MA305 – Lab #4
Finding Roots of an Equation
Due: Tuesday, 10/16/2018

Let's try to find the zero(s) of the function: $F(x) = x^3 + x^2 - 3x - 3$, i.e. the root(s) of $F(x) = 0$.

1. Graphical Method

a. First, of course, we ask: Are there any? Plot it to get an idea what the curve $y = F(x)$ looks like. Remember how? In another terminal type:
username:~$ `gnuplot`
`gnuplot> plot x**3+x**2-3*x-3`

It crosses the x-axis and looks very flat on it, ...many roots? (think: at most how many roots could there be for this F?) Notice the very big range on the y-axis which makes small variations invisible, so we can't really see much at this scale.

b. To resolve it better, let's plot it on a smaller interval, say, [-2,2]:
`gnuplot> set xrange [-2:2]`
`gnuplot> replot` (short cut: `CTRL p`)
An alternative way is:
`gnuplot> plot [-2:2] x**3+x**2-3*x-3`

It looks much better, doesn't it? It crosses the axis three times, so there are three real roots. Let's magnify the scale (zoom in) around the largest root (inside [1,2]), so try:
`gnuplot> set xrange [1:2]`
`gnuplot> replot`
Now it's better resolved, and we can zoom in further:
`gnuplot> set xrange [1.6:1.8]`
`gnuplot> replot`

c. Repeat the zooming in until you can estimate the root to three decimals, and write it down. This is the Graphical Method for finding roots of (simple) functions! Can you "guess" the exact value of the root ? Write it down.

2. Bisection Method (Part I)

a. Now that we know our function has a (single) root in [1,2], and it has opposite signs at $x = 1$ and $x = 2$, we can use the Bisection Method to locate the root as accurately as we want, say, with Tolerance 1.e-6, provided $F(x)$ is continuous on [1,2]. Is it? Justify your answer, mathematically!

b. Read the attached Python program *lab4.py* carefully to understand what's happening. As always, save it in a new directory: ~/`MA305/Lab4`.

```
$ mkdir ~/MA305/Lab4
$ cd ~/MA305/Lab4
$ mv ~/Downloads/lab4.py .
```
Read it over trying to catch misprints... Compile and run it (`python3 lab4.py`). Correct any errors... Fix the printing, if necessary (to print values in columns).

c. Set executable permission to the file `lab4.py` and run it with the following values:

(i) a, b, Tol: 1 2 1e-6

(ii) a, b, Tol: 1 2 1e-8

(iii) a, b, Tol: -4 4 1e-10

```
$ chmod u+x lab4.py
$ ./lab4.py
```

For each run observe the following.
How many bisections did it do? Does the root agree with your estimate from plotting? What is the effect of changing the tolerance (Tol)?

3. Bisection Method (Part II)

a. The bisection program *lab4.py* can be improved in various ways... One necessary improvement is to make it work also for the case:

$$F(a) > 0 > F(b),$$

so in fact for any (opposite) signs at the endpoints. You were supposed to figure out the algorithm ON PAPER at home..., and to have a complete bisection algorithm written ON PAPER, so you'd be ready... So, now incorporate your method in the code, as *lab4a.py*. Follow the steps:

b. NEVER mess up a working debugged code! ALWAYS make a copy of the file with a new name, and make the changes there. First, of course, we create a new dir (`mkdir ~/MA305/Lab4a`). Assuming the file *lab4a.py* is in ~/MA305/Lab4/ (where it belongs!), this can be done in various ways, depending on where you are.
From Lab4/: `cp lab4.py ../Lab4a/lab4a.py`
From Lab4a/: `cp ../Lab4/lab4.py lab4a.py`
From any dir:
`cp ~/MA305/Lab4/lab4.py ~/MA305/Lab4a/lab4a.py`
Try them all. Think what you are doing...

