PH3105 recapitulation problems

Q 1) Consider the following function

```
def f(x):
    s = 0.0
    l = [1,2,3,4]
    for c in l:
        s = s*x + c
    return s
```

Which function of x is returned by $f(x)$?

Write a function that will take a list of the coefficients $a_0, a_1, a_2, \ldots, a_n$ as one argument and the value of a variable x as another and return the value of the polynomial

$$a_0 + a_1 x + a_2 x^2 + \ldots + a_n x^n$$

Write a program **polynomial.py** that will use this function print out a table of the values of the polynomial

$$x^3 - 7x^2 + 5$$

for the values 0, 1, 2, ... 10 for x. Why do you think that the code written in this problem has an advantage over simply returning x**3 -7*x**2+5 (direct calculation)?

The bisection method is a popular way of solving an equation $f(x) = 0$, where the function $f(x)$ is continuous. We first identify a "bracketing interval" (a, b) such that $f(a)$ and $f(b)$ are of opposite signs ($i.e.$ $f(a) f(b) < 0$ - this ensures that there is at least one root of $f(x) = 0$ in this interval. We next bisect the interval at $c = \frac{a+b}{2}$. If now $f(c)$ has the same sign as $f(a)$, the root must lie between c and b - if, on the other hand the sign of $f(c)$ matches that of $f(b)$- the root must lie between a and c. Thus, we have halved the size of the interval. Repeating this a few times rapidly shrinks the interval - leading to a reasonably accurate value for the root.

Q 2) Download the file **bisect.py** from welearn. This has a program that tries to find the root of

$$f(x) = x^3 - 7x^2 + 5 = 0$$

by the bisection method. Unfortunately - all the indentation has been messed up, and there are a few other errors as well. Correct the program.

If you had run the program in question 1 properly, it should have shown that the value of $f(x)$ switches sign between $x = 0$ and $x = 1$, and gain between $x = 6$ and $x = 7$. Thus $(0, 1)$ and $(6, 7)$ are two bracketing intervals. Use the (corrected) bisect.py to find these roots - note down their values in **Worksheet03.txt** .

Q 3) The program **bisect.py** - *even after the corrections* - is not that great! For one, it assumes (but does not check) that the endpoints of the bracketing interval is entered in the correct (ascending) order by the user, does not check whether the interval is actually a bracketing interval - and whether the exact root is actually reached in any given step. Modify the program to write a new one (name it **bisect_better.py**) to address these issues.

If, for example, $f(a) = 0$, you will want to print out "The root is " followed by the value of a, and then quit the program. To quit a running program, use the **exit()** function.

Q 4) The Regula Falsi method, involves moving to a new point given by

$$c = a - \frac{b-a}{f(b)-f(a)} f(a)$$

instead of to

$$c = \frac{a+b}{2}$$

as in the bisection method (Explanation - the value of c in Regula falsi is just the value of x at which the straight line joining the points $(a, f(a))$ and $(b, f(b))$ cuts the X axis). Write a program called **regula_falsi.py** and use it to find the root of

$$f(x) = \cos\left(\frac{x}{x-2}\right) = 0$$

that lies between 1 and 1.5 . You should continue the iteration until the magnitude of the function falls below 10^{-7}. *Can you figure out the "exact" value of this root by directly solving the equation?*

Q 5) In the **secant method**, you start with two points a and b (without caring whether (a, b) is a bracketing interval), and then move on to a new point c which is just the value of x at which the straight line joining the points $(a, f(a))$ and $(b, f(b))$ cuts the X axis. You then repeat the process, starting with a new pair (a, b) where one of the two members is this new number c and the other is whichever one of the two points a and b is closer to c. One way of achieving this is via

```
if abs(a-c) >  abs(b-c):

    a = c       # b is closer to the new point and is retained while a is cha
else:
        b = c
```

Write a program called **secant.py** that will use this method to solve the equation

$$e^{-x} = \log x$$

staring from the initial points $a = 1$, $b = 2$. Stop when the magnitude of $f(x)$ falls below 10^{-7}.

