

## MAE 3210 - Spring 2019 - Project 2 Honor Pledge

**REMINDER:** Projects are to be treated as take-home exams with **NO** collaboration or discussion with other students.

Along with all of your code and results, you must sign, scan, and submit the following form in order to receive credit for project 2. By signing this form you are pledging to Utah State University that you abided by the policy written above in the completion of this project. That is, you did **NOT** discuss the problems assigned, your code, or any challenges inherent to project 2 with other USU students in or out of the class. You are only allowed to ask questions related to this project of the course instructor (Geordie Richards), course TA (Nate Hall), and course grader (Jacob Bryan).

That is, you treated project 2 as a **TAKE-HOME EXAM**. Violation of this policy will be treated as plagiarism and appropriate disciplinary action will be implemented, as outlined in the USU University Policies found on the course syllabus.

  
your name, printed, and signature

## MAE 3210 - Spring 2019 - Project 2

Project 2 is due **online** through Canvas by 11:59PM on Wednesday, April 3.

### IMPORTANT REMARKS (Please read carefully):

- **Projects are to be treated as take-home exams with NO collaboration or discussion with other students.** Plagiarism will be monitored and considered as cheating. **In addition to your project PDF, you are required to sign and submit an honor pledge which can be found adjacent to the project handout, under assignments in Canvas.** Each project is worth 50 points, and a 10 point loss per day will apply to late projects.
- You are required to submit code for all functions and/or subroutines built to solve these problems, which is designed to be easy to read and understand, in your chosen programming language, **and which you have written yourself.** The text from your code should both be copied into a single PDF file submitted on canvas. **Your submitted PDF must also include responses to any assigned questions, which for problems requiring programming should be based on output from your code.** For example, if you are asked to find a numerical answer to a problem, the number itself should be included in your submission.
- In addition to what is required for homework submission, for each numerical answer you reach based on running code you have written yourself, **your submitted project PDF must include a copy of a screenshot** that shows your computer window after your code has been executed, with the numerical answer displayed clearly on the screen.
- Note that, in problem 3 below, you are asked to use code that you have already written for homeworks 5 and 6. **You need to submit any code that you use for this project, including any code that you already wrote and submitted for homeworks 5 and 6.**
- If you are asked to use code that you have written yourself to solve a given problem (e.g. in problem 3 below), but you are unable to get that code working, you may choose, instead, to submit numerical answers based on running built-in functions (e.g. determinant computation, root finders, algebraic solvers already available in MATLAB), and you will receive partial credit. **However, you are required to write all code yourself, without relying on built-in functions, in order to get full points.**

1. Develop an algorithm that uses the Golden section search to locate the minimum of a given function. Rather than using the iterative stopping criteria we have previously implemented, design the algorithm to begin by determining the number of iterations  $n$  required to achieve a desired absolute error  $|E_{a,d}|$  (not a percentage), where  $|E_{a,d}|$  is input by the user. You may gain insight by comparing this approach to a discussion regarding the bisection method on page 132 of the textbook. Test your algorithm by applying it to find the minimum of  $f(x) = 2x + \frac{3}{x}$  with initial guesses  $x_l = 1$  and  $x_u = 5$  and desired absolute error  $|E_{a,d}| = 0.0001$ .
2. A manufacturing firm produces and sells four types of automobile parts. Each part is first fabricated and then finished, and the time required to complete these stages varies between the part types. The required worker hours and profit for each part type are given by the following table:

|                                  | Part A | Part B | Part C | Part D |
|----------------------------------|--------|--------|--------|--------|
| Fabrication time (hrs/100 units) | 2.5    | 1.5    | 2.75   | 2      |
| Finishing time (hrs/100 units)   | 3.5    | 3      | 3      | 2      |
| Profit (\$/100 units)            | 375    | 275    | 475    | 325    |

The capacities of the fabrication and finishing shops for a given month are 640 and 960 hours, respectively. Using MS Excel (or equivalent software), apply the simplex method to determine how many of each part should be produced in order to maximize profit. Submit a screenshot of your completed excel file and generated solution within your project PDF.

3. The dynamic viscosity of water  $\mu$  (in units of  $10^{-3} N \cdot s/m^2$ ) varies with temperature  $T$  ( $^{\circ}C$ ), as evidenced from the following measurements

|       |       |       |       |       |        |        |
|-------|-------|-------|-------|-------|--------|--------|
| $T$   | 0     | 5     | 10    | 20    | 30     | 40     |
| $\mu$ | 1.787 | 1.519 | 1.307 | 1.002 | 0.7975 | 0.6529 |

- (a) Use your polynomial regression algorithm (from homework 5) to fit a parabola to these data points in order to predict  $\mu$  at  $T = 2.5^{\circ}C$ . What is the value you obtained for  $r^2$ ? Generate a plot comparing the data points to your polynomial fit.
- (b) *Andrade's equation* has been proposed as a model for this relationship, which claims that

$$\mu = De^{B/T_a}, \quad (1)$$

where  $T_a$  = absolute temperature of the water ( $K$ ), and  $D$  and  $B$  are constants. Apply a mathematical transformation to (1) that will allow you to solve for  $B$  and  $D$  by applying the general regression algorithm you developed for homework 5. Use Andrade's equation to predict  $\mu$  at  $T = 2.5^{\circ}C$ . Generate a plot comparing the data points to your fit using Andrade's equation.

