Jacqueline K. Garcia
25070118

Feb 13, 2017
NE 155, Slaybaugh

NE 155: Assignment 2

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.

a) $L_k(x) = \prod_{\substack{i=0 \\ i \neq k}}^{n} \frac{x - x_i}{x_k - x_i}$

for x_0; $L_0(x) = \frac{(x-x_1)(x-x_2)(x-x_3)}{(x_0-x_1)(x_0-x_2)(x_0-x_3)}$ for x_1; $L_1(x) = \frac{(x-x_0)(x-x_2)(x-x_3)}{(x_1-x_0)(x_1-x_2)(x_1-x_3)}$

for x_2; $L_2(x) = \frac{(x-x_0)(x-x_1)(x-x_3)}{(x_2-x_0)(x_2-x_1)(x_2-x_3)}$ for x_3; $L_3(x) = \frac{(x-x_0)(x-x_1)(x-x_2)}{(x_3-x_0)(x_3-x_1)(x_3-x_2)}$

b) The polynomial is given by:

$$P_3(x) = f(x_0)L_0(x) + f(x_1)L_1(x) + f(x_2)L_2(x) + f(x_3)L_3(x) =$$

$$P_3(x) = f(x_0)\frac{(x-x_1)(x-x_2)(x-x_3)}{(x_0-x_1)(x_0-x_2)(x_0-x_3)} + f(x_1)\frac{(x-x_0)(x-x_2)(x-x_3)}{(x_1-x_0)(x_1-x_2)(x_1-x_3)}$$
$$+ f(x_2)\frac{(x-x_0)(x-x_1)(x-x_3)}{(x_2-x_0)(x_2-x_1)(x_2-x_3)} + f(x_3)\frac{(x-x_0)(x-x_1)(x-x_2)}{(x_3-x_0)(x_3-x_1)(x_3-x_2)}$$

c) $x_0 = 0, x_1 = 2, x_2 = 3, x_3 = 4, f(x) = \sin\left(\frac{\pi}{2}x\right) + \frac{x^2}{4}$

for x_0; $f(x_0) = 0$, $L_0(x) = \frac{(x-2)(x-3)(x-4)}{-24}$ for x_1; $f(x_1) = 1$, $L_1(x) = \frac{(x)(x-3)(x-4)}{(2)(-1)(-2)}$

Jacqueline K. Garcia
25070118

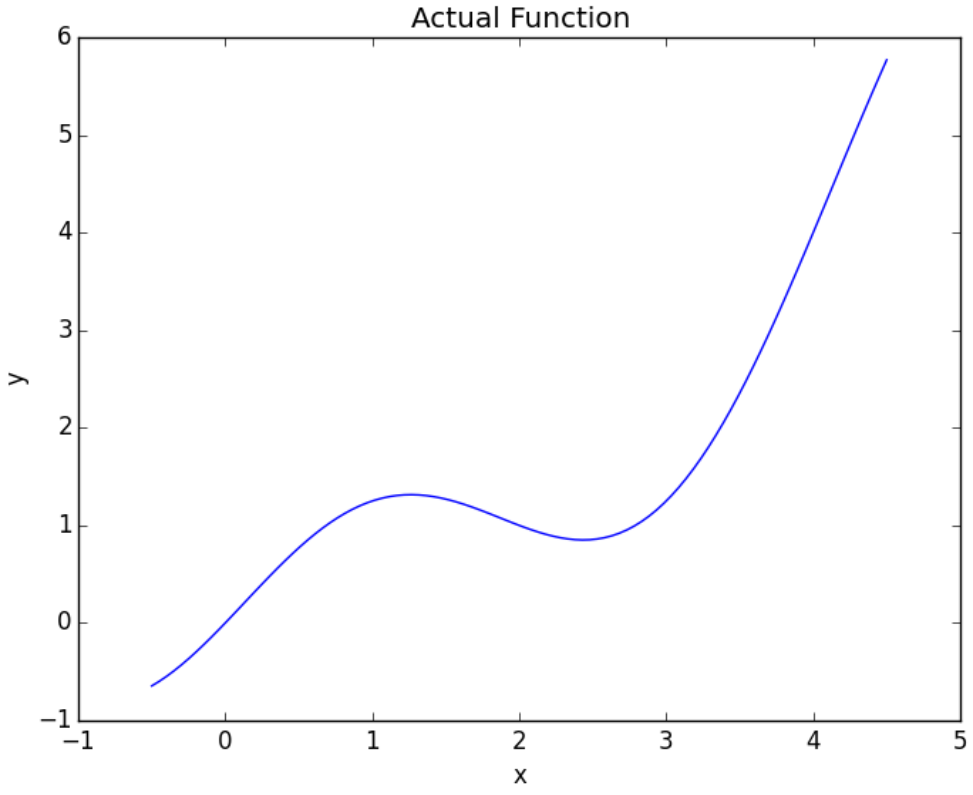
for $x_2; f(x_2) = \frac{5}{4},\ L_2(x) = \frac{(x)(x-2)(x-4)}{(3)(1)(-1)}$ for $x_3;\ f(x_3) = 4,\ L_3(x) = \frac{(x)(x-2)(x-3)}{(4)(2)(1)}$