Q 6) The **Newton-Raphson method** is an iterative method for solving the equation $f(x) = 0$, where the next iterate x_{n+1} is taken to be the value of x where the tangent to the graph of $f(x)$ at the point $(x_n, f(x_n))$ cuts the axis:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

Write a program **NR.py** to implement this algorithm. You should stop the iterations when the gap between two successive iterates is smaller than 10^{-6}, or $|f(x)|$ is less than 10^{-7} or the number of iterations exceed 1000.

Use it to solve the equation

$$x = e^{-x}$$

starting with initial guesses, $x = 1$, $x = 100$ and $x = -100$ and note down how many iterates the program needs to converge to the root (to required accuracy).

Q 7) Consider the equation

$$3x - \tan x = 0$$

This has an obvious root $x = 0$. Other than that, there is a root somewhere between 1 and 1.5 . Use the bisection code you have written to determine this root to six decimal places.

In the python interpreter try to use the iteration scheme

$$x_{n+1} = \frac{1}{3} \tan x_n$$

to determine this root. Start from the initial guess $x_0 = 1.0$, and carry out 100 iterations and note down the fixed point that this approaches, if any. Repeat with the "equivalent" iteration

$$x_{n+1} = \tan^{-1}(3x)$$

again starting from $x_0 = 1.0$. Explain the difference that you observe in the two cases.

Note: the function atan(x) from the math module implements the function $\tan^{-1} x$

Q 8) Write a program that will ask the user for a value of λ and between 0 and 4 (and checks whether the user really did supply values in the correct range). It should then take a random value of x between 0 and 1, carry out the logistic map iteration

$$x_{n+1} = \lambda x_n (1 - x_n)$$

1000 times and then prints out the next 100 iterates. Note down what you observe (the patterns – not necessarily the numbers – and definitely not all the hundred numbers!) for the values of λ given by

$$\lambda = 0.5,\ 0.9,\ 2.0,\ 3.1,\ 3.5,\ 3.9$$

Q 9) Write another program that will repeat the same task, but for the iteration scheme

$$x_{n+1} = \lambda \sin(\pi x_n)$$

and the values

$$\lambda = 0.7,\ 0.72,\ 0.85,\ 1.0$$

Do you notice any common features here with the result for the logistic function?

Q 10) Write a program called **nd_fd.py** that calculates the numerical derivative of the function $\ln x$ at $x = 1$ by the forward difference method :

$$f'(x) \approx \frac{f(x+h) - f(x)}{h}$$

for a series of values of h ranging from $h = 0.1$ to $h = 10^{-6}$ and prints out both the derivative and the error E , which is the difference (absolute value) between this result and the exact value $f'(1) = 1$. Run the program and see whether the output obeys the expected $\mathcal{O}(h)$ behavior of the error.

Modify the program so that instead of writing the result to the screen it prints it out to a file called **nd_fd.out**. Plot $\ln E$ versus $\ln h$ using gnuplot. Fit a straight line to this plot using gnuplot's fit command and check that the slope of this curve is almost 1 (since $E \propto h$, we expect $\ln E \sim c + \ln h$).

Modify the program again, this time to make h go down all the way to 10^{-20}. Plot the graph of $\ln E$ versus $\ln h$ again - can you explain this behavior?

Repeat, but this time for the central difference algorithm

$$f'(x) \approx \frac{f(x+h) - f(x-h)}{2h}$$

Q 11) The trapezoidal formula for estimating the definite integral is given by

$$\int_a^b f(x)\,dx \approx h\left[\frac{f(a) + f(b)}{2} + f(a+h) + f(a+2h) + f(a+3h) + \ldots f\left(a + \overline{N-1}h\right)\right]$$

where $h = \frac{b-a}{N}$. Write a program trapezoid.py that estimates the integral

$$\int_0^1 \sin^2 x dx$$

by the trapezoid method for $N = 1, 10, 100, \ldots 10^6$. Try to find an estimate for the order of error in the estimate as a power of h.

Q 12) Write a program that uses the Simpson-1/3 rule

$$\int_a^b f(x)\, dx \approx \frac{h}{3}\left[f(a) + f(b) + 4\left(f_1 + f_3 + \ldots + f_{2N-1}\right) + 2\left(f_2 + f_4 + \ldots + f_{2N-2}\right)\right]$$

where $h = \dfrac{b-a}{2N}$ and $f_i \equiv f(a + ih)$ to estimate the integral

$$\int_0^1 \frac{dx}{1+x^2}$$

Take the number of intervals, $2N$, to be $10, 20, 40, 80, ...10240$. Try to find an estimate for the order of error in the estimate as a power of h.

Q 13) Write a function called **isAcute()** that will take the coordinates of three points p, q, r as inputs (each one of p, q, r are tuples with two elements each - the X and Y coordinates of the point) and return the Boolean constant True if the triangle formed by these three points is acute angled, and False otherwise. Use this to write a program that will determine the probability of a triangle formed by taking three random points in a 1×1 square being acute angled. Repeat for a 1×2 rectangle.

Note : a triangle is acute angled if its three sides a, b and c obey $a^2 < b^2 + c^2$ where a is the largest side.

Q 14) Consider the popular fairground game in which the player draws a certain number of chits of paper, each bearing a natural number and wins a prize that depends on the sum of the numbers he or she has drawn. The following function simulates a single such game

```
def sumOfNumbers(max=9,N=5):
        # returns the sum of N (default value=5) random natural numbers
        # ranging from 1 to max (default value = 9)
        # must import randint from the random module before usage
        return sum([randint(1,max) for i in range(N)])
```

Write a program called **fairgame.py** which will simulate a large number (maybe 100000) of these games (where the player draws 5 numbers, each in the range 1 to 9) and print out the estimated probability of each possible value of the sum.

We know from the central limit theorem that this probability is almost a Gaussian function

$$P(n) \approx A \exp\left(-\left(\frac{n-n_0}{\Delta}\right)^2\right)$$

Use gnuplot to fit this function and determine the values of the constants A, n_0 and Δ. Plot the estimated probability and the best fit function together.

Q 15) An iterated function system (IFS) is a set of functions, any one of which is chosen at random and applied on a current point to generate the next one. One example is where you start with a random point in the unit square (for this you will need two numbers, chosen randomly from $(0,1)$ to get the random point (x,y). Then either of the functions $f_1(x,y)$ or $f_2(x,y)$ is applied at random (with equal probability) to generate successive points, where

$$\begin{aligned} f_1(x,y) &= \left(\frac{x-y}{2}, \frac{x+y}{2}\right) \\ f_2(x,y) &= \left(1-\frac{x+y}{2}, \frac{x-y}{2}\right) \end{aligned}$$

Write a program that will write 1000000 (a million) points generated by this method to a file and use gnuplot to plot them (with dots).

Q 16) Another example of an IFS is where you start from a random point in the unit square and repeatedly keep on applying at random (with equal probability) any of the three functions given by

$$\begin{aligned} f_1(x,y) &= (R\cos\theta\, x - R\sin\theta\, y, R\sin\theta\, x + R\cos\theta y + 1) \\ f_2(x,y) &= (R\cos\theta x + R\sin\theta y, -R\sin\theta x + R\cos\theta y + 1) \\ f_3(x,y) &= (x,y) \end{aligned}$$

where R and θ are fixed for a particular case. Write a program that will take values of R, and θ (in degrees) as input, generate a million points by using one of the three functions above at every stage. Plot the figures you get in gnuplot (with dots) for the following set of values

(i) $\theta = 30°, R = 0.7$ (ii) $\theta = 45°, R = 0.57$ (iii) $\theta = 120°, R = 0.7$ (iv) $\theta = 150°, R = 0.8$