Now you have *lab4a.py* in ~/MA305/Lab4a/ which you can modify. So, go to ~/MA305/Lab4a/ and edit *lab4a.py* to make bisection work for any (opposite) signs at the endpoints.

c. EVERY new code must be debugged and validated. This requires that you first apply it to a problem for which you know the answer. So, choose a very simple curve (straight line!) with negative slope to debug the code on, e.g.

$$F(x) = 1 - 2x$$

Comment out the cubic and use this $F(x)$. Try it with a couple of different a, b, Tol=1.e-6.

d. Once the new case is working, make sure your code still solves the old case!!! Try it on a line with positive slope.

e. Finally, use your code to locate ALL THREE ROOTS of the cubic (as before), one root at a time of course... choosing [a,b] wisely. Now, explore the links about **gnuplot** on the course page. In particular, learn how to save the plot into a postscript file and convert it to a pdf file.

4. Repeated execution on new input data:

Sometimes we want to run our program repeatedly on new data automatically, e.g. to find all three roots of a cubic equation.

(i) Do:
`$ cp lab4a.py lab4b.py`

(ii) Modify the code `lab4b.py` so that after it finds one root, it goes back and reads user input data again, until it reads equal values for a and b, in which case it should exit. (to reduce the output comment out the line that prints at every iteration)

Test it on this data set.

```
-1.8  -1.5  1e-6   :a, b, Tol
-1.3  -0.5  1e-6   :a, b, Tol
 1.0   2.0  1e-6   :a, b, Tol
  0     0   1e-6   :a=b=0 to terminate!
```

If `a=b=0`, terminate the loop. This should find all three root of the cubic in a single run!

5. *Submit your work (the cleaned up typescript showing your code* ***lab4b.py*** *and output from running on the* ***dat4.txt*** *data file, after appropriate editing , as usual...) through your* **course Canvas**.

Below is the first version of Python code for the bisection method, roughly as we developed it in class. NOTE that this code assumes that $F(a) < 0 < F(b)$, needs improvement!

```
#!/usr/bin/env python3

"""
================================================================================
MA305 - Lab 4: your name - date
Purpose: ... briefly describe what it does ...
! NOTE: this version assumes F(a)<0<F(b), needs improvement !
================================================================================
"""

# Define function here:
def F(x):
    value = -3. + x * ( -3. + x * ( 1. + x ) ) #=x**3+x**2-3.0*x-3.0
    #value=-1+2*x
    return value

# Get input from user and change the string input into number types:
a,b,Tol=input(' Enter the values of a, b, Tol:').split()
a=float(a)
b=float(b)
Tol=float(Tol)

Nmax=1000

# Print on screen:
print('\n Searching for root in [{0: 5.2f}, {1: 5.2f}], with TOL={2:0.8g}'.
    format(a,b,Tol))
print('--------------------------------------------------------------------')

#.............. Find values at endpoints:
Fa = F(a)
Fb = F(b)

#.............. Make sure Bisection is applicable, else exit
if( Fa * Fb > 0 ):
    print(' F(a)*F(b) > 0, Bisection NOT applicable !')
    print(' try different a, b ? ...exiting')
    exit

#.............. Initialization:
print('  n                r            F(r)           error')
n = 0 # initialize the counter
error = abs(b-a) # initialize the error

#---------------- Begin Bisection Loop ----------------
while error > Tol:
    n += 1
```

```
    r = ( a + b ) / 2
    Fr = F(r)
    if( Fr == 0 ):
        print('Got lucky! root =',r,' F(r)=',Fr)
        print(' in',n,' tries',' error <=',error/2)
        break
    elif( Fr < 0 ):
        a = r
        Fa = Fr
    else:
        b = r
        Fb = Fr
    error = error/2 # or: = b-a
    print('{0: 3d} {1: 15.8g} {2: 15.8g} {3: 15.8g}'.format(n, r, Fr, error))
    # print this iterate
    if( n > Nmax ):
        print('....NOT done, iterations exhausted:')
        print(' n=',n,' > Nmax=',Nmax)
        print('try shorter [a,b], or bigger nMAX; Exiting...')
        break
#---------------- end of bisection loop ----------------
print('-------------------------------------------------------------')
print(' Done! best estimate for the root:{0: 21.14g}'.format(r))
```