$$\therefore P_3(x) = \frac{x^3 - 7x^2 + 12x}{4} + \frac{5(x^3 - 6x^2 + 8x)}{-12} + \frac{4(x^3 - 5x^2 + 6x)}{8} =$$

$$P_3(x) = \frac{x^3}{3} - \frac{7}{4}x^2 + \frac{8}{3}x$$

NE_155_q1
Code to generate plot of functions

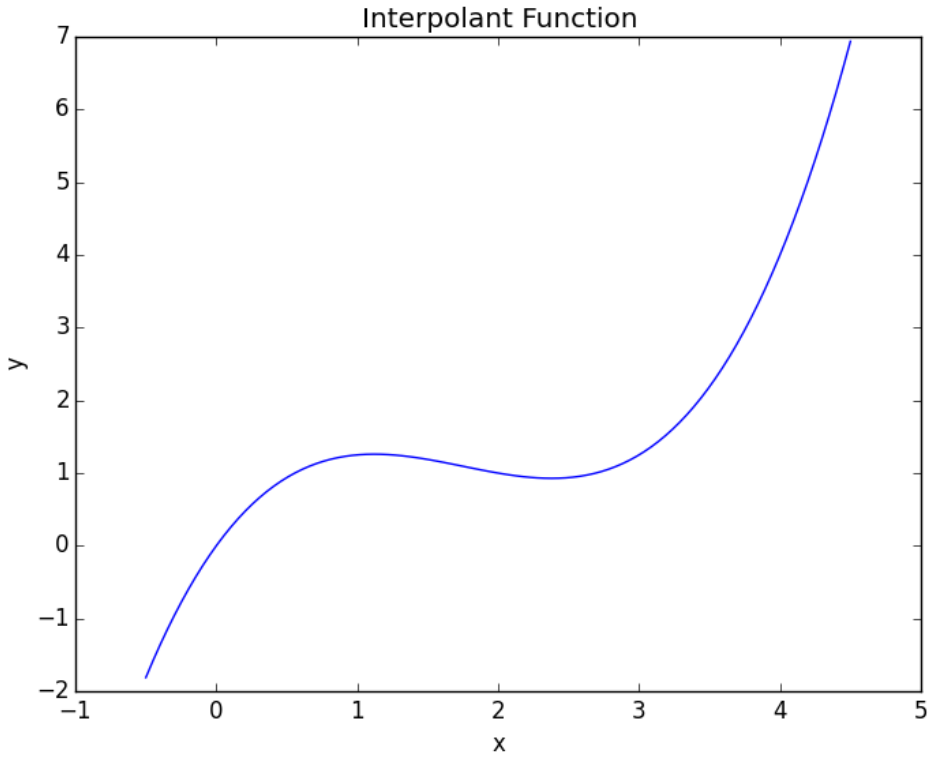
```
import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(-0.5, 4.5, 100)
def y_actual(x):
    return np.sin((np.pi / 2) * x) + (x ** 2) / 4

y_a = []
for i in x:
    y_a.append(y_actual(i))

plt.xlabel('x')
plt.ylabel('y')
plt.title('Actual Function')
plt.plot(x, y_a)
plt.show()
```



