NENG 685
HW 1

Fall 2017
Due Oct. 11, 2017

Name: Kevin Choe

On homework:

- If you work with anyone else, document what you worked on together.
- Show your work.
- Always clearly label plots (axis labels, a title, and a legend if applicable).
- Homework should be done "by hand" (i.e. not with a numerical program such as MATLAB, Python, or Wolfram Alpha) unless otherwise specified. You may use a numerical program to check your work.
- If you use a numerical program to solve a problem, submit the associated code, input, and output (email submission is fine).

Do not write in the table to the right.

Problem	Points	Score
1	20	20
2	30	24
3	15	15
4	30	23
Total:	95	87

+5

1. (20 points) Using four points on the interval $[x_0, x_3]$, do the following:
   (a) (4 points) Construct all of the Lagrange polynomials $L_j(x)$ that correspond to the points x_0, x_1, x_2, and x_3 by hand.
   (b) (4 points) Use these Lagrange polynomials to construct the interpolating polynomial, $P_3(x)$, that interpolates the function $f(x)$ at the points x_0, x_1, x_2, and x_3 by hand.
   (c) (8 points) Using the $P_3(x)$ you derived, create an interpolant for

$$f(x) = \sin(\frac{\pi}{2}x) + \frac{x^2}{4}$$

   over $[x_0, x_3]$ with $x_0 = 0, x_1 = 2, x_2 = 3$, and $x_3 = 4$. You may do this using something like Python or MATLAB, but *write your own functions rather than using the built in ones.*

   Plot the actual function and your interpolant using 100 equally spaced points for x between -0.5 and 4.5.

   (d) (4 points) Repeat what you did in part c but instead use $x_0 = 0, x_1 = 1, x_2 = 2.5$, and $x_3 = 4$.

   Discuss the differences in how well the function is interpolated using the different point sets.

Please see the attached pages in the back for answers!!!

2. (15 points) Using the interpolant $P_3(x)$ derived in question 1:
   (a) (3 points) Write the general expression for the error term, $err(x) = |f(x) - P_3(x)|$.
   (b) (4 points) Given
   $$f(x) = \sin(\frac{\pi}{2}x) + \frac{x^2}{4} \,,$$
   use information about the function to bound the error expression.
   (c) (8 points) Use the values $x_0 = 0, x_1 = 2, x_2 = 3$, and $x_3 = 4$ to get the upper bound of $err(x)$ over this interval. That is, insert the points into the expression from part b, find an expression for x that maximizes error, and then find the x that gives the maximum. Present one final number. You may use a mathematical package to assist you in solving for x.

3. (15 points) We have the following data:
   $$x = [1, 2, 3, 4, 5, 6, 7] \,,$$
   $$f(x) = [1, 4, 10, 12, 5, 4, 0] \,.$$
   (a) (10 points) Using *built in* Python or MATLAB functions, interpolate this data using
   - Piecewise linear interpolation
   - Lagrange polynomial interpolation
   - Spline interpolation

   Create a subplot for each of your interpolants over $[0.75, 6.25]$ using a fine mesh spacing, e.g. 0.05 (note that to use scipy's piecewise linear polynomial interpolation you will need to restrict the range to the exact endpoints, 1.0 and 6.0). Include the data points on the interpolation plots.
   (b) (5 points) Briefly discuss the differences between the resulting interpolations.

4. (30 points) The errors generated by a numerical method on a test problem with various grid resolutions have been recorded in the following table:

Please see the attached pages in the back for answers!!!

Grid Spacing (h)	Error (E)
5.00000e-02	1.036126e-01
2.50000e-02	3.333834e-02
1.25000e-02	1.375409e-02
6.25000e-03	4.177237e-03
3.12500e-03	1.103962e-03
1.56250e-03	2.824698e-04
7.81250e-04	7.185644e-05
3.90625e-04	1.813937e-05

For this numerical method, the error should be of the form

$$E = kh^p$$

(a) (3 points) Write this problem as a linear system $\mathbf{A}\vec{x} = \vec{b}$, where $x = \begin{pmatrix} \ln(k) \\ p \end{pmatrix}$ is the vector of unknowns.

(b) (5 points) Derive the normal equations for this over-determined system: write the matrices in $\mathbf{A}\vec{x} = \vec{b}$ form, where you include formulas / values for each entry.

(c) (9 points) Solve, using the program/language of your choice, the normal equations to obtain a least squares estimate to the parameters k and p.

