Jack Parkinson
October 5th 2022

Phys 410 Homework 1 Write Up

Simply run ‘main.m’ in the zip folder to get the problem1 and problem2 roots

Problem 1:

Introduction:
This question was asked from an implementation of a hybrid function that uses a combination of Bisection and Newton’s Method to find the root of a function of the form f(x). Bisection is a root finding method applied to a continuous function to approximate the root x-coordinate. The method works by initially specifying a maximum x-Coordinate and minimum x-Coordinate and then calculates the midpoint of this function. From here, the bisection performs iterative steps to converge to a root with a given tolerance. The specifics of bisection will be explained in the methods sections. Newton's Method is also a root finding algorithm that also approximates the root of a function, but uses linear approximation with the derivative of the function divided by the function evaluated as a point for the slope. The specifics of Newton’s Method will be explained in the methods sections

Methods:
The hybrid function takes in 5 parameters: hybrid(f, dfdx, xmin, xmax, tol1, tol2):
- f: a function whose root we are trying to find
- dfdx: the derivative function of the function whose root we are trying to find
- xmin: the initial bracket minimum
- xmax : the initial bracket maximum
- tol1: the relative convergence criterion tolerance for bisection
- tol2: the relative convergence criterion tolerance for Newton’s Method

The hybrid function works by first calling the bisection method to the defined tol1, then passing the approximate root from bisection to the Newton’s Method function as demonstrated below in the simple pseudocode:

```
hybrid(f, dfdx, xmin, xmax, tol1, tol2):
   bival = bisection(f, xmin, xmax, tol1)
   root = newtowns(f, dfdx, bival, tol2)
```

As mentioned in the introduction, the bisection method iteratively finds the x-coordinate midpoint of xmin and xmax and then uses the following criterion to test if the function has converged:

`NotConverged = abs((xmin-xmax)/(xmid)) > tol1`

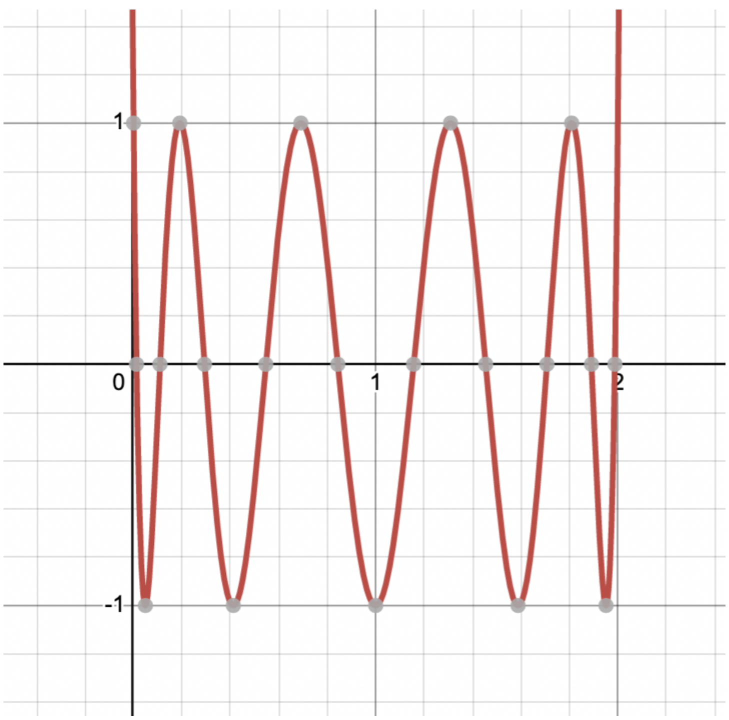
If the statement IsConverged is false the bisection iterative loop (a while loop) is terminated and returns the final xmid value. From here, the newton's method is called with: `newtowns(f, dfdx, bival, tol2)`. The one dimensional Newton's methods works on the simple linear approximation: $x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$, where in the context of my newton's algorithm x_{n+1} is x_new, x_n is x_new, $f(x_n)$ is f(x_old) and $f'(x_n)$ is dfdx(x_old). From here we can iteratively perform the linear approximation until the clause notConverged is false where:

```
notConverged = tol2 < (x_new - x_old)/(x_new);
```

Also, choosing appropriate parameters of: f, dfdx, xmin, xmax, tol1, tol2, is important to producing accurate roots. For example if we take the function:

$$f(x) = 512x^{10} - 5120x^9 + 21760x^8 - 51200x^7 + 72800x^6 - 64064x^5 + 34320x^4 - 10560x^3 + 1650x^2 - 100x + 1$$

We can see from the graph below that there are 10 roots:



From this graph we can construct the 10 xmin and xmax intervals with the knowledge that $f(max)f(min) < 0$:

root1: `hybrid(f, dfdx, xmin = 0, xmax = 0.1, tol1, tol2);`

root2: `hybrid(f, dfdx, xmin = 0.1, xmax = 0.2, tol1, tol2);`
root3: `hybrid(f, dfdx, xmin = 0.2, xmax = 0.4, tol1, tol2);`
etc

The last thing to mention is choosing the tolerance: the tolerance1 should be enough so that the bisection function can calculate a reasonable x-value for the root without being too exhaustive. With this in mind as well as the fact that I needed 12 digits of accuracy on the root I chose tol1 = 0.001. For choosing tol2, since the accuracy needed to be on the order of 12 digits I chose

$$tol2 = 10^{-12}$$

Results:
By running hybrid for 10 different xmin and xmax brackets I was able to determine the following 10 x-coordinates of the roots of the function

$$f(x) = 512x^{10} - 5120x^9 + 21760x^8 - 51200x^7 + 72800x^6 - 64064x^5 + 34320x^4 - 10560x^3 + 1650x^2 - 100x + 1$$

Root 1: 0.012311659404862
Root 2: 0.108993472390755
Root 3: 0.292893218813450
Root 4: 0.546009500260435
Root 5: 0.843565534959457
Root 6: 1.156434468151233
Root 7: 1.453990525373039
Root 8: 1.707106781180045
Root 9: 1.891006859887581
Root 10: 1.987695325215235

Conclusion:
I was able to verify the results above with wolfram alpha's calculation of the roots, and saw the hybrid function did calculate 12 digits of accuracy.

Problem 2:

Introduction:

This problem utilizes an n-dimensional Newton's method to calculate a single root. Newton's multidimensional method uses the Jacobian matrix on a given vector of functions to try to find the root of the given vector of functions. The multidimensional Newton Function can be written as: $x^{n+1} = x^{n} - inv(J(x^{n})) * f(x^{n})$, where $inv(J(x^{n}))$ is the inverse of the $n \times n$ jacobian matrix of the n-dimensional vector x. This form will be iterated until the tolerance (talked in more detail in the method section below) is reached at which point the function will return a vector of x-coordinates of the root.

Methods:

The newtond function has 4 inputs:

- f: a function that implements an inline nonlinear system of equations. The function is of the form f(x) where x is a length-n vector, and % which returns length-m column vector.
- jac: Function which is of the form jac(x) where x is a length-d vector, and which returns the d x d matrix of Jacobian matrix elements.
- x0: Initial estimate for iteration (length-d column vector).
- tol: Convergence criterion: routine returns when relative magnitude of update from iteration to iteration is <= tol.

From here we implement the iterative Newton D method described in the introduction:

$$x^{n+1} = x^{n} - inv(J(x^{n})) * f(x^{n})$$

The vector of functions we are finding the root of is formatted as:

```
f = [(x^2 + y^4 + z^6 -2); (cos(x*y*z^2)- (x + y + z)) ;  (y^2 + z^3 - (x + y - z)^2)]
```

And the jacobian matrix is formatted as:

```
Jac = [2*x, 4*y^3, 6*z^5; (-y*z^2*sin(x*y*z^2)-1), (-x*z^2*sin(x*y*z^2)-1), (-2*x*y*z*sin(x*y*z^2)-1); (2*z - 2*y - 2*x), (-2*x + 2*z), (3*z^2 - 2*z + 2*x + 2*y);].
```

A last note: The formula used to calculate the tolerance of the newton's method is:

$$\left\|\delta \underline{x}^{(n)}\right\|_2 = \sqrt{\frac{\sum_{i=1}^{d}\left(\delta x_i^{(n)}\right)^2}{d}} \leq \epsilon$$

Where δx = x_old - x_new.

Also, I chose the tolerance of tolerance = 10^{-12} because I wanted 12 digits of accuracy

Results:

On running the newtond method with the described function, jacobian matrix, tolerance described in the methods in addition to the suggested x0 =(-1, 0.75, 1.5) by the assignment my function returned a vector of roots to be (x,y,z) = (-0.577705133337193, 0.447447204972240, 1.084412371724927)

Conclusion:
I was able to obtain the roots of a vector of n-dimensional functions to at least 12 digits of accuracy. I verified that the accuracy exceeded 12 digits by plugging these values back into the vector of functions which returned an ans of:
1.0e-15 *

```
                 0
  -0.111022302462516
   0.444089209850063
```

which signifies that I have converged onto roots with at least 12 digits of accuracy.