Jacqueline K. Garcia
25070118

NE_155_q1
Code to generate plot of functions

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

## Interpolant Function

def y_inter(x):
    '''
    interpolant function
    '''
    return (x ** 3) / 3 - (7 / 4) * x ** 2 + (8 / 3) * x

y_in = []
for i in x:
    y_in.append(y_inter(i))

plt.xlabel('x')
plt.ylabel('y')
plt.title('Interpolant Function')
plt.plot(x, y_in)
plt.show()
```



d) $x_0 = 0,\ x_1 = 1,\ x_2 = 2.5,\ x_3 = 4$

Jacqueline K. Garcia
25070118

for $x_0; f(x_0) = 0,\ L_0(x) = \frac{(x-1)(x-2.5)(x-4)}{(-1)(-2.5)(-4)}$ for $x_1;\ f(x_1) = \frac{5}{4},\ L_1(x) = \frac{(x)(x-2.5)(x-4)}{(1)(-1.5)(-3)}$

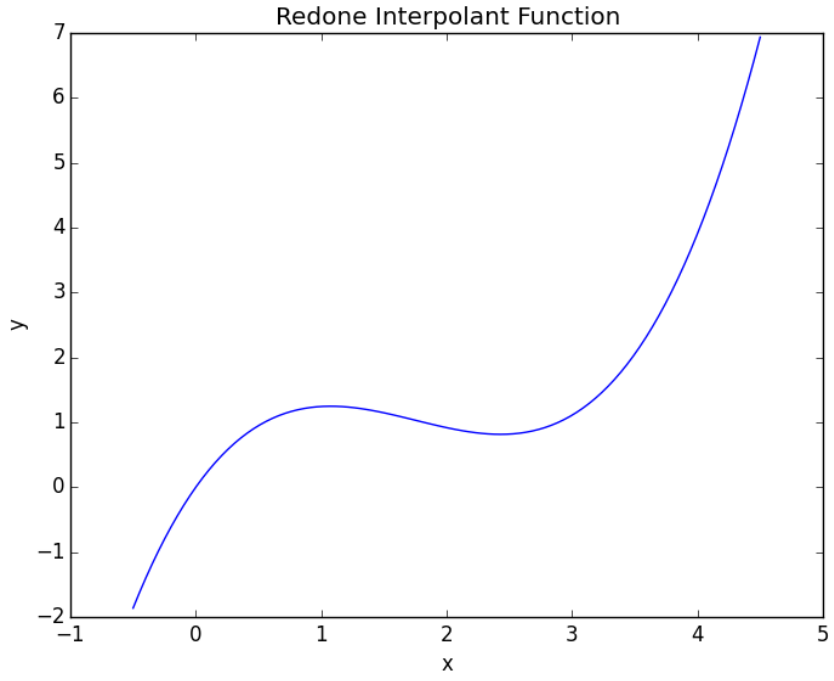
for $x_2; f(x_2) = 0.8554, L_2(x) = \frac{(x)(x-1)(x-4)}{(2.5)(1.5)(-1.5)}$, for $x_3; f(x_3) = 4,\ L_3(x) = \frac{(x)(x-1)(x-2.5)}{(4)(3)(1.5)}$

$$P_3(x) = \frac{5}{4}\frac{(x^3 - 6.5x^2 + 10x)}{(4.5)} + .8554\frac{x^3 - 5x^2 + 4x}{-5.625} + 4\frac{x^3 - 3.5x^2 + 2.5x}{18}$$

$$P_3(x) = 0.348x^3 - 1.829x^2 + 2.725x$$

NE_155_q1
Code to generate plot of functions

```
## Redone Interpolant

def y_reinter(x):
    '''
    redone interpolant function
    '''
    return (0.348 * x ** 3) - (1.829 * x ** 2) + 2.725 * x

y_rein = []
for i in x:
    y_rein.append(y_reinter(i))