(d) (6 points) Solve for the parameters k and p using SciPy's CurveFit function

(e) (7 points) Make a log-log and a lin-lin plot that displays both the input data and the function $E = kh^p$. Comment on the differences between the two approximations. (Checkout Python's matplotlib.pyplt.loglog command.)

BONUS (5 points): submit your code by providing read/clone access to an online version control repository where your code is stored (e.g. github or bitbucket). If you don't know what that means and want to learn about it, come talk to me or check out resources here: `http://software-carpentry.org/lessons.html` For Windows, you want to setup the Git GUI. NOTE: You will not be able to do this from AFIT's systems.

Please see the attached pages in the back for answers!!!

1(a) construct Lagrange polynomials $L_i(x)$ that correspond to the points $x_0 x_1, x_2, x_3$ by hand.

$$L_3(x) = \prod_{\substack{i=0 \\ i \neq k}}^{n} \frac{(x-x_2)}{(x_k-x_2)} \qquad n=3 \text{ in this case}$$

$$i=0, \quad L_0(x) = \frac{(x-x_1)(x-x_2)(x-x_3)}{(x_0-x_1)(x_0-x_2)(x_0-x_3)}$$

$$i=1, \quad L_1(x) = \frac{(x-x_0)(x-x_2)(x-x_3)}{(x_1-x_0)(x_1-x_2)(x_1-x_3)}$$

$$i=2, \quad L_2(x) = \frac{(x-x_0)(x-x_1)(x-x_3)}{(x_2-x_0)(x_2-x_1)(x_2-x_3)}$$

$$i=3, \quad L_3(x) = \frac{(x-x_0)(x-x_1)(x-x_2)}{(x_3-x_0)(x_3-x_1)(x_3-x_2)}$$

(b) use L.P. to construct interpolating polynomials, $P_3(x)$ by hand.

$$P_3(x) = \sum_{k=0}^{3} f(x_k) L_k(x)$$

$$= f(x_0)\cdot L_0(x) + f(x_1)\cdot L_1(x) + f(x_2)\cdot L_2(x) + f(x_3)\cdot L_3(x)$$

(c) use $P_3(x)$ derived to create interpolant for

$$f(x) = \sin\left(\frac{\pi}{2}x\right) + \frac{x^2}{4}$$

I used pyton for this problem; solution can be found in Github under "GradFall17 NENG 685" by "dabotop2"

(d) repeat (c) with different x values and discuss the differences in how well the function is interpolating

- you can find plots for (c) and (d) for comparison and discussion statement in "HW1_1_d.py" in Github.

2. (a) write the general expression for the error term.

$$\text{error}(x) = |f(x) - P_3(x)| = \frac{f^{4}\xi}{4!}(x-x_0)(x-x_1)(x-x_2)(x-x_3)$$

(b) Given $f(x) = \sin\left(\frac{\pi}{2}x\right) + \frac{x^2}{4}$, use information about the function to bound the error expression.

$$\frac{d^4}{dx^4}(f(x)) = \frac{d^4}{dx^4}\left(\sin\left(\frac{\pi x}{2}\right) + \frac{x^2}{4}\right)$$

$$= \frac{d^3}{dx^3}\left(\frac{d}{dx}\left(\sin\left(\frac{\pi x}{2}\right) + \frac{x^2}{4}\right)\right) = \frac{d^2}{dx^2}\left(\frac{d}{dx}\left(\frac{\pi\cos\left(\frac{\pi x}{2}\right) + x}{2}\right)\right)$$

$$= \frac{d}{dx}\left(\frac{d}{dx}\left(\frac{1}{2}\left(-\frac{\pi^2}{2}\sin\left(\frac{\pi x}{2}\right) + 1\right)\right)\right) = \frac{d}{dx}\left(-\frac{\pi^3}{8}\cos\left(\frac{\pi x}{2}\right)\right)$$

$$= \frac{\pi^4}{16}\sin\left(\frac{\pi x}{2}\right)$$

Since it's a sin function we can bound it between -1 and 1

$$-1 \leq \frac{\pi^4}{16}\sin\left(\frac{\pi x}{2}\right) \leq 1$$

we are interested in finding max error, this term

so my function(e) becomes

$$\text{error}(x) = \frac{\underbrace{\frac{\pi^4}{16}\sin\left(\frac{\pi x}{2}\right)\xi}_{\leq 1}}{4!}(x-x_0)(x-x_1)(x-x_2)(x-x_3)$$

$$\underbrace{\cdots}\leq e \leq \frac{\pi^4}{16}\cdot\frac{1}{4!}(x-x_0)(x-x_1)(x-x_2)(x-x_3)$$

we can ignore this term for now since we are looking for e_{max}

Plug in the given values $x_0 = 0,\ x_1 = 2,\ x_2 = 3,\ x_3 = 4$

$$e \leq \frac{\pi^4}{384}(x-0)(x-2)(x-3)(x-4)$$

now find x that maximizes e!

