Math 4610 Tasksheet 3 - Kaden Taylor A02257212

Task1: Rewrite of Code in C.

Here is the code for Fixed point C:

```
double fixedPoint(double (*g)(double), double x0, double tol, int maxIter)
{
  double error = 100;
  double x1 = 0.0;
  for(int i=0; i<maxIter && error>tol; i++)
    {
      x1 = g(x0);
      error = abs(x1 - x0);
      x0 = x1;
    }

  // printf("\nroot value = %f\n", x0);
  return x0;
}
```

Here is the code for Bisect:

```
double bisect(double (*f)(double), double a, double b, double tol)
{
  //
  // set up storage for the work to be done
  // ------------------------------------------------------------
  //
  double fa = f(a);
  double fb = f(b);
  //
  // test the endpoints for a root in the interval
  // ------------------------------------------------------------
  //
  if(fa*fb >= 0.0)
    {
      //printf("There may not be a root in [a,b]: f(a)*f(b) = %e", fa*fb);
      exit(-1);
    }
  double c;
  double fc;
  //
  //compute the number of
  int k = ( (int) ( log(tol) - log(b-a) ) / log(0.5) + 1);
```

```
  //
  // do the iterations needed to get a close enough approximation to a root
  //--------------------------------------------------------------------------
  //
  for(int i=0; i<k; i++)
    {
      c = 0.5 * (a + b);
      fc = f(c);
      if(fa*fc < 0.0)
        {
          b = c;
          fb = fc;
        }
      else
        {
          a = c;
          fa = fc;
        }
    }
  //
  // return the computed approximation of the root
  //--------------------------------------------------------------------------
  //
  //printf("\nroot value = %f\n", c);
  return c;
  //
}
```

Here is the code for newton in c:

```
double newton(double (*f)(double), double (*df)(double), double x0, double tol)
{

  if ( fabs(f(x0)) < tol )
    {
      return x0;
    }
  else
    {
      return newton(f, df, x0 - f(x0)/df(x0), tol);
    }

}
```

Here is the code for secant in c:

```
double secant(double (*f)(double), double x0, double x1, double tol)
{
  //printf("Iteration, x0: %f, x1: %f,error: %f\n", x0, x1, fabs(f(x1)) );

  if ( fabs(f(x1)) < tol )
    {
```

```
      return x1;
    }
  else
    {
      return secant(f, x1, ((x1-x0) * f(x0)) / (f(x1) - f(x0)), tol);
    }
}
```

Here is the code for hybrid bisect newton in c:

```
double hnewton(double (*f)(double), double (*df)(double), double a, double b, double tol, int maxIter)
{
  double error = 10.0*tol;
  double iteration =0;
  double x0 = 0.5*(a+b);

  while (error > tol && iteration < maxIter)
    {
      double x1 = x0 - f(x0)/df(x0);
      double newton_error = fabs(x1 - x0);
      if (newton_error > error)
        {
          double fa = f(a);
          double fb = f(b);
          for (int i = 0; i < 4; i++)
            {
              double c = 0.5*(a+b);
              double fc = f(c);
              if (fa*fc < 0)
                {
                  b = c;
                  fb = fc;
                }
              else
                {
                  a = c;
                  fa = fc;
                }
            }
          error = fabs(b-a);
          x0 = 0.5*(a+b);
        }
      else
        {
          x0 = x1;
          error = newton_error;
        }
      iteration = iteration + 1;
    }
  return x0;
}
```

Here is the hybrid bisect secant in c:

```
double hsecant(double (*f)(double), double a, double b, double tol, int maxIter)
{
  double error = 10.0*tol;
  int iteration = 0;
  double x0 = .49*(a+b);
  double x1 = .51*(a+b);
  while (error > tol && iteration < maxIter)
    {
      double x2 = x0 - ((x1-x0)*f(x0)) / (f(x1) - f(x0));
      double secant_error = fabs(x2 - x0);
      if (secant_error > error)
        {
          double fa = f(a);
          double fb = f(b);
          for (int i = 0; i < 4; i++)
            {
              double c = 0.5*(a+b);
              double fc = f(c);
              if (fa*fc < 0)
                {
                  b = c;
                  fb = fc;
                }
              else
                {
                  a = c;
                  fa = fc;
                }
            }
          error = fabs(b-a);
          x0 = .49*(a+b);
          x1 = .51*(a+b);
        }
      else
        {
          x0 = x1;
          x1 = x2;
          error = secant_error;
        }
      iteration = iteration + 1;
    }
  return x1;
}
```

There are also some .h header files that can be seen in my repository.

Task 2: Passing a Function to a Function.

