**ENGR 102 Sect 508 Lab 11a**

**100 points+[20points]**

**Reading assignment:**

|  |  |
| --- | --- |
| **Lecture Slides** | **L11** |
| **zyBook chapter 10** | **Complete all participation and challenge activities** |
| **Kincaid handout Ch 4 Bisection Algorithm** | **L06, p87**  **Newton method** |
| **Kincaid handout Ch 7 Differentiation** | **L06, p 501-503**  **Ch 7. Numerical Differentiation and Integration** |
| **Kincaid handout Ch 6 Spline Interpolation** | **Ch6 Spline Interpolation** |

*Attention!!*

*Team submission. one submission per team.*

*Submit* *your Py-files together with your word/pdf file with screenshots of your tests outputs. Include any derivations, comments and supplemental notes in your word/pdf files.*

*No pictures by the phone – it is impossible to read. You will be allowed to resubmit and reupload HW as many times as you want to within the due date/time, only last submission will be graded. No late submissions. For submission you may use this file as template: rename file including your name. Do not forget to put your name inside of this file as well. If it is a team work use Team Header, include the team number and all team members.*

**[50 points] Activity #1: Newton’s Method – to be done as a team**

This activity will help you see the operation of Newton’s Method by writing functions. You may find it useful to refer back to assignment 6 (the bisection method for finding roots, and the computation of derivatives by evaluating a polynomial numerically). You may reuse code from your team’s work on that project if you wish.

You should create both a program and a document that describes parts of your program.

Newton’s method is a way of getting successively better approximations to a root of a function, *f*. The process is to compute a sequence of approximations to the root, *xi*. The initial value, a “guess” at the root, is given as *x0*. Then, subsequent *xi* are found by the formula:

1. Begin by creating a function, **F**, that takes in some value, x, and returns the value of some function *f* at that value, x. Your function F can be an arbitrary function – it may be a polynomial, trigonometric, exponential/logarithmic, etc. function, or any combination. Initially, please choose a function that is very simple, for which you can easily compute the roots and derivatives by hand, e.g. x2-1. Write in the document what function you used.

We used function =>

1. We will next look at turning the numerical derivative calculations that we computed in lab 6 into a function. Recall that a derivative of a function at any value, *x*, could be approximated by computing a difference: or or , for a sufficiently small value of *a*.
   1. Write a function, **deriv**, that computes an approximate derivative of the function F at a particular value. The function deriv should take in a value x, and return the (approximate) derivative of *f*, i.e. *f’(x)*, at that value.
2. Next, create a function named **newton\_step** that performs one step of Newton’s method. Using the functions F and deriv, newton\_step should take as input a guess at a root, *xi*, and should return a new guess for the root, *xi+1*.
3. Next, create a function **newton** that will take an initial guess for a root, *x0*, and will compute a sequence of root approximations. The function should create and return a list of successive approximations to a root. It can stop when the difference between successive approximations is no more than 10-6.
4. Finally, create a complete program that asks a user for an initial guess of a root, and prints the set of root approximations that Newton’s method computes from that initial guess.

Once you have the program working, try the following. You should write the answers to the following questions in the document that accompanies your program.