I used wolfram maximum find calculator (Attached in the back) to find it.

$e_{max} \approx 0.2388$ @ $x \approx 2.469$

-6pts: Note you had the bound in part b. Substituting and taking the derivative gives you the inflection points. Your approach could work, but you missed that the error expression has an abs value

Max/Min Finder

Find the maximum

of \frac{\pi ^4}{:

Submit

(no global maxima found)

Need a step by step solution for this problem? >>

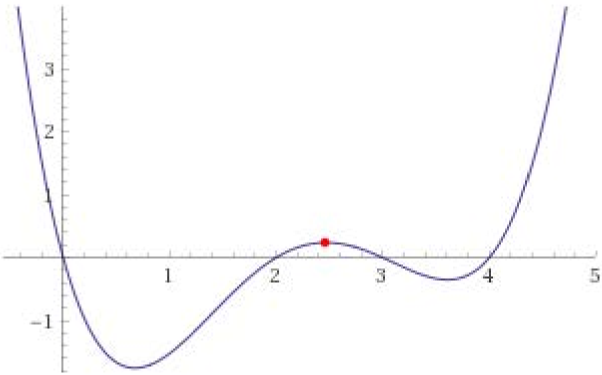
Local maximum:

More digits | Exact form

$$\max\left\{\frac{1}{384}\pi^4 (x-0)(x-2)(x-3)(x-4)\right\} \approx 0.238799 \text{ at } x \approx 2.46909$$

Need a step by step solution for this problem? >>

Plot:



(x from −0.4 to 4.9)

3. (a) with the give following data for x and f(x), interpolate these data using python "built in" function.

- Piecewise linear interpolation
- L.P. interpol
- Spline interpol

• You can find my code, 3 plots for each methods (w/ data points plot) in "HW1_3.Py" in Github.

(b) Briefly discuss the differences btw the resulting interpol's

• Discussion statement is included in "HW1_3.Py" in Github.

4. (a) Given error form, $E = kh^P$; write this problem as a linear system $A\vec{x} = \vec{b}$, where $x = \begin{pmatrix} \ln(k) \\ P \end{pmatrix}$ is the vector of unknowns.

$E = kh^P \rightarrow \ln(E) = \ln(kh^P) = \ln(k) + \ln(h^P) = \ln(k) + P\ln(h)$

$E_0 \ldots E_8$

$\underbrace{\ln(E)}_{\vec{b}} = \underbrace{\ln(k)}_{x_0} + \underbrace{P}_{x_1} \underbrace{\ln(h)}_{\vec{A}}$ $h_0 \vdots h_8$ ← I'm not writing 8 repetitive eq's with differen E ($E_0 \sim E_8$) and h ($h_0 \sim h_8$). This form represent the idea.

(b) Derive the normal equations for this sys: write the matrices in $\vec{A}\vec{x} = \vec{b}$ form.

$$\underbrace{\begin{pmatrix} 1 & \ln(h_0) \\ 1 & \ln(h_1) \\ 1 & \vdots \\ 1 & \\ 1 & \ln(h_m) \end{pmatrix}}_{\vec{A}} \underbrace{\begin{pmatrix} \ln(k) \\ P \end{pmatrix}}_{\vec{x}} = \begin{pmatrix} \ln(E_0) \\ \ln(E_1) \\ \vdots \\ \ln(E_8) \end{pmatrix}$$

where m=8

-5pts: See pg 7-8 of lesson 2 notes

Use transpose, A^T : $A^TA\vec{x} = A^T\vec{b}$, we can get $\vec{x}$ matrix.

(c) use python to solve (and w/ normal equation) for k and P via least squares methods

• Code, Plot, and k, P values can be found in "HW1_4" in Github.

-2pts: Really close, probably a typo

$P = 1.777$ $\quad$ $K = 27.053$

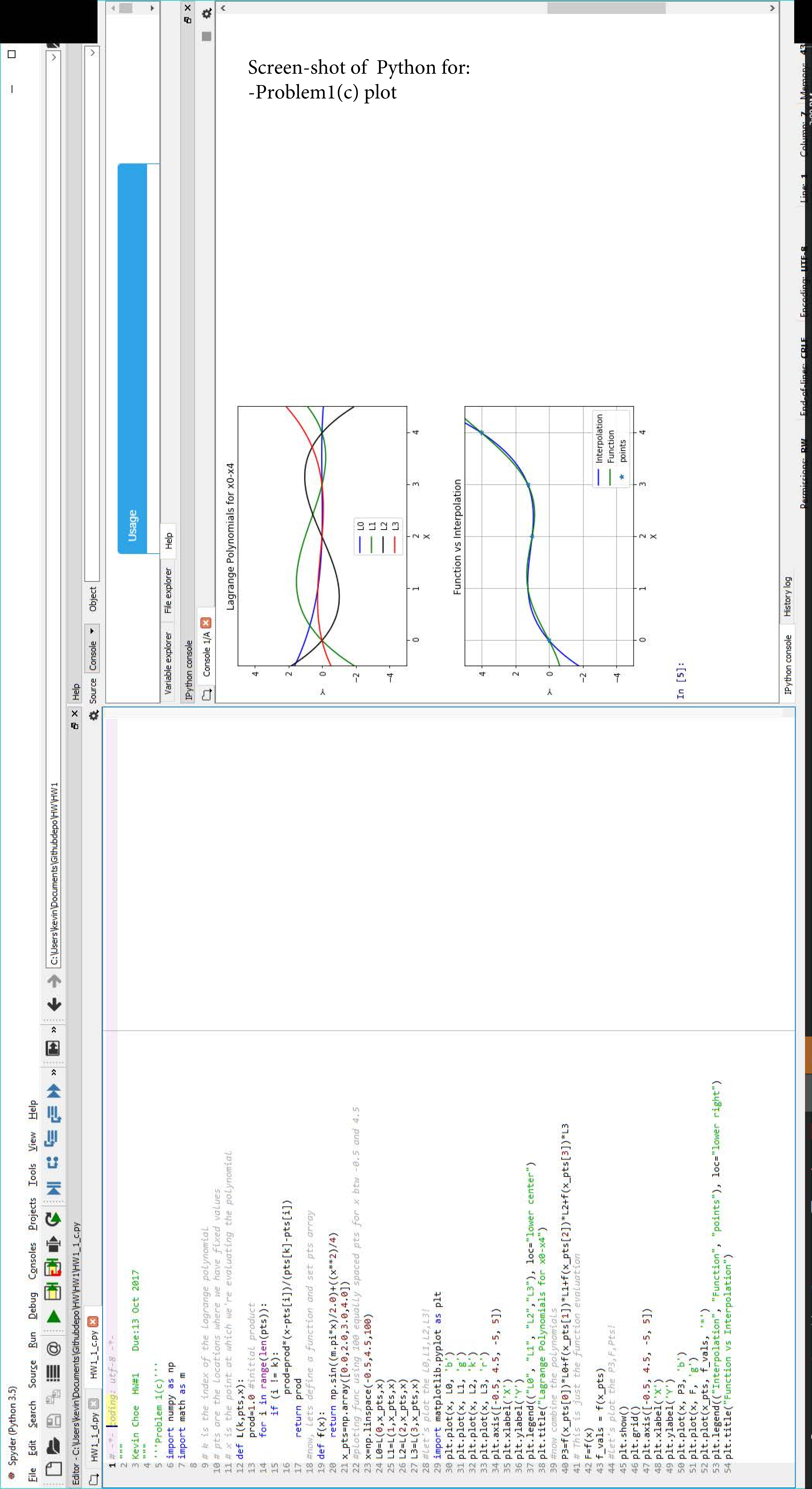
(d) Solve for K, P using Scipy's Curve fit.

• Code, Plot, and K, P values can be found in "HW1_4"

$P = 1.571$ $\quad$ $K = 11.432$

(e) log-log plot, lin-lin plot, comments can be found in "HW1_4"

Screen-shot of Python for:
-Problem1(c) plot



Screen-shot of Python for:
-Problem1(d) plot & discussion (in green text)

Spyder (Python 3.5)

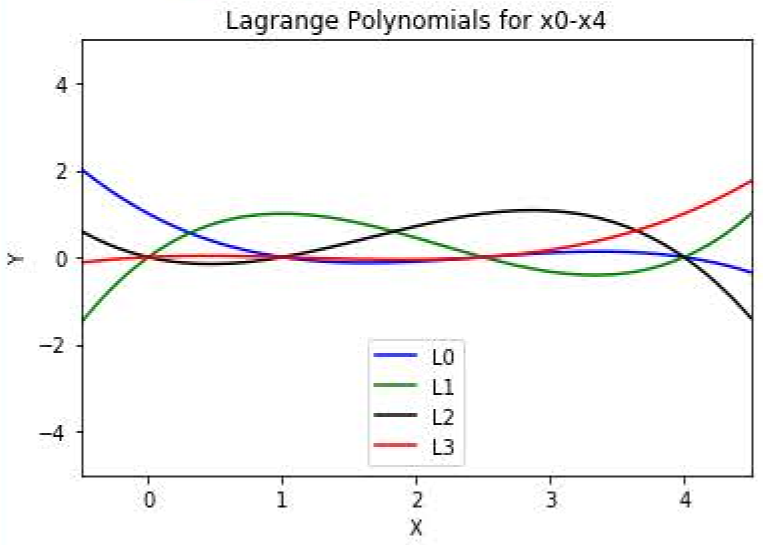
Editor - C:\Users\kevin\Documents\Githubdepo\HW\HW1\HW1_1_d.py

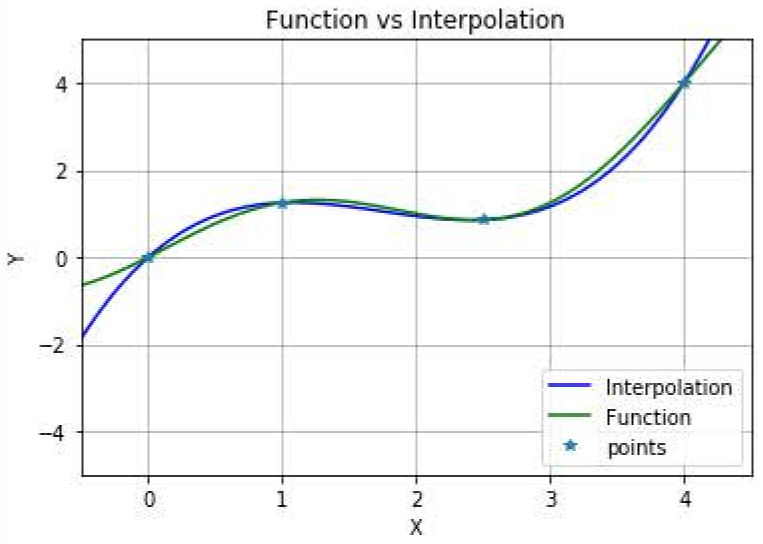
HW1_1_d.py

```python
'''1(d): changed x_pts array from (c)'''
import numpy as np
import math as m

# k is the index of the lagrange polynomial
# pts are the locations where we have fixed values
# x is the point at which we're evaluating the polynomial
def L(k,pts,x):
    prod=1.0 #initial product
    for i in range(len(pts)):
        if (i != k):
            prod=prod*(x-pts[i])/(pts[k]-pts[i])
    return prod
#now, lets define a function and set pts array
def f(x):
    return np.sin((m.pi*x)/2.0)+((x**2)/4)
x_pts=np.array([0.0,1.0,2.5,4.0])
#ploting func using 100 equally spaced pts for x btw -0.5 and 4.5
x=np.linspace(-0.5,4.5,100)
L0=L(0,x_pts,x)
L1=L(1,x_pts,x)
L2=L(2,x_pts,x)
L3=L(3,x_pts,x)
#let's plot the L0,L1,L2,L3!
import matplotlib.pyplot as plt
plt.plot(x, L0, 'b')
plt.plot(x, L1, 'g')
plt.plot(x, L2, 'k')
plt.plot(x, L3, 'r')
plt.axis([-0.5, 4.5, -5, 5])
plt.xlabel('X')
plt.ylabel('Y')
plt.legend(("L0", "L1", "L2","L3"), loc="lower center")
plt.title("Lagrange Polynomials for x0-x4")
#now combine the polynomials
P3=f(x_pts[0])*L0+f(x_pts[1])*L1+f(x_pts[2])*L2+f(x_pts[3])*L3
# This is just the function evaluation
F=f(x)
f_vals = f(x_pts)
#let's plot the P3,F,Pts!
plt.show()
plt.axis([-0.5, 4.5, -5, 5])
plt.xlabel('X')
plt.ylabel('Y')
plt.grid()
plt.plot(x, P3, 'b')
plt.plot(x, F, 'g')
plt.plot(x_pts, f_vals, '*')
plt.title("Function vs Interpolation")
plt.legend(("Interpolation", "Function", "points"), loc="lower right")

"""#1(d)Discuss the differences in how well the function is interpolated using
the different point sets:

As the point interval (between the 2nd and 3rd x-points) becomes smaller
(narrower from plot (d) to plot (c)), the interpolation plot represents more
similarity in shapes to the actual function plot. More data points cause
narrower interval which then provide better smoothness (quality) and accuracy
to an interpolation plot!"""
```

IPython console

Console 1/A





In [6]:

Permissions: RW End-of-lines: CRLF Encoding: UTF-8 Line: 8 Column: 1 Memory: 44 %

Screen-shot of Python for:
-Problem3 (a) plots
-Problem3 (b) discussion (in green text)

Spyder (Python 3.5)

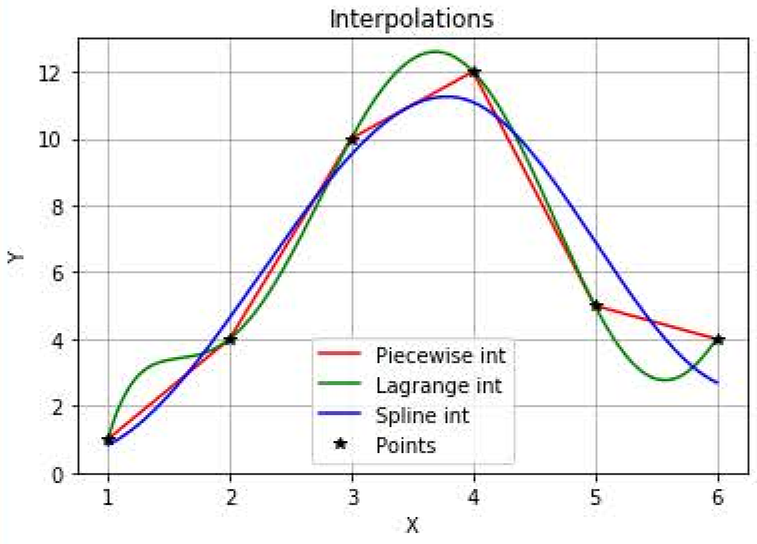
Editor - C:\Users\kevin\Documents\Githubdepo\HW\HW1\HW1_3.py

HW1_3.py | HW1_4.py

```python
# -*- coding: utf-8 -*-
"""
Kevin Choe  HW#1    Due:13 Oct 2017
"""
'''Problem3.(a),(b)'''
#3.(a) use built in funcs to display data pts & plots of 3 diff interpolations
import matplotlib.pyplot as plt
import numpy as np
import scipy.interpolate as interp
# Points provided
x_points =  np.array([1.0, 2.0, 3.0,  4.0,  5.0, 6.0, 7.0])
y_points =  np.array([1.0, 4.0, 10.0, 12.0, 5.0, 4.0, 0.0])
x = np.linspace(1,6,200) # x Range from 1 to 6
# Piecewise linear interpolation
Pp_scipy=interp.interp1d(x_points, y_points) # built-in Piecewise int
PWvals_scipy=Pp_scipy(x)
print('Piecewise Output:')
print(PWvals_scipy)
plt.plot(x, PWvals_scipy,'r', label='Piecewise int')
# Lagrange polynomial interpolation
Pl_scipy = interp.lagrange(x_points, y_points) # built-in Lagrange int
Lvals_scipy=Pl_scipy(x)
print('Lagrange data pts:')
print(Lvals_scipy)
plt.plot(x, Lvals_scipy, 'g',label='Lagrange int')
 # Spline Interpolation
from scipy.interpolate import UnivariateSpline
Ps_scipy = UnivariateSpline(x_points, y_points) # built-in Spline int
Svals_scipy = Ps_scipy(x)
print('Spline Output:')
print(Svals_scipy)
plt.plot(x, Svals_scipy,'b', label='Spline int')
#Let's indicate our points on the plot!
plt.plot(x_points, y_points, 'k*',label='Points')
plt.legend(loc='lower center')
plt.axis([0.75, 6.25, 0, 13])
plt.grid()
plt.xlabel('X')
plt.ylabel('Y')
plt.title("Interpolations")
"""P3.(b) discuss the diff btw the resulting interpolations

In mathematical analysis, smoothness outlines how many function
derivatives are taken and are continuous. As seen in the plot, the spline plot
has the worst smoothness, meaning it lacks number of data points (point
derivatives). Between the Lagrange and Piecewise interpolations,
while both plots are smooth enough to represent quality of data sets, the
Lagrange plot hits all 4 given points whereas the Piecewise plot still
represented the norm of all data points. Thus, Lagrange method best
represented not only the quantity data points (quality=smoothness) but also
the continuity of point derivatives."""
```

IPython console — Console 2/A

```
  7.87767974   7.99987778   8.12092938   8.24076457   8.35931339
  8.47650586   8.59227203   8.70654194   8.8192456    8.93031307
  9.03967436   9.14725953   9.2529986    9.35682161   9.45865859
  9.55843958   9.65609461   9.75155372   9.84474694   9.93560431
 10.02405586  10.11003162  10.19346163  10.27427593  10.35240455
 10.42777752  10.50032488  10.56997667  10.63666291  10.70031365
 10.76085892  10.81822875  10.87235317  10.92316223  10.97058596
 11.01455438  11.05499754  11.09184548  11.12502822  11.1544758
 11.18011825  11.20188562  11.21970792  11.23351521  11.24323751
 11.24880486  11.2501473   11.24719485  11.23987755  11.22812544
 11.21186855  11.19103692  11.16556059  11.13536957  11.10039392
 11.06057056  11.01593956  10.96662043  10.91273467  10.85440383
 10.79174942  10.72489299  10.65395605  10.57906013  10.50032677
 10.41787749  10.33183383  10.2423173   10.14944943  10.05335177
  9.95414582   9.85195313   9.74689522   9.63909362   9.52866986
  9.41574546   9.30044195   9.18288087   9.06318374   8.94147209
  8.81786745   8.69249134   8.5654653    8.43691085   8.30694952
  8.17570284   8.04329234   7.90983954   7.77546598   7.64029319
  7.50444268   7.368036     7.23119466   7.0940402    6.95669414
  6.81927802   6.68191336   6.54472169   6.40782454   6.27134344
  6.13539991   6.00011549   5.86561169   5.73201006   5.59943212
  5.46799939   5.33783341   5.20905571   5.08178781   4.95615124
  4.83226752   4.7102582    4.59024479   4.47234883   4.35669183
  4.24339534   4.13258088   4.02436998   3.91888417   3.81624496
  3.71657391   3.61999252   3.52662233   3.43658487   3.35000167
  3.26699426   3.18768415   3.11219289   3.040642     2.97315301
  2.90984745   2.85084684   2.79627271   2.7462466    2.70089002]
```



In [2]:

IPython console | History log

Permissions: RW End-of-lines: CRLF Encoding: UTF-8 Line: 1 Column: 1 Memory: 45 %

Screen-shot of Python for:
-Problem4(c) k & p values via Least Squares: k=27.0534; p=1.7769
-Problem4(d) k & p values via Curve Fitting: k=11.4322 p=1.5708
-Problem4(e) log-log plot, lin-lin plot for (c) and (d) followed by discussion (in green text)

Spyder (Python 3.5)

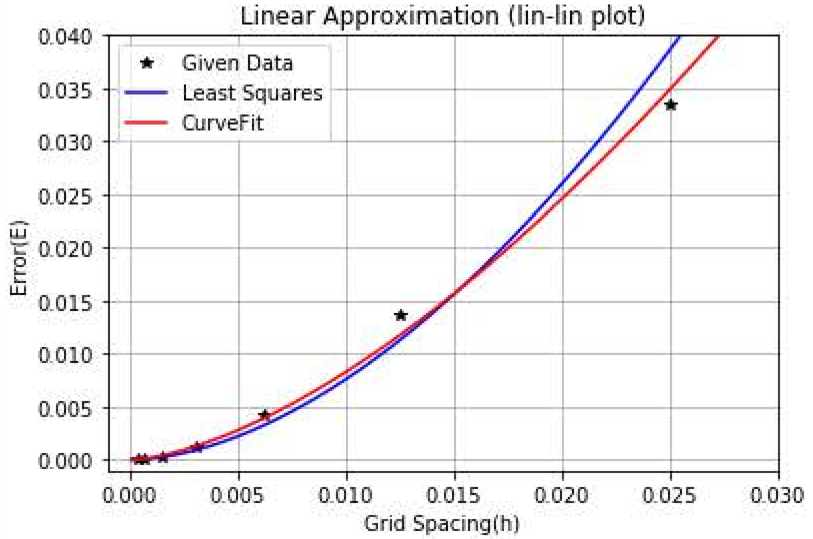
Editor - C:\Users\kevin\Documents\Githubdepo\HW\HW1\HW1_4.py

