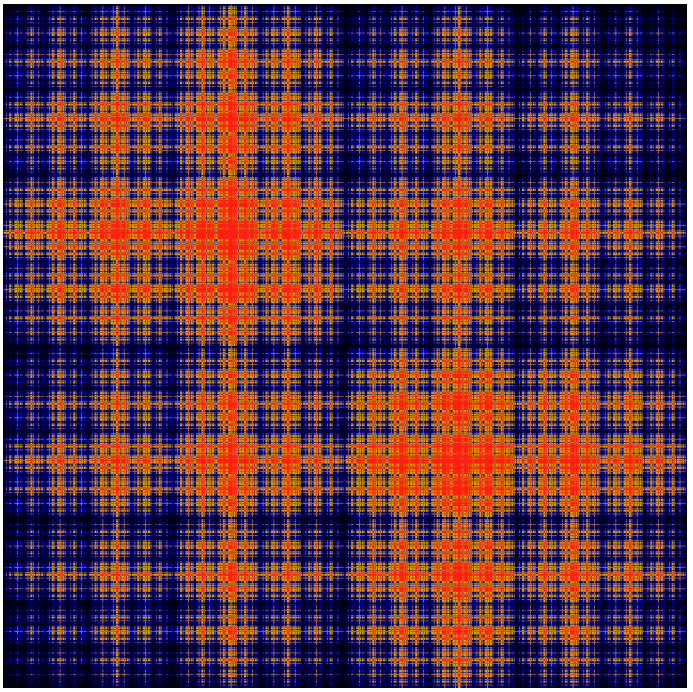
Applied Mathematics
and
Computing

Volume I



List of Contributors

J. Humpherys
Brigham Young University

J. Webb
Brigham Young University

R. Murray
Brigham Young University

J. West
University of Michigan

R. Grout
Brigham Young University

K. Finlinson
Brigham Young University

A. Zaitzeff
Brigham Young University

Preface

This lab manual is designed to accompany the textbook *Foundations of Applied Mathematics* by Dr. J. Humpherys.

Lab 23

Application: Newton's Method

Lesson Objective: *Understand Newton's Method*

One important technique in technical computing is Newton's method. The goal of Newton's method is to find x^* such that $f(x^*) = 0$. This method is especially important in optimization, where our goal is to find minima and maxima of functions. Newton's method is an iterative method (much like eigenvalue finders: remember how those provably have to be iterative?). Newton's method, in one dimension, is defined as follows:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

Essentially Newton's method approximates a function by its tangent line, and then uses the zero of the tangent line as the next guess for x_n.

Newton's method is powerful because of the speed of convergence. In many cases Newton's method converges to the actual root quadratically, meaning that the error term is squared at every iteration. This fast convergence makes it a very powerful algorithm.

Newton's method does suffer from the flaw that its convergence is dependent upon an initial guess. If the initial guess is not sufficiently close the convergence can be much slower, or may never occur. There are even certain pathological functions for which newton's method will never converge. However, these functions are very rare, and as a rule Newton's method converges very quickly.

Problem 1 Write a Newton's method function that runs whether or not the user inputs a derivative function. If the user gives a derivative function, use that. Otherwise estimate the derivative numerically within the Newton's method function. In python this can be done by defining the derivative function as a keyword argument.

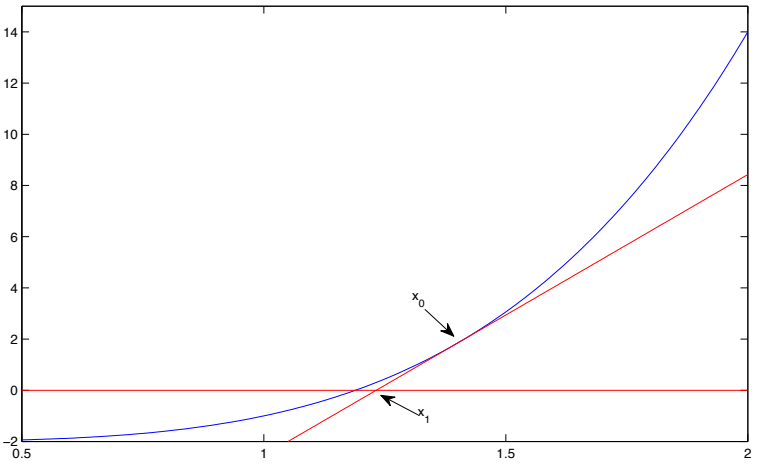


Figure 23.1: An illustration of how one iteration of Newton's method works

Your function definition will look something like this `def new_newton(f, x0, df=None, tol=0.001)`

Compare the performance of Newton's method when you input the derivative and when you don't. How well does each converge? Which runs faster? Try the following functions:

- $cos(x)$
- $sin(1/x) * x^2$
- $(sin(x)/x) - x$

Problem 2 Test your newton's method function on $x^{1/3}$. Use random starting points around zero. What do you see? Prove that for any non-zero starting point that newton's method will diverge.

Problem 3 A basin of attraction can be loosely defined as a set that will eventually converge to a specific root. Pick random points on the interval $[-2, 2]$ as starting

points and apply newton's method to the function $x^3 - 2x + 1/2$. Display the basins of attraction for this particular function. What do you observe?

Problem 4 Extend your Newton's method even further so that it will work on systems of equations. Suppose that $F : \mathbb{R}^n \to \mathbb{R}^n$. The relevant equation is

$$x_{i+1} = x_i - J^{-1}F(x_i)$$

Note that you **should not** calculate the inverse Jacobian. `sp.solve(A,b)` gives you the solution x to the equation $Ax = b$. Use this fact to calculate $J^{-1}F$ from J and F. You should be able to make this function work whether or not the user inputs a Jacobian. This also means that you will have to implement your own `jacobian` function.