

Assignment Four

MT4112

Deadline Sunday 8th January 11:00pm

All work **must** be done in the sub-directory ‘assign4’ off your home directory. When the assignment deadline has passed you will not be able to write to this directory anymore. The work will then be copied from your ‘assign4’ directory and marked. **REMEMBER** follow the directions given in the question exactly when writing your code.

Question One

The ‘*bisection method*’ is a simple root finding method. The thinking behind the bisection method is based on the property that a function’s value, $f(x)$, will change sign when it crosses over the line $f(x) = 0$. Suppose $f(x)$ is a continuous function defined on the interval $[a, b]$ and for simplicity assume that the interval $[a, b]$ contains only one root. The algorithm can be described as follows.

- Calculate new estimate of the root p where $p = \frac{1}{2}(a + b)$.
- The process checks if ‘ABS(a-b)/2 < tol’ or if the maximum number of iterations has been exceeded or if ‘ $f(p) == 0$ ’ (hit the root exactly).
- If all the above conditions are false then the iterative process continues else the process stops.
- The value p is then assigned to either a or b such that $f(a)f(b) < 0$. This forms a new interval $[a, b]$, half the size of the previous interval, that still contains the root.
- The above algorithm is repeated.

Write a Fortran 90 code to find a root using the bisection method **please follow all the guidelines below**.

- You **must** write compile etc. your code in the directory ‘assign4’.
- Create a main program file called ‘roots.f90’ and a module file ‘roots_mod.f90’ (Note the underscore character “_”).
- In your module file write a function to implement the iterative bisection algorithm. The function only returns on convergence or if the maximum number of iterations has been exceeded. It has the header

```
‘FUNCTION bisection(a,b,tol,maxiters,iters,conv,output)’
```

This function is called once from the main program unit and returns a ‘REAL’ variable holding the estimate of the root. The two ‘REAL’ variables ‘ a ’ and ‘ b ’ will hold the limits bounding the root. The ‘REAL’ value ‘tol’ is the prescribed tolerance. ‘maxiters’ an ‘INTEGER’ type that on entrance to the function holds the maximum number of iterations allowed. ‘iters’ is an ‘INTEGER’ type that returns to the main program the number of iterations taken. ‘conv’ is a one dimensional ‘LOGICAL’ array of two elements. On termination of the method the first element will hold ‘.TRUE.’ only if the method converged successfully and the second element will hold ‘.TRUE.’ only if the maximum number of iterations has been exceeded. ‘output’ is a logical type that on entry to ‘bisection’ if set to ‘.TRUE.’ the ‘bisection’ function will produce the output table to the screen else it outputs nothing. This allows the user of the code to decide if a table is to be printed to the screen by setting output in the main program unit ‘roots.f90’.

- In your module file write a function that has the header

```
‘FUNCTION end_loop(num1,num2,tol,maxiters,iters)’
```

that is called from within the function 'bisection'. It returns a one dimensional 'LOGICAL' array of two elements. It tests if the method has converged (element one is then set to '.TRUE.')

and if the max number of iterations has been exceeded (element two is then set to '.TRUE.'). The two 'REAL' variables 'num1' and 'num2' are the limits containing the root. 'maxiters' and 'iters' are as previously described.

- In the module file write a function with the header 'FUNCTION f(x)' that simply returns the value of the function you are trying to find a root for. It accepts as an argument the 'REAL' variable 'x'. This can be called from 'bisection' when required.
- Your main program 'roots.f90' should use the function 'bisection' to solve for the given functions, $f(x)$ below. No looping takes place in the main program unit. It should inform the user of successful convergence and the number of iterations taken. If the maximum number of iterations has been exceeded the user should also be informed. If the method converged it will also print the **final** estimate of the root followed by an empty (blank) line. To print a blank line you can simply feed and empty string to the print statement. 'PRINT*, ""'.

For each of the following three functions use your code to estimate the root that lies in the given initial interval. Use a 'tol' value of 0.0001 and 'maxiters' equal to 400.

- (1) $f(x)=\cos(x/20) + (\sin(x/20))^3$, with $a = -20$ & $b = 0$
- (2) $f(x)=2x^3 - x^2 + x - 1$, with $a = -1$ & $b = 1$
- (3) $f(x)=x^3 + 4x^2 - 10$, with $a = 1$ & $b = 2$

For each case use the 'output' logical data type created in your code to flag your 'bisection' function to print out a table as shown below in the example program output. That is to print at each iteration the iteration number 'iters', the boundaries of the current interval, 'a' and 'b', the current prediction p and $f(p)$, as in the example table below. NOTE the final root estimation and number of iterations taken are printed from the main program NOT from the 'bisection' function.