- (c) Use your Newton's interpolating polynomial algorithm (from homework 6) to predict  $\mu$  at  $T = 2.5^{\circ}C$ . Generate a plot comparing the data points to your polynomial interpolation.
- (d) Use your cubic spline interpolation algorithm (from homework 6) to predict  $\mu$  at  $T = 2.5^{\circ}C$ . Generate a plot comparing the data points to your cubic spline interpolation.

```

"""
Numerical Methods Project 2

Problem 1

@author: Jacob Needham
"""
import math

'''
One function to rule them all #LOTR
'''
def Main():
    a,b,Ea = getEa()
    n = numberOfIterations(Ea,a,b)
    goldenSearchMin(Ea,n,a,b)

'''
Function takes in lower and upper limits, and absolute error from user and
returns a tuple with answered data.
'''
def getEa():
    a = float(input("What is the lower limit to search?\n"))
    b = float(input("What is the upper limit to search?\n"))
    Ea = float(input("What is the desired absolute error?\n"))
    print() #blank line to add space
    return a,b,Ea

'''
function to find min of
'''
def function(x):
    y = 2*x + 3/x
    return y

'''
Rounded function to find number of iterations to guarantee a desired error. Over rounds
to ensure that the error is on the low side
'''
def numberOfIterations(Ea,a,b):
    n = int(math.ceil(math.log2((abs(b-a))/Ea)))
    return n

'''
Function finds the min of a function based on upper limit, lower limit, and numer of
iterations
'''
def goldenSearchMin(Ea,n,L,U):
    iCount = 0
    R = ((5**.5)-1)/2
    xL = L
    xU = U
    d = R *(xU - xL)
    x1 = xL + d
    x2 = xU - d
    f1 = function(x1)

```

```

f2 = function(x2)
if f1<f2 :
    xopt = x1
    fx = f1
else:
    xopt = x2
    fx = f2
for iCount in range(n):
    d = R*d
    xinit = xU - xL
    if f1 < f2:
        xL = x2
        x2 = x1
        x1 = xL + d
        f2 = f1
        f1 = function(x1)
    else:
        xU = x1
        x1 = x2
        x2 = xU - d
        f1 = f2
        f2 = function(x2)
    iCount += 1
    if f1 < f2:
        xopt = x1
        fx = f1
    else:
        xopt = x2
        fx = f2
    if xopt == 0:
        printStatement(n,Ea,fx,xopt)
        print("And in this particular case the root has zero error")
        exit
printStatement(n,Ea,fx,xopt)

'''
Generic print statment
'''
def printStatement(n, Ea, fx, xopt):
    print("The program ran for", n, 'iterations with less than', Ea, 'error.')
    print("The minimal value found was:", round(fx,5))
    print("The x value for the minimal value is:",round(xopt,5))

Main()

```

```

1 """
2 Numerical Methods Project 2
3
4 Problem 1
5
6 @author: Jacob Needham
7 """
8 import math
9
10 '''
11 One function to rule them all #LOTR
12 '''
13 def Main():
14     a,b,Ea = getEa()
15     n = numberOfIterations(Ea,a,b)
16     goldenSearchMin(Ea,n,a,b)
17
18
19 '''
20 Function takes in lower and upper limits, and absolute error from user and
21 returns a tuple with answered data.
22 '''
23 def getEa():
24     a = float(input("What is the lower limit to search?\n"))
25     b = float(input("What is the upper limit to search?\n"))
26     Ea = float(input("What is the desired absolute error?\n"))
27     print() #blank line to add space
28     return a,b,Ea
29
30 '''
31 function to find min of
32 '''
33 def function(x):
34     y = 2*x + 3/x
35     return y
36
37 '''
38 Rounded function to find number of iterations to guarantee a desired error. Over rounds
39 to ensure that the error is on the low side
40 '''
41 def numberOfIterations(Ea,a,b):
42     n = int(math.ceil(math.log2((abs(b-a))/Ea)))

```

```

In [102]: runfile('C:/Users/Needh/Documents/Spring 2019/Numerical
Methods/Project 2/Problem 1.py', wdir='C:/Users/Needh/Documents/
Spring 2019/Numerical Methods/Project 2')

What is the lower limit to search?
1

What is the upper limit to search?
5

What is the desired absolute error?
.0001

The program ran for 16 iterations with less than 0.0001 error.
The minimal value found was: 4.89898
The x value for the minimal value is: 1.22472

In [103]:


```

| Simplex Method Optimization for Automobile Parts |        |        |        |        |              |           |
|--|--------|--------|--------|--------|--------------|-----------|
|  | Part A | Part B | Part C | Part D | Combined     | Available |
| Number of Products Made                          | 0      | 192    | 128    | 0      |              |           |
| Fab time / 100 units                             | 2.5    | 1.5    | 2.75   | 2      | 640          | 640       |
| finish time /100 units                           | 3.5    | 3      | 3      | 2      | 960          | 960       |
| Unit Profit (\$/100 units)                       | 375    | 275    | 475    | 325    |              |           |
|  |        |        |        |        | TOTAL PROFIT |           |
| PROFIT   | 0      | 52800  | 60800  | 0      | 113600       |           |



Set Objective:  

To: ☒ Max ☐ Min ☐ Value Of:

By Changing Variable Cells:  

Subject to the Constraints:

- \$G\$7 <= \$H\$7

\$G\$8 <= \$H\$8
-   


Add


Change

Delete

Reset All

Load/Save

☒ Make Unconstrained Variables Non-Negative

Select a Solving Method:  

Options

Solving Method

Select the GRG Nonlinear engine for Solver Problems that are smooth nonlinear. Select the LP Simplex engine for linear Solver Problems, and select the Evolutionary engine for Solver problems that are non-smooth.