plt.xlabel('x')
plt.ylabel('y')
plt.title('Redone Interpolant Function')
plt.plot(x, y_rein)
plt.show()
```



The function is interpolated slightly better (it looks more like the original function) in certain areas, however, the difference between the previous interpolation and the new one is not that notable. Also, the first interpolation and the second interpolation appear to match up to the original function in separate areas.

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 the x that maximizes the expression, and present one final number. You may use a mathematical package to assist you in this calculation.

a) $err(x) = |f(x) - P_3(x)| = \left|\frac{f^{n+1}(\xi)}{(n+1)!}\prod_{i=0}^{n}(x - x_i)\right| = \left|\frac{f^4(\xi)}{4!}(x - x_0)(x - x_1)(x - x_2)(x - x_3)\right|$

b) $f(x) = \sin\left(\frac{\pi}{2}x\right) + \frac{x^2}{4} \rightarrow differentiate! \rightarrow f'(x) = \frac{\pi}{2}\cos\left(\frac{\pi}{2}x\right) + \frac{x}{2} \rightarrow differentiate! \rightarrow$

$$f''(x) = -\left(\frac{\pi}{2}\right)^2 \sin\left(\frac{\pi}{2}x\right) + \frac{1}{2} \rightarrow differentiate! \rightarrow f'''(x) = -\left(\frac{\pi}{2}\right)^3 \cos\left(\frac{\pi}{2}x\right)$$
$$\rightarrow differentiate! \rightarrow f^{iv}(x) = \left(\frac{\pi}{2}\right)^4 \sin\left(\frac{\pi}{2}x\right)$$

total error function: $err(x) = \left|\left(\frac{\left(\frac{\pi}{2}^4 \sin\left(\frac{\pi}{2}x\right)\right)}{24}\right)(x - x_0)(x - x_1)(x - x_2)(x - x_3)\right|$

The $f^{iv}(x)$ function is at a maximum at x = 1 and is at a minimum at x = 3. At x = 1, the function $f^{iv}(x)$ is equal to $\left(\frac{\pi}{2}\right)^4$ and at $x = 3$, the function is equal to $-\left(\frac{\pi}{2}\right)^4$. In terms of magnitude, the function is at 0 for the values 0, 2, and 4.

c) Error(x) = $\left|\frac{f^4(\xi)}{4!}(x - x_0)(x - x_1)(x - x_2)(x - x_3)\right|$

$$x_0 = 0, \quad x_1 = 2, \quad x_2 = 3, \quad x_3 = 4$$

The error becomes:

$$err(x) = \left|\frac{f^4(\xi)}{4!}(x)(x - 2)(x - 3)(x - 4)\right| = \left|\frac{\left(\frac{\pi}{2}\right)^4 \sin\left(\frac{\pi}{2}x\right)}{24}(x^4 - 9x^3 + 26x^2 - 24x)\right|$$

After optimizing this function utilizing Mathematica, the largest magnitude error is obtained at x = 0.830691 where the error is |-1.63448| = 1.63448.

Jacqueline K. Garcia
25070118

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.

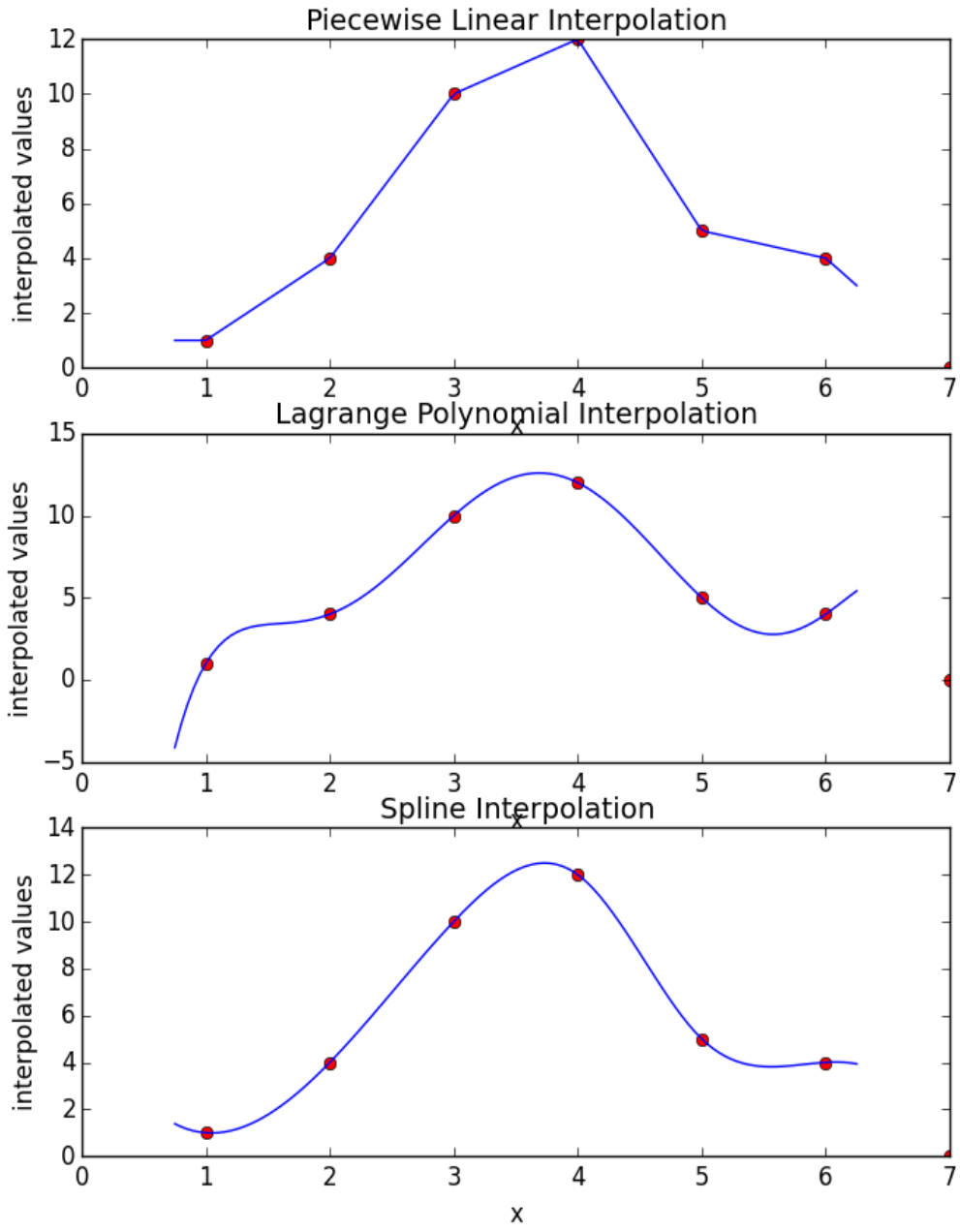
(b) (5 points) Briefly discuss the differences between the resulting interpolations.

a) Using Python:

NE_155_q3
Code used to generate three subplots: one for each interpolating method

