John Peddie EEE3096 Prac 2 PDDJOH001

Raspberry Pi: Cross Compiling

August 2018

This report is split into 3 parts as per the Practical Manual. Cross compilation was done using Windows and the Pi was accessed using Ubuntu Bash Shell for windows.

Part 1: Cross compiling:

```
C:\Users\johnp\Documents\00WORK\eee3096s\Practicals\prac 2\Roots1>arm-linux-gnueabihf-g++ -ggdb roots.c -o rootsCompiled
.exe
```

This was done using the GNU toolchain for windows.

Part 2: Testing:

In order to test the roots.c function, 4 separate polynomials were chosen, these are as follows:

$$x^3 + 3x - 5 = 0$$

$$3x^3 + 3x^2 - 5 = 0$$

$$3x^4 + 2x^2 - 1 = 0$$

$$2x^5 + 3x^4 + 5x^3 - 13x^2 - 5x + 2 = 0$$

These were solved using the roots.c function. The algorithms and their outputs are as follows:

Roots1:

Input:

```
#define F_STRING "x^3 + 3*x - 5 = 0"
// specify two approximations. low and high value to search
#define DEFAULT_X1  -10
#define DEFAULT_X2  10
// specify the number of iterations
#define ITERATIONS 10
return(pow(x,3)+3*x-5);//This return the value of the function
```

Output:

```
pi@raspberrypi:~/EEE3096/eee3096Pracs/prac 2/Roots1 $ ./rootsCompiled.exe
Clock resolution: 1 ns
Calculate bisection method in C
 Function: x^3 + 3*x - 5 = 0
The time it took is 0.000213
The approximation to the root is 1.152344
```

Roots2:

Input:

```
#define F_STRING "3*x^3 + 3*x^2 - 5 = 0"
// specify two approximations. low and high value to search
#define DEFAULT_X1  -10
#define DEFAULT_X2  10
// specify the number of iterations
#define ITERATIONS 10
return(3*(pow(x,3))+3*pow(x,2)-5);//This return the value of the function
```

Output:

```
pi@raspberrypi:~/EEE3096/eee3096Pracs/prac 2/Roots2 $ ./roots2Compiled.exe
Clock resolution: 1 ns
Calculate bisection method in C
 Function: 3*x^3 + 3*x^2 - 5 = 0
The time it took is 0.000213
The approximation to the root is 0.917969
```

Roots3:

Input:

```
#define F_STRING "3*x^4 + 2*x^2 - 1 = 0"
// specify two approximations. low and high value to search
#define DEFAULT_X1  -10
#define DEFAULT_X2  10
// specify the number of iterations
#define ITERATIONS 15
return(3*(pow(x,4))+2*pow(x,2)-1);//This return the value of the function
```

Output:

```
pi@raspberrypi:~/EEE3096/eee3096Pracs/prac 2/Roots3 $ ./roots3Compiled.exe
Clock resolution: 1 ns
Calculate bisection method in C
 Function: 3*x^4 + 2*x^2 - 1 = 0
The time it took is 0.000216
The approximation to the root is -0.566406
```

Roots4:

Input:

```
#define F_STRING "2*x^5 + 3*x^4 + 5*x^3 -13*x^2 - 5*x +2 = 0"
// specify two approximations. low and high value to search
#define DEFAULT_X1  -10
#define DEFAULT_X2  10
// specify the number of iterations
#define ITERATIONS 21
return(2*pow(x,5) + 3*pow(x,4) + 5*pow(x,3) - 13*pow(x,2) - 5*x +2);//This return the value of the function
```

(The string of the function is erroneous, however this has no effect on the numerical output.)

Output:

```
pi@raspberrypi:~/EEE3096/eee3096Pracs/prac 2/Roots4 $ ./roots4Compiled.exe
Clock resolution: 1 ns
Calculate bisection method in C
 Function: 2*x^5 + 3*x^4 + 5*x*3 -13*x^2 - 5*x +2 = 0
The time it took is 0.000273
The approximation to the root is -0.566406
```

These outputs also contain timing data which is important for the performance testing part of this report.

Once these outputs were gathered, they were compared to the roots generated by the roots method of the Numpy library in python. Only the real roots are important for this case, and have therefore been highlighted:

```
In [12]: from numpy.polynomial import Polynomial as P


         #x^3 + 3*x - 5 = 0
         print("Roots of: x^3 + 3*x - 5 = 0")
         p = P([-5, 3, 0, 1])
         print(p.roots())

         #3*x^3 + 3*x^2 - 5 = 0
         print("Roots of: 3*x^3 + 3*x^2 - 5 = 0")
         p1 = P([-5, 0, 3, 3])
         print(p1.roots())

         #3*x^4 + 2*x^2 - 1 = 0
         print("Roots of: 3*x^4 + 2*x^2 - 1 = 0")
         p2 = P([-1, 0, 2, 0, 3])
         print(p2.roots())

         #2*x^5 + 3*x^4 + 5*x*3 -13*x^2 - 5*x +2 = 0
         print("Roots of: 2*x^5 + 3*x^4 + 5*x*3 -13*x^2 - 5*x +2 = 0")
         p3 = P([+2, -5, -13, 5, 3, 2])
         print(p3.roots())

         Roots of: x^3 + 3*x - 5 = 0
         [-0.57708575-1.99977096j -0.57708575+1.99977096j  1.15417150+0.j        ]
         Roots of: 3*x^3 + 3*x^2 - 5 = 0
         [-0.96470911-0.92874797j -0.96470911+0.92874797j  0.92941823+0.j        ]
         Roots of: 3*x^4 + 2*x^2 - 1 = 0
         [-0.57735027+0.j  0.00000000-1.j  0.00000000+1.j  0.57735027+0.j]
         Roots of: 2*x^5 + 3*x^4 + 5*x*3 -13*x^2 - 5*x +2 = 0
         [-1.23898497-1.98213122j -1.23898497+1.98213122j -0.56227614+0.j
           0.25282844+0.j          1.28741764+0.j        ]
```

As it can be seen, the accuracy of the roots.c program is questionable. The method of determining roots has a relatively low accuracy. This is especially evident in the higher order polynomials. This is due to the iterations used in the program. A higher number of iterations will yield a more accurate result however it would take longer and use more of the limited resources the Raspberry Pi has to offer.

Part 3: Performance Testing:

Performance testing was done using the included Timer.ccp library and the method's it included. The Tic() and Toc() methods essentially started and stopped a timer, the Toc() method returning the timer's time. By placing Tic() at the start of the method and Toc() at the end, accurate timing information can be obtained. This is demonstrated in a screenshot of the modified roots.c file at the end of this section.

The results obtained can be seen in the outputs of the functions listed above. Summarized:

Function:	Time taken to find roots: (Seconds)
$x^3 + 3x - 5 = 0$	0.000213
$3x^3 + 3x^2 - 5 = 0$	0.000213
$3x^4 + 2x^2 - 1 = 0$	0.000216
$2x^5 + 3x^4 + 5x^3 - 13x^2 - 5x + 2 = 0$	0.000273

As can be seen from the results, it appears an increase in terms in the polynomial has a larger affect on the time taken for the program to run than the increase in order. This is also affected by an increase in iterations. It is evident that an increase in iterations does not have a linear effect on the time taken to run the function.

Snippet of Roots.c with the Tic() and Toc() functions evident:

```c
    tic(); //Begins the timer
    int iter = ITERATIONS; // iteration number
    // Print intro information
    printf("Calculate bisection method in C\n Function: ");
    printf(F_STRING "\n");
    // Get the two approximation values
    double x0 = DEFAULT_X1,
           x1 = DEFAULT_X2;
    if (INTERACTIVE) {
            printf("Enter the first approximation to the root\n");
            scanf("%lf",&x0);
            printf("Enter the second approximation to the root\n");
            scanf("%lf",&x1);
            printf("Enter the number of iterations you want to perform\n");
            scanf("%d",&iter);
    }

    // Variables to be used during the iterations
    int ctr=1; // current iteration
    double l1=x0;
    double l2=x1;
    double r,f1,f2,f3;

    // ------------------- Initial checks ----------------------------
    // Check if initail approximations are actually the roots themselves
    if(F(l1)==0) r=l1;
    else if(F(l2)==0) r=l2;
    else {
     // This implementas the bisection algorithm
    while(ctr <= iter) {
      f1=F(l1);
      r=(l1+l2)/2.0;
      f2=F(r);
      f3=F(l2);
      if(f2==0) {
          r=f2;
          break;
        }
      #if(DEBUG_LEVEL>=1)
        printf("The root after %d iteration is %lf\n",ctr,r);
      #endif
      if(f1*f2<0) l2=r;
      else if (f2*f3<0) l1=r;
      ctr++;
      }
    }
    // Display the appriximated root found
    printf("The time it took is %lf\n", toc()); //ends the timer and outputs the time
    printf("The approximation to the root is %lf\n",r);
}
```

The Repository for the code and access of all functions can be found at the following link:
https://github.com/YGDFLAZ/eee3096Pracs