Help

Solve

Close

```
"""
```

## Numerical Methods Project 2

### Problem 3 Part A

Use your polynomial regression algorithm (from homework 5) to fit a parabola to these data points in order to predict  $u$  at  $T = 2.5$  C. What is the value you obtained for  $r^2$ ? Generate a plot comparing the data points to your polynomial  $t$ .

| x | 0     | 5     | 10    | 20    | 30     | 40     |
|---|-------|-------|-------|-------|--------|--------|
| y | 1.787 | 1.519 | 1.307 | 1.002 | 0.7975 | 0.6529 |

@author: Jacob Needham

```
"""
```

```
import matplotlib.pyplot as graph
import numpy as np
import math
```

```
x = np.matrix([0,5,10,20,30,40], dtype = 'float')
y = np.matrix([1.787,1.519,1.307,1.002,.7957,.6529], dtype = 'float')
m = 3 #number of basis functions plus a constant
n = 6 #height of matrix
z = np.matrix([[None for x in range(n)] for y in range(m)],dtype='float')
```

```
#Hard coding each part of the basis function
```

```
def a0(x):
    y = 1
    return y
```

```
def a1(x):
    y = x
    return y
```

```
def a2(x):
    y = x**2
    return y
```

```
#This populates the z matrix
```

```
for row in range(m):
    for column in range(n):
        if row == 0:
            z[0,column] = a0(x[0,column])
        if row == 1:
            z[1,column] = a1(x[0,column])
        if row == 2:
            z[2,column] = a2(x[0,column])
```

```
#orients matrix for output
```

```
z=z.transpose()
y=y.transpose()
```

```
#this solves for the solution vector that contains coefficients a1-an
```

```
#solution vector A = (z(t)*z)^-1 * z(t) * y
```

```
A = np.dot(np.dot(np.linalg.inv(np.dot(z.transpose(),z)),z.transpose()),y)
```

```

a0 = A[0,0]
a1 = A[1,0]
a2 = A[2,0]

#Printing solution equation
print('OUTPUT')
print("The quadratic function that most accurately pass through the points is:")
print("y= ",round(a0,4)," + ",round(a1,4),"*x + ",round(a2,4),"*x^2 + ",sep = '')

def function(x):
    y = a0 + a1*x + a2*x**2
    return y
'''
Interpolating quadratic function to find temperature at T = 2.5 C
'''
print("The interpolated value at 2.5 C is", round(function(2.5),4))

'''
Finding Correlation Coefficient and Standard Error
'''
#Standard Error
Sr = 0
for i in range(n):
    Sr += (y.item(i)-a0-a1*function(x.item(i))-a2*function(x.item(i)))**2
Sr = math.sqrt(Sr/(n-m+3))

#Correlation Coefficient
yBar = y.sum()/n
St = 0
for i in range(n):
    St += ((y.item(i)-yBar))**2
r = math.sqrt((St-Sr)/St)

print("The coefficient of determination is:",round(r,4))

'''
Building a plot to graph points and the function
'''
def function2(x):
    y = x
    return y

def plotSpace ():
    #setting up function
    x = np.arange(-100,100,.001)
    y = function(x)
    y2 = function2(x) #this is used to get the full range of y values for the axis lines
    graph.ylim(0,2)
    graph.xlim(-5, 45)
    #plotting function
    graph.plot(x, y)

    #plotting data set