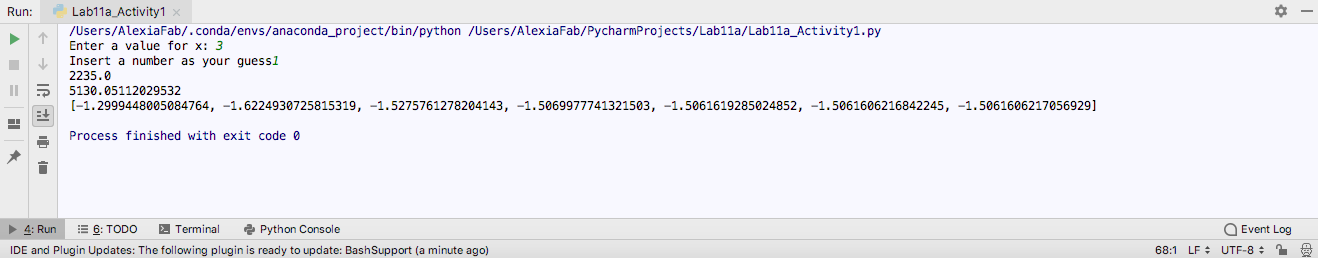
1. Change the function F to one that does not have any real roots (e.g. x2+1). Record this function in your document. What happens to your program? You may want to print out the values computed inside the function newton to see this better.
2. Change the function F to one that is more complex, where you do **not** know the derivative/roots ahead of time. Write in the document what the function is, as well as the answer to the following questions.
   1. Test out several different starting guesses for the root. Do all of your guesses converge to the same root? Yes
   2. Of several guesses, about how many successive values of *xi* are computed before the program has converged to a root?
   3. From your memory of the bisection approach for finding a root more accurately, which do you think converges to the root faster (i.e. which takes fewer iterations) – Bisection or Newton’s method?
   4. Compare the number of evaluations of the function F between the Bisection method and Newton’s method when we must approximate the derivative numerically.
      1. For each iteration of the Bisection method, how many times is the function F evaluated?
      2. For each iteration of Newton’s method, how many times is the function F evaluated?
      3. Does this change your thoughts about your answer in (c), above (comparing which converges faster between Bisection and Newton’s method)? Why or why not?

Turn in your program generated in part g with the document you put together.

Code:

*# By submitting this assignment, all team members agree to the following:  
# “Aggies do not lie, cheat, or steal, or tolerate those who do”  
# “I have not given or received any unauthorized aid on this assignment”  
#  
# Names: Alexia Perez  
# Bethany Gawalis  
# Sam Lyzzaik  
# Tyler Scataglia  
# Section: 508  
# Assignment: Lab 11a  
# Date: 09-11-2018***from** math **import \*  
import** numpy  
  
*# This is the full code for the problem, it is separated by parts because we worked on those first  
# but everything is working as one.  
  
# Part A: Create a function f that takes in a value for x and returns f(x)***def f**(*x*)**:  
 return** *x***\*\***7 **+** *x***\*\*** 3 **+** 21  
  
  
*# Part B: Create a function that will approximate the derivative of f(x):***def deriv**(*x*)**:** a **=** 10 **\*\* -**5  
 **return** (f(*x* **+** a) **-** f(*x*)) **/** a  
  
  
*# Part C: Create a function newton\_step that performs one step of Newton's method.  
# This function will use both f(x) and deriv(x) to take an input of a root Xi and return Xi+1:***def newton\_step**(*x*)**:** x1 **=** *x* **-** f(*x*) **/** deriv(*x*)  
 **return** x1  
  
  
*# Part D: Compute several approximations of the root until the difference between them is less than 10\*\*-6:***def newton**(*x\_0*)**:** step\_list **=** list()  
 approx\_one **=** *x\_0* approx\_two **=** newton\_step(*x\_0*)  
 step\_list.append(approx\_two)  
 **while** abs(approx\_one **-** approx\_two) **>** 10 **\*\* -**6**:** approx\_one **=** approx\_two  
 approx\_two **=** newton\_step(approx\_one)  
 step\_list.append(approx\_two)  
 **return**(step\_list)  
  
  
**def main**()**:** x **=** float(input(**'Enter a value for x: '**))  
 x\_0 **=** int(input(**'Insert a number as your guess'**))  
 new\_list **=** newton(x\_0)  
 print(f(x))  
 print(deriv(x))  
 print(new\_list)  
  
  
main()

Output:



Challenges: One interesting thing we can do in Python is to easily pass a function as an argument to another function! That is, we could call a function dosomething(myfunc) where myfunc is a function. Then, inside of the body of dosomething, we could have code such as myfunc(x).

1. [10 points] See if you can rewrite the Newton’s Method program so that the newton routine takes in both a function and an initial guess (instead of assuming the function is F). Test with multiple functions.
2. [10 points] See if you can modify the Newton routine so that it will detect when it seems there are no real roots and output a message accordingly.