Iters	a	b	p	f(p)
1	1.000000	2.000000	1.500000	2.375000
2	1.000000	1.500000	1.250000	-1.796875
3	1.250000	1.500000	1.375000	0.162109
4	1.250000	1.375000	1.312500	-0.848389
5	1.312500	1.375000	1.343750	-0.350983
6	1.343750	1.375000	1.359375	-0.096409
7	1.359375	1.375000	1.367188	0.032356
8	1.359375	1.367188	1.363281	-0.032150
9	1.363281	1.367188	1.365234	0.000072
10	1.363281	1.365234	1.364258	-0.016047
11	1.364258	1.365234	1.364746	-0.007989
12	1.364746	1.365234	1.364990	-0.003960
13	1.364990	1.365234	1.365112	-0.001944

The root is estimated as : 1.365112
The number of iterations taken was 13

Output the three 'tables' your code generates to the **same** text file called 'q1.txt'. This is achieved when you run the code by redirecting the output from the display to the text file using the ">". If you

need to **append** results to an already existing text file you can use a different redirection operator “>>” instead of “>”.

If you write code to solve the first of the three mathematical functions, $f(x)$, above and call the executable ‘roots’ then instead of just typing ‘roots’ to run the code type,

```
roots > q1.txt
```

edit the code and recompile to solve for the second function then

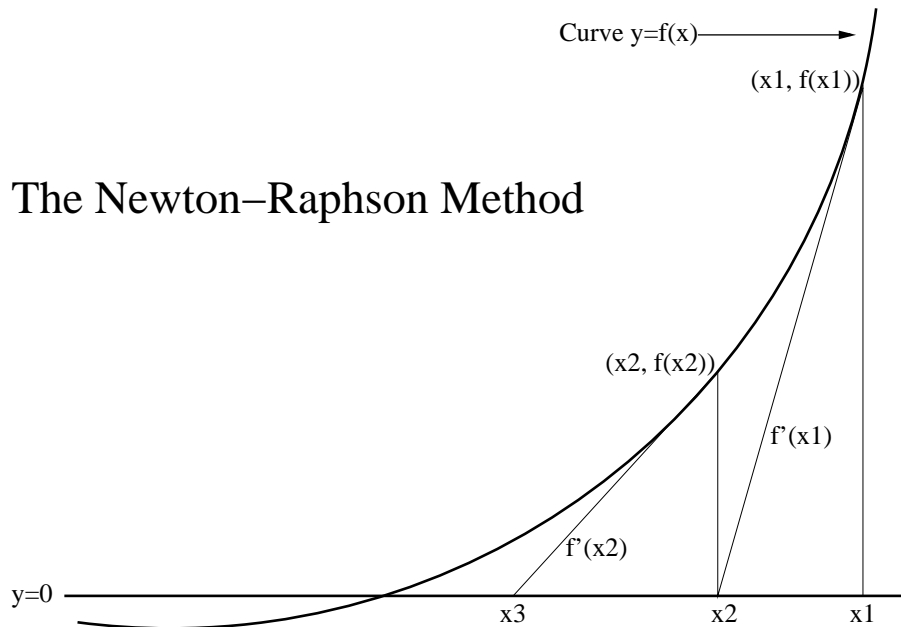
```
roots >> q1.txt
```

edit the code and recompile to solve for the third function then

```
roots >> q1.txt
```

Question Two

The Newton-Raphson method is another root finding method that typically converges more quickly than the bisection method. The diagram shows how this method works. The initial estimate is x_1 the tangent line at $(x_1, f(x_1))$ cuts the line $y = 0$ at x_2 . The point x_2 is the new more accurate approximation to the root. The process is repeated with new tangent line at $(x_2, f(x_2))$ cutting the line $y = 0$ at the new estimate x_3 , and so on The algorithm terminates when two successive estimates are less than a given tolerance or ‘maxiters’ has been exceeded.



The Newton–Raphson Method

Figure 1: The Newton-Raphson Method

- In your module file for question one, ‘roots_mod.f90’, add a new function to implement the Newton-Raphson method that returns the estimate of the root.

```
‘FUNCTION newt_raph(x,tol,maxiters,iters,conv,output)’
```

‘x’ is the initial estimate. The other arguments serve the same purpose as in ‘bisection’. The function ‘end_loop’ is also called from within ‘newt_raph’ and serves the same purpose.

- Write a function ‘ `FUNCTION df(x)` ’ that simply returns the derivative, $f'(x)$ at the given point x . Note you can analytically work out the derivative $f'(x)$ from the function $f(x)$ given and hard code it into the function ‘ `df(x)` ’.
- Test your code by finding the root of $f(x) = x^2 - 2$ with a starting value of $x = 12$ and redirect the results into a file ‘ `q2.txt` ’

When finished you should have the following files available for marking in your ‘ `assign4` ’ directory.

Main program : ‘ `roots.f90` ’

Module file : ‘ `roots_mod.f90` ’

Results : ‘ `q1.txt` ’

Results : ‘ `q2.txt` ’