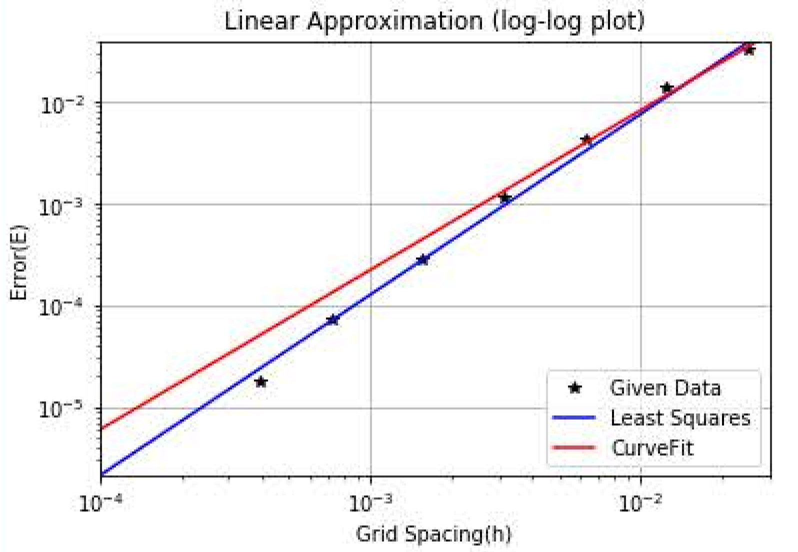
HW1_4.py

```python
ATA=AT*A # A^T*A
ATb=AT*lnEmat # A^T*b
'''Let's Calculate our X matix by using least spares approx '''
Xmat= np.linalg.lstsq(ATA,ATb) # Creates a Tuple Solution
Xvector=Xmat[0]
k=np.exp(np.asscalar(Xvector[0]))#calculate k, not ln(k) by using np.exp
p=np.asscalar(Xvector[1])#0,1 for vector numbers
print("Least Square Approximation")
print("p=",p,"k=",k)

'''HW1 Problem 4 (d)'''
from scipy.optimize import curve_fit
def f(x,K,P):
    return K*x**P #Error function
KPstorage, est=curve_fit(f,h,E)
K=KPstorage[0]
P=KPstorage[1]
print("Curve_fit Approximation")
print("p=",P,"k=",K)

'''HW1 Problem 4 (e)'''
import matplotlib.pyplot as plt
x=np.linspace(-0.5,0.5,5000)#define x for plot

'''lin-lin plot'''
plt.plot(h,E,"k*") #Data plot
LS_E=k*x**p #y data for Least Squares
plt.plot(x,LS_E,"b") #Least Squares plot
CF_E=K*x**P #y data for Curve_fit
plt.plot(x,CF_E,"r") #Curve_fit plot

plt.xlabel("Grid Spacing(h)")
plt.ylabel('Error(E)')
plt.title("Linear Approximation (lin-lin plot)")
plt.legend(("Given Data","Least Squares","CurveFit"),loc="upper left")
plt.axis([-0.001, 0.03, -0.001, 0.04])
plt.grid()

'''log-log plot'''
plt.figure(2)
plt.loglog(h,E,"k*") #Data plot
plt.loglog(x,LS_E,"b") #Least Squares plot
plt.loglog(x,CF_E,"r") #Curve_fit plot
plt.xlabel("Grid Spacing(h)")
plt.ylabel('Error(E)')
plt.title("Linear Approximation (log-log plot)")
plt.legend(("Given Data","Least Squares","CurveFit"),loc="lower right")
plt.axis([-0.001, 0.03, -0.001, 0.04])
plt.grid()

'''P4 (e) discuss the differences btw LS and CF Appx.
Just by looking at the lin-lin plot, it is very hard to tell the difference
between the two approximation methods, Curve Fitting and Least Squares.
However, if we take a look at the log-log plot we can tell that the Least
Squares method approximates more accurate vector of unknowns (its plot is more
precisely shaped  similar to the actual data points. This means that if we get
 more data sets and shorten the spaces between each data point squares, the
Least Squares method would even get a better approximation of vector of
unknowns, in comparison to the Curve fitting method.'''
```

IPython console — Console 2/A

```
Least Square Approximation
p= 1.7769039924081196 k= 27.0534478639
Curve_fit Approximation
p= 1.57085108317 k= 11.432284886
C:/Users/kevin/Documents/Githubdepo/HW/HW1/HW1_4.py:50: RuntimeWarning: invalid value encountered in power
  LS_E=k*x**p #y data for Least Squares
C:/Users/kevin/Documents/Githubdepo/HW/HW1/HW1_4.py:52: RuntimeWarning: invalid value encountered in power
  CF_E=K*x**P #y data for Curve_fit
```





Permissions: RW End-of-lines: CRLF Encoding: UTF-8 Line: 49 Column: 30 Memory: 44 %