**[50 points] Activity #2: An Interpolation Function – to be done as a team**

We have encountered the process of interpolation a few times in this course – most recently in Lab 9. You may wish to refer back to your team’s work on prior assignments when doing this one, and may reuse code from them if you wish. You will create 2 versions of an interpolation program, and should save these as 2 different files, labeled 2a and 2b.

1. Your team is to create a function that will perform linear interpolation from a set of measured data. The function should take as input a list of values at which samples were taken, and then another list giving the measurements (you can assume each measurement is a single value) at those values. It should also take in a query value, and should give the best estimate it can of the value at that query. Be sure to handle values that are outside of the range, by extrapolating.

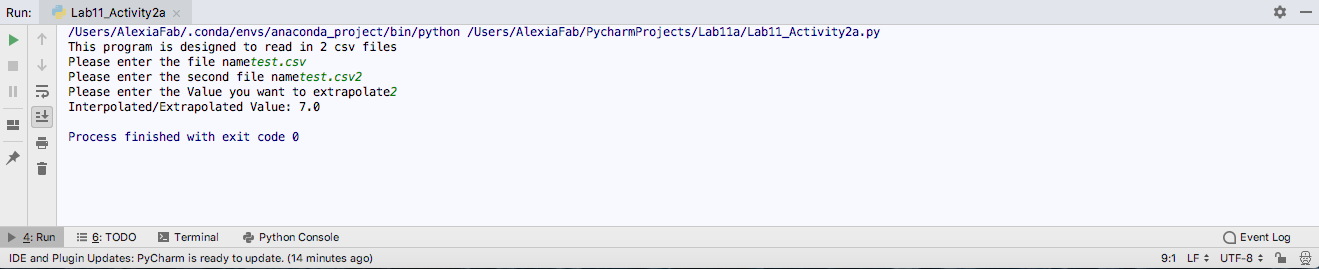
You should write a program that allows you to test your function by reading the lists from a file where each line of the file is a pair of numbers separated by spaces: the value where the sample was taken, and the measurement at that value. Your program should ask the user for the name of the file and for a query value.

Important: The two lists will correspond to each other: i.e. for the i-th value in the first list, the measurement will be the i-th element of the second list (these are called parallel lists or arrays). But, you should **not** assume that the input values are in increasing/decreasing order. That is, the values in the first list can be in any random ordering, not necessarily from smallest to largest or largest to smallest. You will have to account for this in your program, and there is more than one way to do so. As a team, you should discuss what options you can think of to handle the data arriving in any order like that, and decide what you think the best option for handling it is.

Code:

*# By submitting this assignment, all team members agree to the following:  
# “Aggies do not lie, cheat, or steal, or tolerate those who do”  
# “I have not given or received any unauthorized aid on this assignment”  
#  
# Names: Alexia Perez  
# Bethany Gawalis  
# Sam Lyzzaik  
# Tyler Scataglia  
# Section: 508  
# Assignment: Lab 11a  
# Date: 09-11-2018***from** math **import \*  
import** numpy  
  
file **=** open(**'test.csv'**, **'w'**)  
file.write(**'2,3,4,5,6,7'**)  
file.close()  
  
  
file **=** open(**'test.csv2'**, **'w'**)  
file.write(**'7,9,11,13,14,18'**)  
file.close()  
  
  
  
  
**def inter\_exter**(*x\_list*,*y\_list*,*value*)**:** y\_max **=** max(*y\_list*)  
 y\_min **=** min(*y\_list*)  
 x\_at\_max **=** *x\_list*[*y\_list*.index(y\_max)]  
 x\_at\_min **=** *x\_list*[*y\_list*.index(y\_min)]  
 Y **=** y\_min **+** (*value* **-** x\_at\_min) **\*** ((y\_max **-** y\_min) **/** (x\_at\_max **-** x\_at\_min))  
 **return** Y  
  
  
print(**"This program is designed to read in 2 csv files"**)  
file\_name **=** input(**"Please enter the file name"**)  
file\_name2 **=** input(**"Please enter the second file name"**)  
userV **=** int(input(**"Please enter the Value you want to extrapolate"**))  
File **=** open(**'test.csv'**,**'r'**)  
File2 **=** open(**'test.csv2'**, **'r'**)  
xList **=** []  
yList **=** []  
temp **=** []  
**for** next\_line **in** File**:** *# Stores the data from the 1st file into one large list* xList **=** next\_line.split(**","**)  
**for** next\_line **in** File2**:** *# Stores the data from the 2nd file into one large list* yList **=** next\_line.split(**","**)  
File.close()  
File2.close()  
**for** x **in** range(len(xList))**:** *# Changes data from strings to float values* xList[x] **=** float(xList[x])  
 yList[x] **=** float(yList[x])  
print(**"Interpolated/Extrapolated Value:"**,inter\_exter(xList,yList,userV))

Output:



1. Extend your program from part (a) so that it will handle not just single values but vector data.

Your program should ask a user for a file name, and the number of dimensions, n, of the vector data. It should then read from the file, assuming the values at which measurements are taken are the first entry per line, and then there will be n entries, all space-separated. You may find it easier to store your vector data in numpy arrays.

Your interpolation function should return the vector data either using tuples or using arrays from the numpy module.

Code:

*# By submitting this assignment, all team members agree to the following:  
# “Aggies do not lie, cheat, or steal, or tolerate those who do”  
# “I have not given or received any unauthorized aid on this assignment”  
#  
# Names: Alexia Perez  
# Bethany Gawalis  
# Sam Lyzzaik  
# Tyler Scataglia  
# Section: 508  
# Assignment: Lab 11a  
# Date: 09-11-2018***from** math **import \*  
import** numpy  
**import** csv  
  
**def getMax** (*dim*)**:** max **=** float(list[0][*dim*])  
 **for** x **in** range(len(list))**:  
 if** (list[x][*dim*] **>** (max))**:** max **=** list[x][*dim*]  
 **return** max  
  
**def atMax** (*dim*)**:** max **=** float(list[0][*dim*])  
 **for** x **in** range(len(list))**:  
 if** (list[x][*dim*] **>** (max))**:** max **=** list[x][0]  
 **return** max  
  
**def getMin** (*dim*)**:** min **=** float(list[0][*dim*])  
 **for** x **in** range(len(list))**:** *#print(list[x][dim])  
 #print(min)* **if** (float(list[x][*dim*]) **<** (min))**:** min **=** list[x][*dim*]  
 **return** min  
  
**def atMin** (*dim*)**:** min **=** float(list[0][*dim*])  
 **for** x **in** range(len(list))**:  
 if** (list[x][*dim*] **<** (min))**:** min **=** list[x][0]  
 **return** min  
  
n **=** int(input(**"Please enter the number of dimensions"**))  
userX **=** int(input(**"Please enter the X-Value you want to extrapolate"**))  
File **=** open(**'test3.csv'**, **'r'**)  
inp **=** csv.reader(File)  
list **=** list(inp)  
**for** i **in** range(len(list))**:  
 for** j **in** range(len(list[i]))**:** list[i][j]**=** float(list[i][j])  
temp **=** []  
i **=** 0  
  
**for** next\_line **in** File**:** temp **=** next\_line.replace(**"**\n**"**,**" "**)  
 temp **=** temp.split(**" "**)  
 temp.pop()  
 **if** i **==** 0**:** list[i] **=** temp  
 **else:** list.append(temp)  
 i **+=** 1  
**for** x **in** range(len(list))**:  
 for** y **in** range(n)**:** list[x][y] **=** float(list[x][y])  
print(getMin(3))  
print(atMin(3))  
print(getMax(3))  
print(atMax(3))  
  
**for** x **in** range(1,n)**:** yMin **=** getMin(x)  
 yMax **=** getMax(x)  
 xMin **=** atMin(x)  
 xMax **=** atMax(x)  
 Y **=** yMin **+** (userX **-** xMin)**\***((yMax**-**yMin)**/**(xMax**-**xMin))  
 print(**"For the"**,x,**"dimension:"**,Y)

Output:

