move to x_1
  (d) Accept new point at random if has lower energy
    – Probability of acceptance should depend on 3 things: e, e', temperature (T)
    – Temperate will start at a relatively high number and be lowered at each successive iteration through this algorithm
    – $Pr(e, e', T)$ should be decreasing in e, increasing in e', and decreasing in T
    – The first two conditions mean that high energy points are more likely to get selected
    – The last means that the probability of accepting a low energy point declines as the number of iterations increase
  (e) If x_1 is selected, move there and go back to Step (a) with x_1 as the initial point
  (f) Repeat until T is zero (i.e., a maximum number of iterations has been reached) or when $|x_{k+1} - x_k| < \varepsilon$, where k denotes the iteration.
- Benefits of simulated annealing:
  – More likely to find global max (min) and not get stuck in local max (min)
- Costs of simulated annealing:
  – Not guaranteed to find the optimum
  – Convergence may not be quick - depends on how quickly reduce temp, but if reduce temp more quickly then more likely to get stuck in local max (min)
- Notes:
  – Sometimes this algorithm is called basin-hopping (e.g. in SciPy)
  – This method is related to the Metroplis-Hastings algorithm

Summary

Some methods and when they might be most useful (from Humphreys, Jarvis, Evans (2017), Chapter 9):

1. If the dimension of the problem is not too big
   (a) If x_0 is close to $x*$
     i. If computing $(D^2 f(x))^{-1} Df(x)$ is cheap and accurate, use Newton method as it has fastest convergence
     ii. If computing $(D^2 f(x))^{-1} Df(x)$ is expensive or error-prone, then
       - If $f = r^T r$, use Gauss-Newton (possibly with Levenberg-Marquart Modification)
       - Otherwise, try BFGS
   (b) 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)
   (c) 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.