```
import numpy as np
import matplotlib.pyplot as plt
import  scipy.interpolate

## Interpolating using Python's built-in functions

x = np.linspace(0.75, 6.25, 110)

xp = [1, 2, 3, 4, 5, 6, 7]
fp = [1, 4, 10, 12, 5, 4, 0]

## piecewise-linear-interpolation component
pli = np.interp(x, xp, fp)

plt.subplot(311)
plt.xlabel('x')
plt.ylabel('interpolated values')
plt.title('Piecewise Linear Interpolation')
plt.plot(xp, fp, 'ro')
plt.plot(x, pli)

## Lagrange-Polynomial-interpolation component

lpi = scipy.interpolate.lagrange(xp, fp)

y_lpi = []

for i in x:
    y_lpi.append(lpi(i))

plt.subplot(312)
```

```
plt.xlabel('x')
plt.ylabel('interpolated values')
plt.title('Lagrange Polynomial Interpolation')
plt.plot(xp, fp, 'ro')
plt.plot(x, y_lpi)

## Spline-interpolation component (cubic)

data_points = scipy.interpolate.splrep(xp, fp)
si = scipy.interpolate.splev(x, data_points)

plt.subplot(313)
plt.xlabel('x')
plt.ylabel('interpolated values')
plt.title('Spline Interpolation')
plt.plot(xp, fp, 'ro')
plt.plot(x, si)
plt.show()
```



b) The piecewise linear interpolation simply connects the data points directly and linearly. The Lagrange interpolation creates a polynomial function that passes through all the data points. The spline interpolation behaves like a cubic function between each pair of data points so that the end result is a combination of connected cubic functions. The difference between the three different interpolation methods is the end behavior. This is due to the difference in equations between all of them. For the linear interpolation, the ends follow the linear trend and therefore do not curve. The Lagrange interpolation graph shows the first end curving downward and the last end curving

upward because the end points follow the trend of the polynomial function. For the spline interpolation, the endpoints behave like cubic functions. Therefore, for the first end point, the graph curves upward because it is following the trend of the function from the first data point. The last endpoint curves downward because of the cubic function the last data point goes by.

4. (20 points) The errors generated by a numerical method on a test problem with various grid resolutions have been recorded in the following table:

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 $x = \begin{pmatrix} \log(k) \\ p \end{pmatrix}$ is the vector of unknowns.

(b) (5 points) Derive the normal equations for this over-determined system.

(c) (9 points) Solve the normal equations to obtain a least squares estimate to the parameters k and p.

(d) (3 points) Make a log-log plot that displays both the input data and the function $E = kh^p$. (Checkout Python's `matplotlib.pyplt.loglog` or MATLAB's `loglog` command)

a) $k = \frac{E}{h^p} \rightarrow \log(k) = \log\left(\frac{E}{h^p}\right) = \log(E) - \log(h^p) = \log(E) - P * \log(h) \rightarrow$

$$\log(E) = \log(k) + P * \log(h)$$

In matrix format:

$$\begin{bmatrix} 1 & \log(h_1) \\ \vdots & \vdots \\ 1 & \log(h_8) \end{bmatrix} \begin{bmatrix} \log(k) \\ P \end{bmatrix} = \begin{bmatrix} \log(E_1) \\ \vdots \\ \log(E_8) \end{bmatrix}$$

b) $$A^T \begin{bmatrix} 1 & \log(h_1) \\ \vdots & \vdots \\ 1 & \log(h_8) \end{bmatrix} \begin{bmatrix} \log(k) \\ P \end{bmatrix} = A^T \begin{bmatrix} \log(E_1) \\ \vdots \\ \log(E_8) \end{bmatrix} =$$

$$\begin{bmatrix} 1 & \cdots & 1 \\ \log(h_1) & \cdots & \log(h_8) \end{bmatrix} \begin{bmatrix} 1 & \log(h_1) \\ \vdots & \vdots \\ 1 & \log(h_8) \end{bmatrix} \begin{bmatrix} \log(k) \\ P \end{bmatrix} = \begin{bmatrix} 1 & \cdots & 1 \\ \log(h_1) & \cdots & \log(h_8) \end{bmatrix} \begin{bmatrix} \log(E_1) \\ \vdots \\ \log(E_8) \end{bmatrix} =$$

$$\begin{bmatrix} 8 & \sum_{i=1}^{8} \log(h_i) \\ \sum_{i=1}^{8} \log(h_i) & \sum_{i=1}^{8} \log^2(h_i) \end{bmatrix} \begin{bmatrix} \log(k) \\ P \end{bmatrix} = \begin{bmatrix} \sum_{i=0}^{8} \log(E_i) \\ \sum_{i=0}^{8} \log(h_i)\log(E_i) \end{bmatrix}$$

c) $\text{inv}(A^T A) * (A^T A) * x = \text{inv}(A^T A) * A^T * b$

$$\therefore x = inv(A^T A) * A^T * b$$

NE_155_q4
Code that solves for the values "k" and "P"

```
import numpy as np

h = [5 * 10 ** (-2), 2.5 * 10 ** (-2), 1.25 * 10 ** (-2), 6.25 * 10 **
(-3),
     3.125 * 10 ** (-3), 1.5625 * 10 ** (-3), 7.8125 * 10 ** (-4),
     3.90625 * 10 ** (-4)]

E = [1.036126 * 10 ** (-1), 3.333834 * 10 ** (-2), 1.375409 * 10 ** (-
2),
     4.177237 * 10 ** (-3), 1.103962 * 10 ** (-3), 2.824698 * 10 ** (-
4),
     7.185644 * 10 ** (-5), 1.813937 * 10 ** (-5)]

log_h = np.log(h)
log_E = np.log(E)

A = np.zeros((8, 2))
for i in range(8):
    A[i][0] = 1
    A[i][1] = log_h[i]

b = np.zeros((8,1))
for i in range(8):
    b[i][0] = log_E[i]

x_A     =     np.dot((np.linalg.inv(np.dot(np.transpose(A),     A))),
np.transpose(A))
x = np.dot(x_A, b)

p = x[1][0]

k = np.exp(x[0][0])

print("The value of p is {}".format(p))
print("The value of k is {}".format(k))
```

Returns:

```
In [35]: run NE_155_q4
The value of p is 1.790291393986538
The value of k is 28.432770995208546
```

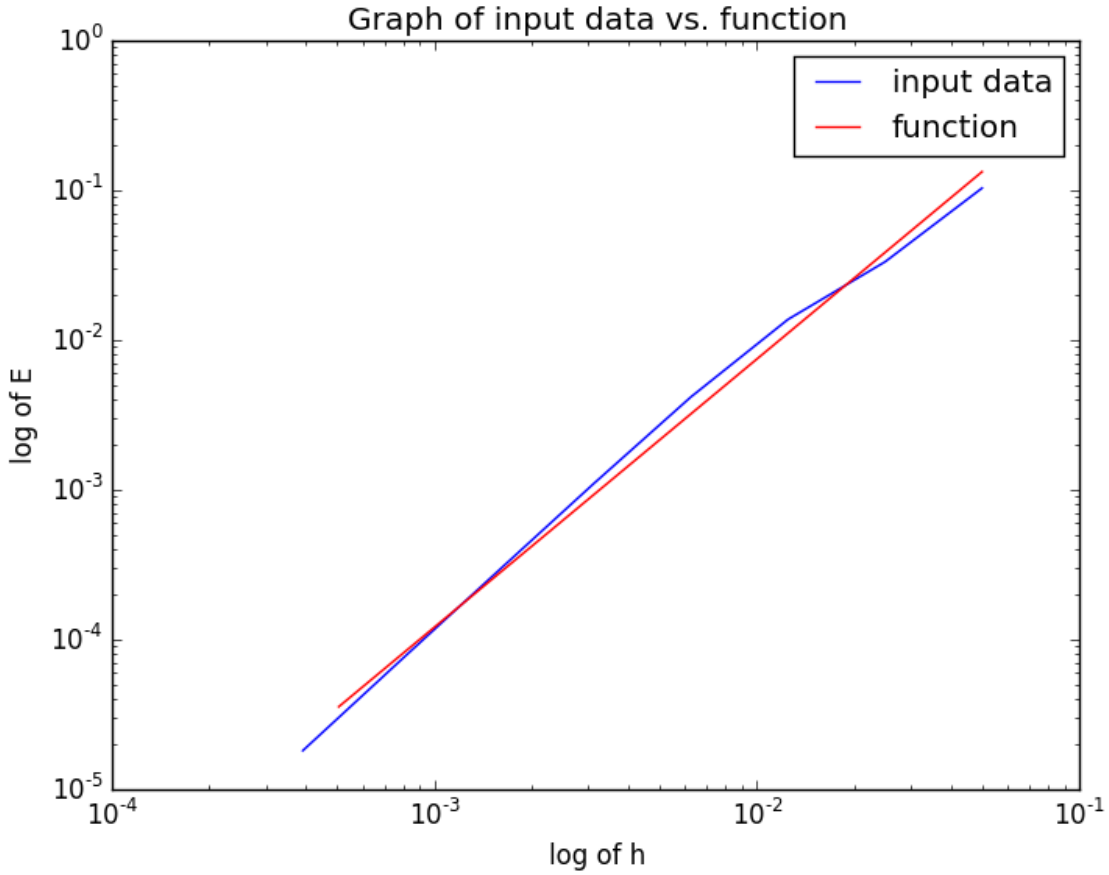
d) .

NE_155_q4

Code generated the plot containing input values and actual function

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

h = [5 * 10 ** (-2), 2.5 * 10 ** (-2), 1.25 * 10 ** (-2), 6.25 *
10 ** (-3),
     3.125 * 10 ** (-3), 1.5625 * 10 ** (-3), 7.8125 * 10 ** (-
4),
     3.90625 * 10 ** (-4)]

E = [1.036126 * 10 ** (-1), 3.333834 * 10 ** (-2), 1.375409 * 10
** (-2),
     4.177237 * 10 ** (-3), 1.103962 * 10 ** (-3), 2.824698 * 10
** (-4),
     7.185644 * 10 ** (-5), 1.813937 * 10 ** (-5)]

log_h = np.log(h)
log_E = np.log(E)

A = np.zeros((8, 2))
for i in range(8):
    A[i][0] = 1
    A[i][1] = log_h[i]

b = np.zeros((8,1))
for i in range(8):
    b[i][0] = log_E[i]

x_A   =   np.dot((np.linalg.inv(np.dot(np.transpose(A),   A))),
np.transpose(A))
x = np.dot(x_A, b)

p = x[1][0]

k = np.exp(x[0][0])

#print("The value of p is {}".format(p))
#print("The value of k is {}".format(k))

plt.loglog(h, E)

def function(k, p, h):
    return k * h ** p

axis = np.linspace(0, .05, 100)

values = []
for i in axis:
    values.append(function(k, p, i))

plt.loglog(axis, values, "r")
plt.xlabel('log of h')
```

```
      plt.ylabel('log of E')
      plt.title('Graph of input data vs. function')
      plt.legend(("input data", "function"))
plt.show()
```



BONUS: 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`.