```

```

xSeries  = [0,5,10,20,30,40]
ySeries  = [1.787,1.519,1.307,1.002,.7957,.6529]
graph.plot(xSeries,ySeries,'ro')

#setting up graph
graph.xlabel('x - axis')
graph.ylabel('y - axis')
graph.title('Project 2 Problem 3 Part A')

#plotting axis
graph.plot(x, x*0 + 0 , linewidth = .5, color = 'black')
graph.plot(x*0 +0, y2, linewidth =.5, color = 'black')

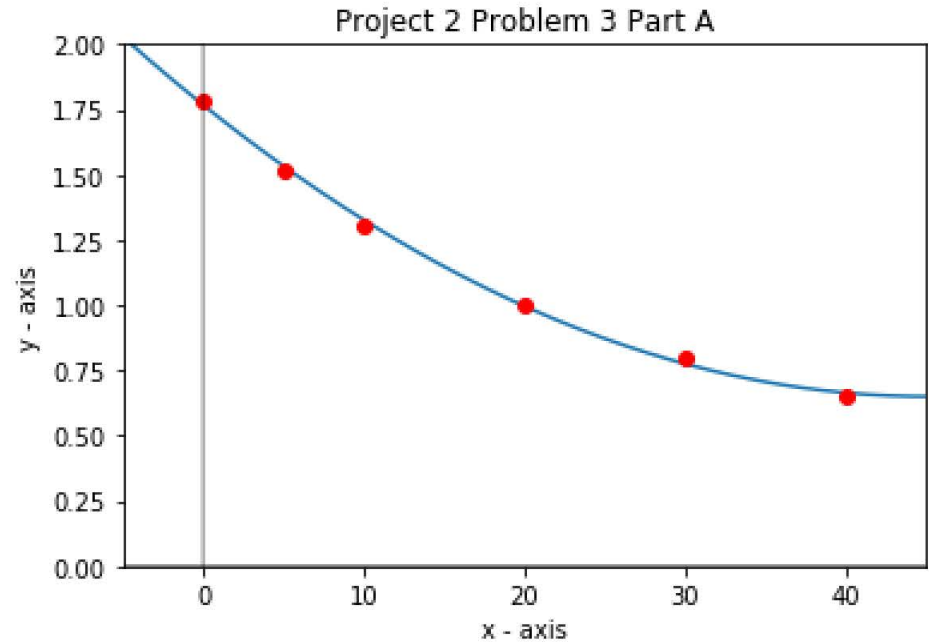
#grpahing
graph.show()

plotSpace()

```

```
1 """
2 Numerical Methods Project 2
3
4 Problem 3 Part A
5 Use your polynomial regression algorithm (from homework 5) to fit a parabola to these
6 data points in order to predict u at T = 2.5 C. What is the value you obtained for r^2?
7 Generate a plot comparing the data points to your polynomial t.
8
9 x | 0      5      10     20     30     40
10 -----
11 y | 1.787 1.519 1.307 1.002 0.7975 0.6529
12
13 @author: Jacob Needham
14 """
15 import matplotlib.pyplot as graph
16 import numpy as np
17 import math
18
19 x = np.matrix([0,5,10,20,30,40], dtype = 'float')
20 y = np.matrix([1.787,1.519,1.307,1.002,.7957,.6529], dtype = 'float')
21 m = 3                                #number of basis functions plus a constant
22 n = 6                                #height of matrix
23 z = np.matrix([[None for x in range(n)] for y in range(m)],dtype='float')
24
25
26 #Hard coding each part of the basis function
27 def a0(x):
28     y = 1
29     return y
30
31 def a1(x):
32     y = x
33     return y
34
35 def a2(x):
36     y = x**2
37     return y
38
39
40 #This populates the z matrix
41 for row in range(m):
42     for column in range(n):
```

```
In [104]: runfile('C:/Users/Needh/Documents/Spring 2019/Numerical
Methods/Project 2/Problem 3 Part A.py', wdir='C:/Users/Needh/
Documents/Spring 2019/Numerical Methods/Project 2')
OUTPUT
The quadratic function that most accurately pass though the points is:
y= 1.7675 + -0.0496*x + 0.0005*x^2 +
The interpolated value at 2.5 C is 1.647
The coefficient of determination is: 0.5402
```



In [105]:

```
"""
```

## Numerical Methods Project 2 Problem 3 Part B

Andrade's equation has been proposed as a model for this relationship, which claims that (see equation) where  $T_a$  = absolute temperature of the water (K), and D and B are constants. Apply a mathematical transformation to (1) that will allow you to solve for B and D by applying the general regression algorithm you developed for homework 5. Use Andrade's equation to predict  $u$  at  $T = 2.5$  C. Generate a plot comparing the data points to your  $t$  using Andrade's equation.

@author: Jacob Needham

```
"""
```

```
import matplotlib.pyplot as graph
import numpy as np
import math

def Main():

    x = np.matrix([0,5,10,20,30,40], dtype = 'float')
    y = np.matrix([math.log(1.787),math.log(1.519),math.log(1.307),math.log(1.002),\
                  math.log(.7957),math.log(.6529)], dtype = 'float')
    m = 2                                     #number of basis functions plus a constant
    n = 6                                     #height of matrix
    z = np.matrix([[None for x in range(n)] for y in range(m)],dtype='float')

    coef1,coef2 = genRegAlg(m,n,x,y,z,a0,a1)

    plotSpace(coef1,coef2)

    print("The interpolated value at x = 2.5 C =",round(function(2.5,coef1,coef2),4))

#Hard coding each part of the basis function
def a0(x):
    y = 1
    return y

def a1(x):
    y = 1/(x+273.15)
    return y

def genRegAlg(m,n,x,y,z,a0,a1):

    #Populating the z matrix
    for row in range(m):
        for column in range(n):
            if row == 0:
                z[0,column] = a0(x[0,column])
            if row == 1:
                z[1,column] = a1(x[0,column])

    #orients matrix for output
    z=z.transpose()
    y=y.transpose()

    #this solves for the solution vector that contains coefficients a1-an
    #solution vector A = (z(t)*z)^-1 * z(t) * y
```

```

A = np.dot(np.dot(np.linalg.inv(np.dot(z.transpose(),z)),z.transpose()),y)

coef1 = A[0,0]
coef2 = A[1,0]

return coef1, coef2
#Printing solution equation
#print("The base function that most accuratly pass though the points is:")
#print("y= ",round(a0,4)," + ",round(a1,4),"*x + ", sep = '')

def function(x,a0,a1):
    y = math.exp(a0) * math.exp( a1/(x+273.15))
    return y

def function2(x):
    y = x
    return y

def plotSpace (coef1,coef2):

    for i in range(0,4500):
        graph.plot(i/100,function(i/100,coef1,coef2),',')

    #setting up function
    x = np.arange(-100,100,.001)
    #y = function(x)
    y2 = function2(x) #this is used to get the full range of y values for the axis lines
    graph.ylim(0,2)
    graph.xlim(-5, 45)

    #plotting function
    #graph.plot(x, y)

    #plotting data set
    xSeries = [0,5,10,20,30,40]
    ySeries = [1.787,1.519,1.307,1.002,.7957,.6529]
    graph.plot(xSeries,ySeries,'ro')

    #setting up graph
    graph.xlabel('x - axis')
    graph.ylabel('y - axis')
    graph.title('Project 2 Problem 3 Part A')

    #plotting axis
    graph.plot(x, x*0 + 0 , linewidth = .5, color = 'black')
    graph.plot(x*0 +0, y2, linewidth =.5, color = 'black')

    #grpahing
    graph.show()

Main()

```

## Numerical Methods Project 2

### Problem 3 Part B

Algebra

$$\mu = D e^{B/T_a + 273.15}$$

$$\ln \mu = \ln D e^{B/T_a + 273.15}$$

$$= \ln D + \ln e^{B/T_a + 273.15}$$

$$= \ln D + \frac{B}{T_a + 273.15}$$

$$\ln \mu = \text{Const 1} + \frac{\text{Const 2}}{T_a + 273.15}$$

⇓

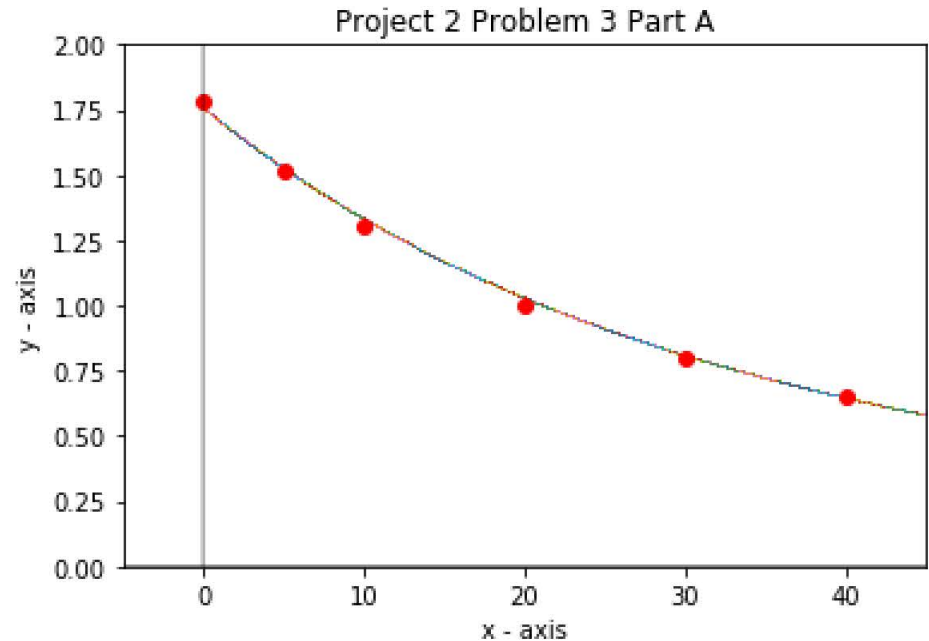
∴ y data from output needs to have log natural taken of it to solve for Const 1 and Const 2.

$$\mu = e^{\text{Const 1}} \cdot e^{\frac{\text{Const 2}}{T_a + 273.15}}$$



```
1 """
2 Numerical Methods Project 2 Problem 3 Part B
3
4 Andrade's equation has been proposed as a model for this relationship, which claims that
5 (see equation) where Ta = absolute temperature of the water (K), and D and B are
6 constants. Apply a mathematical transformation to (1) that will allow you to solve for
7 B and D by applying the general regression algorithm you developed
8 for homework 5. Use Andrade's equation to predict u at T = 2.5 C. Generate a plot
9 comparing the data points to your t using Andrade's equation.
10
11 @author: Jacob Needham
12 """
13
14 import matplotlib.pyplot as graph
15 import numpy as np
16 import math
17
18 def Main():
19
20     x = np.matrix([0,5,10,20,30,40], dtype = 'float')
21     y = np.matrix([math.log(1.787),math.log(1.519),math.log(1.307),math.log(1.002),\
22                   math.log(.7957),math.log(.6529)], dtype = 'float')
23     m = 2                                     #number of basis functions plus a constant
24     n = 6                                     #height of matrix
25     z = np.matrix([[None for x in range(n)] for y in range(m)],dtype='float')
26
27     coef1,coef2 = genRegAlg(m,n,x,y,z,a0,a1)
28
29     plotSpace(coef1,coef2)
30
31     print("The interpolated value at x = 2.5 C =",round(function(2.5,coef1,coef2),4))
32
33 #Hard coding each part of the basis function
34 def a0(x):
35     y = 1
36     return y
37
38 def a1(x):
39     y = 1/(x+273.15)
40     return y
41
42 def genRegAlg(m,n,x,y,z,a0,a1):
```

In [106]: runfile('C:/Users/Needh/Documents/Spring 2019/Numerical Methods/Project 2/Problem 3 Part B.py', wdir='C:/Users/Needh/Documents/Spring 2019/Numerical Methods/Project 2')



The interpolated value at x = 2.5 C = 1.6293

In [107]:

```
# -*- coding: utf-8 -*-
"""
```

## Numerical Methods Project 2 Problem 3 Part C

Use your Newton's interpolating polynomial algorithm (from homework 6) to predict  $u$  at  $T = 2.5$  C. Generate a plot comparing the data points to your polynomial interpolation.

@author: Jacob Needham

```
"""
```

```
import matplotlib.pyplot as graph
import numpy as np
```

```
...
```

Main function takes input data from table with  $x$  and  $y$  values and runs Fdd function.

```
...
```

```
def Main():
```

```
    #put numbers here
```

```
    x = [0,5,10,20,30,40]
```

```
    y = [1.787,1.519,1.307,1.002,.7957,.6529]
```

```
    n = len(x)
```

```
    Xi = 2.5 #input x value to solve for here
```

```
    q = FddMatrix(x,y,n,Xi)
```

```
    plotSpace(x,y,n)
```

```
    print("Interpolating at 2.5 C yields:",q)
```

```
...
```

FddMatrix takes in vectors  $x$ ,  $y$  from table and solves for linear fit between points

```
...
```

```
def FddMatrix(x,y,n,Xi):
```

```
    #initilizing Fdd matrix and placeholder matrix for y
```

```
    Fdd = np.matrix([[0 for x in range(n)] for y in range(n)],dtype = 'float')
```

```
    y_working = np.zeros((n,1), dtype = 'float')
```

```
    for i in range(n): #For loop coppies y value into first
        Fdd[i,0] = y[i] #column.
```

```
    for j in range(1,n): #For loop fills A matrix using modified
        for i in range(0,n-j): #Eq. 18.2 for linear interpolation from
            Fdd[i,j] = (Fdd[i+1,j-1] - Fdd[i,j-1])/(x[i+j]-x[i]) #the book.
```

```
    #initializing necessary place-holding matrices
```

```
    x_temp = [1]
```

```
    y_working[0] = Fdd[0,0]
```

```
    for order in range(1,n):
```

```
        x_temp.append(x_temp[order-1]*(Xi - x[order-1]))
```

```
        y_working[order] = y_working[order-1] + Fdd[0,order]*x_temp[order]
```

```
    return y_working[n-1]
```

```
    #formatting output and printing solution
```

```
    #final_y = str(y_working[n-1])
```

```
    #final_y = final_y.strip(' /[//]')
```

```
    #print('At x = ',Xi,' the interpolated y value is: ', final_y, sep = '')
```

```
def function2(x):
```

```
    y = x
```

```

    return y

def plotSpace (x,y,n):

    for i in range(0,4500):
        graph.plot(i/100,FddMatrix(x,y,n,i/100),',')

    #setting up function
    x = np.arange(-100,100,.001)
    #y = function(x)
    y2 = function2(x) #this is used to get the full range of y values for the axis lines
    graph.ylim(0,2)
    graph.xlim(-5, 45)

    #plotting function
    #graph.plot(x, y)

    #plotting data set
    xSeries = [0,5,10,20,30,40]
    ySeries = [1.787,1.519,1.307,1.002,.7957,.6529]
    graph.plot(xSeries,ySeries,'ro')

    #setting up graph
    graph.xlabel('x - axis')
    graph.ylabel('y - axis')
    graph.title('Project 2 Problem 3 Part A')

    #plotting axis
    graph.plot(x, x*0 + 0 , linewidth = .5, color = 'black')
    graph.plot(x*0 +0, y2, linewidth =.5, color = 'black')

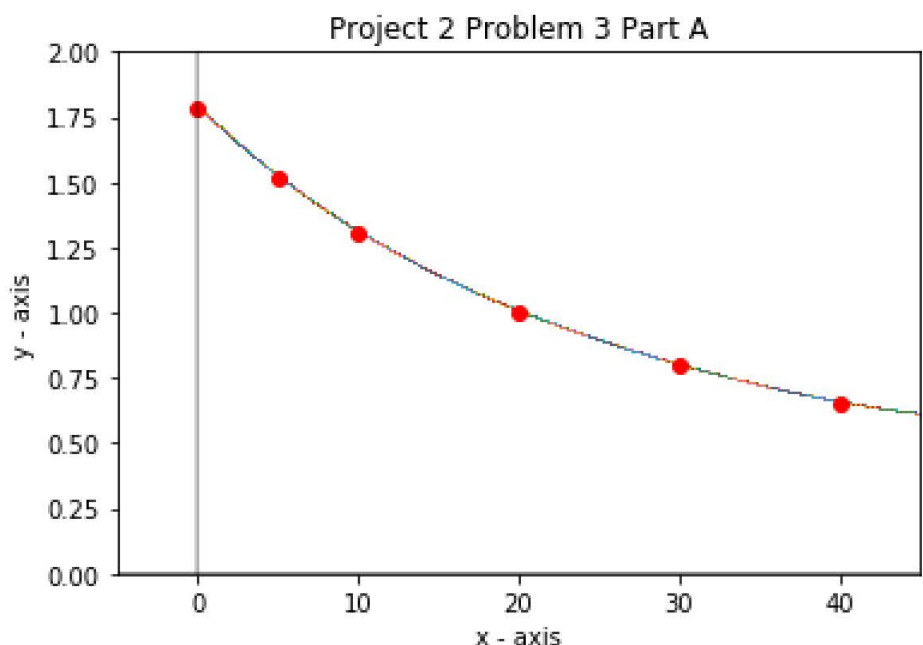
    #grpahing
    graph.show()

```

Main()

```
1 # -*- coding: utf-8 -*-
2 """
3 Numerical Methods Project 2 Problem 3 Part C
4
5 Use your Newton's interpolating polynomial algorithm (from homework 6) to predict u at
6 T = 2.5 C. Generate a plot comparing the data points to your polynomial interpolation.
7
8 @author: Jacob Needham
9 """
10 import matplotlib.pyplot as graph
11 import numpy as np
12
13 '''
14 Main fucntion takes input data from table with x and y values and runs Fdd function.
15 '''
16 def Main():
17     #put numbers here
18     x = [0,5,10,20,30,40]
19     y = [1.787,1.519,1.307,1.002,.7957,.6529]
20     n = len(x)
21
22     Xi = 2.5 #input x value to solve for here
23     q = FddMatrix(x,y,n,Xi)
24
25     plotSpace(x,y,n)
26
27     print("Interpolating at 2.5 C yields:",q)
28
29 '''
30 FddMatrix takes in vectors x, y from table and solves for linear fit between points
31 '''
32 def FddMatrix(x,y,n,Xi):
33     #initilizing Fdd matrix and placeholder matrix for y
34     Fdd = np.matrix([[0 for x in range(n)] for y in range(n)],dtype = 'float')
35     y_working = np.zeros((n,1), dtype = 'float')
36     for i in range(n):
37         Fdd[i,0] = y[i] #For loop coppies y value into first
38                             #column.
39     for j in range(1,n):
40         for i in range(0,n-j):
41             Fdd[i,j] = (Fdd[i+1,j-1] - Fdd[i,j-1])/(x[i+j]-x[i]) #the book.
42
43     #initializing necessary place-holding matrices
```

In [108]: runfile('C:/Users/Needh/Documents/Spring 2019/Numerical Methods/Project 2/Problem 3 Part C.py', wdir='C:/Users/Needh/Documents/Spring 2019/Numerical Methods/Project 2')



Interpolating at 2.5 C yields: [ 1.64513284]

In [109]:

Permissions: RW

End-of-lines: CRLF

Encoding: UTF-8

Line: 85

Column: 19

Memory: 54 %

```
# -*- coding: utf-8 -*-
"""
```

## Numerical Methods Project 2 Problem 3 Part C

Use your Newton's interpolating polynomial algorithm (from homework 6) to predict  $u$  at  $T = 2.5$  C. Generate a plot comparing the data points to your polynomial interpolation.

@author: Jacob Needham

```
"""
```

```
import matplotlib.pyplot as graph
import numpy as np
```

```
...
```

Main function takes input data from table with  $x$  and  $y$  values and runs Fdd function.

```
...
```

```
def Main():
```

```
    #put numbers here
```

```
    x = [0,5,10,20,30,40]
```

```
    y = [1.787,1.519,1.307,1.002,.7957,.6529]
```

```
    n = len(x)
```

```
    Xi = 2.5 #input x value to solve for here
```

```
    q = FddMatrix(x,y,n,Xi)
```

```
    plotSpace(x,y,n)
```

```
    print("Interpolating at 2.5 C yields:",q)
```

```
...
```

FddMatrix takes in vectors  $x$ ,  $y$  from table and solves for linear fit between points

```
...
```

```
def FddMatrix(x,y,n,Xi):
```

```
    #initilizing Fdd matrix and placeholder matrix for y
```

```
    Fdd = np.matrix([[0 for x in range(n)] for y in range(n)],dtype = 'float')
```

```
    y_working = np.zeros((n,1), dtype = 'float')
```

```
    for i in range(n): #For loop coppies y value into first
        Fdd[i,0] = y[i] #column.
```

```
    for j in range(1,n): #For loop fills A matrix using modified
        for i in range(0,n-j): #Eq. 18.2 for linear interpolation from
            Fdd[i,j] = (Fdd[i+1,j-1] - Fdd[i,j-1])/(x[i+j]-x[i]) #the book.
```

```
    #initializing necessary place-holding matrices
```

```
    x_temp = [1]
```

```
    y_working[0] = Fdd[0,0]
```

```
    for order in range(1,n):
```

```
        x_temp.append(x_temp[order-1]*(Xi - x[order-1]))
```

```
        y_working[order] = y_working[order-1] + Fdd[0,order]*x_temp[order]
```

```
    return y_working[n-1]
```

```
    #formatting output and printing solution
```

```
    #final_y = str(y_working[n-1])
```

```
    #final_y = final_y.strip(' /[//]')
```

```
    #print('At x = ',Xi,' the interpolated y value is: ', final_y, sep = '')
```

```
def function2(x):
```

```
    y = x
```

```

    return y

def plotSpace (x,y,n):

    for i in range(0,3000):
        graph.plot(i/100,FddMatrix(x,y,n,i/100),',')

    #setting up function
    x = np.arange(-100,100,.001)
    #y = function(x)
    y2 = function2(x) #this is used to get the full range of y values for the axis lines
    graph.ylim(0,2)
    graph.xlim(-5, 45)

    #plotting function
    #graph.plot(x, y)

    #plotting data set
    xSeries = [0,5,10,20,30,40]
    ySeries = [1.787,1.519,1.307,1.002,.7957,.6529]
    graph.plot(xSeries,ySeries,'ro')

    #setting up graph
    graph.xlabel('x - axis')
    graph.ylabel('y - axis')
    graph.title('Project 2 Problem 3 Part A')

    #plotting axis
    graph.plot(x, x*0 + 0 , linewidth = .5, color = 'black')
    graph.plot(x*0 +0, y2, linewidth =.5, color = 'black')

    #grpahing
    graph.show()

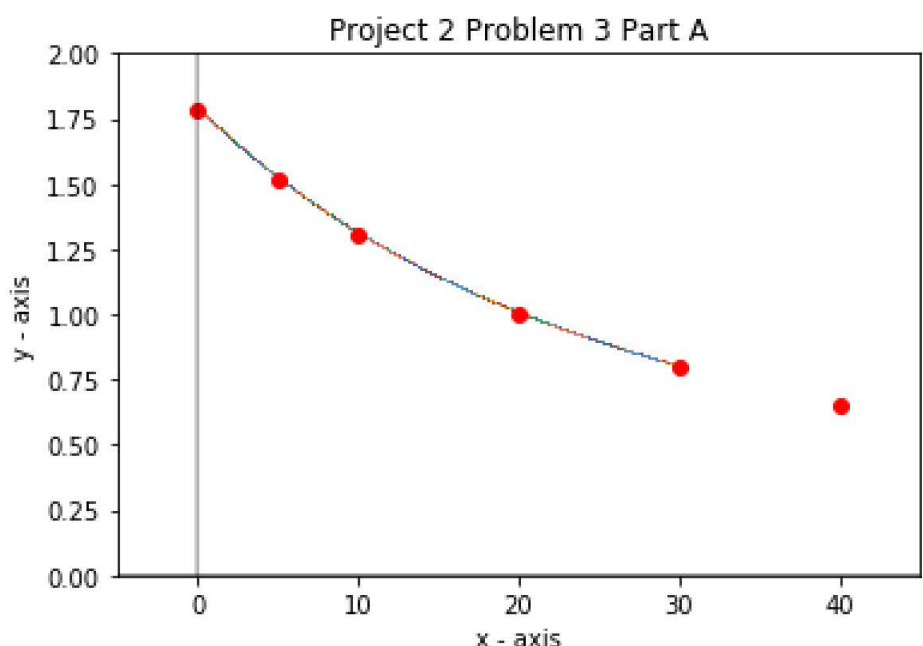
```

Main()



```
1 # -*- coding: utf-8 -*-
2 """
3 Numerical Methods Project 2 Problem 3 Part C
4
5 Use your Newton's interpolating polynomial algorithm (from homework 6) to predict u at
6 T = 2.5 C. Generate a plot comparing the data points to your polynomial interpolation.
7
8 @author: Jacob Needham
9 """
10 import matplotlib.pyplot as graph
11 import numpy as np
12
13 '''
14 Main fucntion takes input data from table with x and y values and runs Fdd function.
15 '''
16 def Main():
17     #put numbers here
18     x = [0,5,10,20,30,40]
19     y = [1.787,1.519,1.307,1.002,.7957,.6529]
20     n = len(x)
21
22     Xi = 2.5 #input x value to solve for here
23     q = FddMatrix(x,y,n,Xi)
24
25     plotSpace(x,y,n)
26
27     print("Interpolating at 2.5 C yields:",q)
28
29 '''
30 FddMatrix takes in vectors x, y from table and solves for linear fit between points
31 '''
32 def FddMatrix(x,y,n,Xi):
33     #initilizing Fdd matrix and placeholder matrix for y
34     Fdd = np.matrix([[0 for x in range(n)] for y in range(n)],dtype = 'float')
35     y_working = np.zeros((n,1), dtype = 'float')
36     for i in range(n): #For loop coppies y value into first
37         Fdd[i,0] = y[i] #column.
38     for j in range(1,n): #For loop fills A matrix using modified
39         for i in range(0,n-j): #Eq. 18.2 for linear interpolation from
40             Fdd[i,j] = (Fdd[i+1,j-1] - Fdd[i,j-1])/(x[i+j]-x[i]) #the book.
41
42     #initializing necessary place-holding matrices
43     '''
```

In [110]: runfile('C:/Users/Needh/Documents/Spring 2019/Numerical Methods/Project 2/Problem 3 Part C.py', wdir='C:/Users/Needh/Documents/Spring 2019/Numerical Methods/Project 2')



Interpolating at 2.5 C yields: [ 1.64513284]

In [111]: