APPM 4600 Lab 6

Variants of Newton's method for nonlinear system

1 Overview

In this lab, you will investigate techniques different techniques for building Quasi-Newton methods. In class, we focused on creating methods that eliminate the need to invert a matrix at each iteration of Newton. The main part of this lab will be dedicated to creating techniques that allow you to approximate the Jacobian. You will build variants of Newton's method that uses this approximate Jacobian.

2 Before lab

A common technique for approximating derivatives is finite differences. In this before lab exercise, you will test two finite difference techniques for approximating the derivative of a function at a point.

Finite difference approximations approximate derivatives with function evaluations. You will notice that they closely align with the definition you know from calculus of the derivative.

Here are the two techniques for approximating $f'(s)$.

Forward Difference

$$f'(s) \sim \frac{f(s+h) - f(s)}{h}$$

Centered Difference

$$f'(s) \sim \frac{f(s+h) - f(s-h)}{2h}$$

Consider the function $f(x) = \cos(x)$ and the task of approximating the derivative at $x = \pi/2$. (I picked a point where you know the derivative on purpose but you can pick another location.)

1. Use the two difference formulas above to approximate the derivative with

$$h = 0.01 * 2. **(-numpy.arange(0, 10)).$$

   This halves the value of h each time.

2. Determine the order of each of the approximation techniques.

3 Lab Day: Additional Quasi-Newton methods

In this lab you will build 3 Quasi-Newton methods. The first one is a more sophisticated version of Lazy Newton ("Slacker Newton"). In the other two you will use your finite difference technique from the before lab exercise to construct approximates to the Jacobian.

3.1 Slacker Newton

The advantage of Lazy Newton is that you only have to evaluate the inverse of a matrix one time for all iterations. This comes at a cost of a greater likelihood of not converging. To remedy this, an option to compute a new inverse of a Jacobian when you suspect your Lazy Newton is not converging or converging too slowly. This is what we will call *Slacker Newton*. It is not completely lazy.

3.2 Exercises: Build Slacker Newton

1. Decide on a condition under which you will think you should recompute the Jacobian. This could involve the distance between iterates, norms of a particular vector (i.e. understanding what the Newton update is), etc.

2. Implement your version of Slacker Newton. You can start with the Lazy Newton code from class.

3. Consider the vector function $\mathbf{f}(\mathbf{x}) = \begin{bmatrix} 4x_1^2 + x_2^2 - 4 \\ x_1 + x_2 - \sin(x_1 - x_2) \end{bmatrix}$. This function has roots near $(1, 0)^T$ and $(-1, 0)^T$. Apply Lazy Newton's method with an initial guess of $\mathbf{x}_0 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$. Set the stopping criterion for $\epsilon = 10^{-10}$.

4. How does your Slacker Newton perform compared with your lab partner's version?

5. How does it perform for the example problem from class?

3.3 Exercises: Newton with an approximate Jacobian

1. In the Newton code from class replace the evaluation of the Jacobian with the evaluation of the approximate Jacobian using finite differences. (You can chose whichever version of finite differences you want.) Allow for different values of h to be passed in to the approximate Jacobian evaluator.

2. Apply your approximate Jacobian version of Newton to the problem in the previous section with different values of h_j = step size in the direction j. Start with $h_j = (10^{-7})|x_j|$. Then try $h_j = (10^{-3})|x_j|$. How does the performance of root finding method depend on h_j?

3. How does the method perform for the example problem from class?

3.4 Additional Exercise: Hybrid Newton

You can put the two ideas of the modified Newton's methods we designed in lab together to get some advantages. Of course you will inherit some of the disadvantages as well.

1. Create a new variant of a root finding methods by, starting with your Slacker Newton's Method and building the approximate Jacobian instead of the true Jacobian.

2. Next make is so that you start with $h_j = 10^{-3}|x_j|$ in computing the first approximate Jacobian. Then make it so that every time you need to recompute the approximate Jacobian, the constant in front of the $|x_j|$ in the evaluation of h_j is halved.

3. Apply this new method to the problem in the first set of exercises and the problem from class. How does it perform compared to the other methods?

4 Deliverables

Push your hybrid root finding method to Git and report your solutions to the questions in the exercise section on Canvas.