The functions calls, passing functions, and printing is handled in my testRootFinding.cc

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#include "fixedPoint.h"
#include "bisect.h"
#include "newton.h"
#include "secant.h"
#include "hybrid_newton.h"
#include "hybrid_secant.h"

double gval(double);
double fval(double);
double dfval(double);


int main()
{
  double a = -2.0;
  double b = 3.0;
  double x0 = -.001;
  double x1 = .001;
  double tol = .00000001;
  double maxIter = 10;

  double fixedPointVal = fixedPoint(gval, x0, tol, maxIter);
  double bisectVal = bisect(fval, a, b, tol);
  double newtonVal = newton(fval, dfval, x0, tol);
  double secantVal = secant(fval, x0, x1, tol);
  double hnewtonVal = hnewton(fval, dfval, a, b, tol, maxIter);
  double hsecantVal = hsecant(fval, a, b, tol, maxIter);


  printf("Fixed Point Root: %f\n", fixedPointVal);
  printf("Bisect Root: %f\n", bisectVal);
  printf("Newton Root: %f\n", newtonVal);
  printf("Secant Root: %f\n", secantVal);
  printf("Hybird Newton Root: %f\n", hnewtonVal);
  printf("Hybrid Secant Root: %f\n", hsecantVal);
}

double gval (double xval)
{
  double gval = xval - (xval*exp(-xval));
  return gval;
}

double fval (double xval)
{
  double fval = xval * exp(-xval);
  //printf("xval = %f, fval = %f\n", xval, fval);
  return fval;
}

double dfval(double xval)
{
```

```
  double dfval = exp(-xval) - xval*exp(-xval);
  // printf("xval = %f, fval = %f\n", xval, dfval);

  return dfval;
}
```

Task 3: Complete the test

When the `testRootFinding.cc` is complied and ran using:

```
% gcc -o testRootFinding testRootFinding.c *.o
% gcc ./testRootFinding
```

These results are obtained:

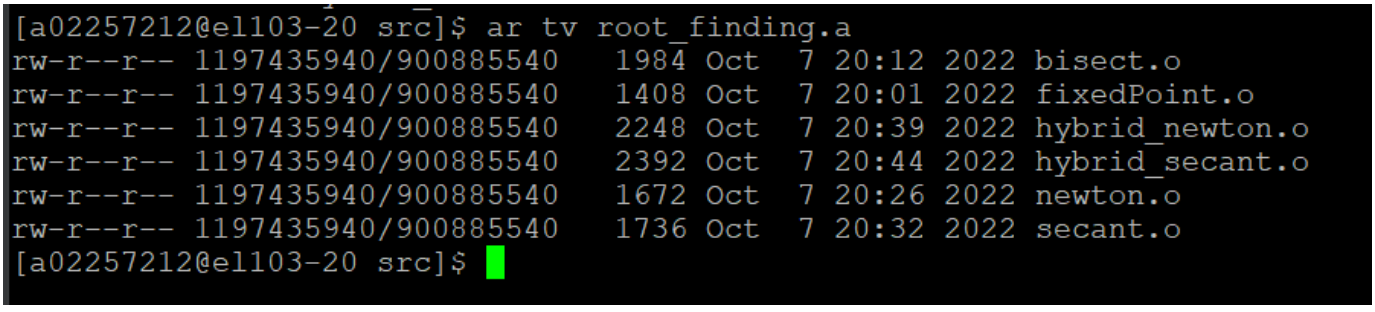
```
Fixed Point Root: 0.000001
Bisect Root: 0.000000
Newton Root: -0.000000
Secant Root: 0.000000
Hybird Newton Root: -0.000000
Hybrid Secant Root: -0.000000
```

Task 4: Shared Library Creation

I created the shared library and when the `%ar tv root_finding.a` command is run, the following results are obtained:

```
rw-r--r-- 1197435940/900885540   1984 Oct  7 20:12 2022 bisect.o
rw-r--r-- 1197435940/900885540   1408 Oct  7 20:01 2022 fixedPoint.o
rw-r--r-- 1197435940/900885540   2248 Oct  7 20:39 2022 hybrid_newton.o
rw-r--r-- 1197435940/900885540   2392 Oct  7 20:44 2022 hybrid_secant.o
rw-r--r-- 1197435940/900885540   1672 Oct  7 20:26 2022 newton.o
rw-r--r-- 1197435940/900885540   1736 Oct  7 20:32 2022 secant.o
```

A screen shot can be seen on my repository in the hw3/doc section and here:



Task 5: Test the shared library

These commands in the terminal were run:

```
$ gcc -o testRootFinding testRootFinding.cc root_finding.a  -lm
$ ./testRootFinding
```

Here is the output obtained:

```
Fixed Point Root: 0.000001
Bisect Root: 0.000000
Newton Root: -0.000000
Secant Root: 0.000000
Hybird Newton Root: -0.000000
Hybrid Secant Root: -0.000000
```

A screen shot of this work in the terminal is in my repository under hw3/doc/ and here: