Computational Sciences and Informatics Course 690

Assignment 3 (REVISED)

Jericho McLeod

Introduction:

The following problems and solutions reflect studies designed to reinforce course lectures pertaining to the fitting and use of linear systems to solve problems, as well as linear straight line and polynomial regression. Emphasis has been placed on comparisons of methods related to the introduction of error based upon the direction and type of regression.

Methods:

The assigned problems are from Scientific Computing, by Michael T. Heath, are Chapter 3's first and second exercises, as well as a selection from Numerical Methods for Engineers, by Steven Chapra and Raymond Canale, specifically; Chapter 17's third and sixth problems. These will henceforth be referred to respectively as problems one through four.

Problem one related to iteratively fitting a polynomial of *n* degrees to a dataset and observing the fit. This was accomplished using python code available in the appendix.

The second problem involved setting up a least-squares problem for an overdetermined system (m>n). This was done via the $Ax \cong b$ layout that is frequent for this type of problem, and the linear system was solved using python's linear algebra library.

The third problem fit a least squares regression to data presented as two vectors, and in addition to the slope an intercept to find the standard error of the estimate as well as the correlation coefficient, then to repeat the previous steps with the variables reversed and interpret the results.

The least squares problem was solved using the previously provided methods. Standard error was obtained using the following formula:

$$SE = \frac{\sigma}{\sqrt{n}} = \sqrt{\frac{S}{n}} \text{ where } S = \sum_{i=1}^{n}(y_i - a_0 - a_1 x_i)^2$$

The correlation coefficient was used using Pearson's Correlation, given as follows:

$$r = \frac{cov(x,y)}{\sqrt{var(x)} * \sqrt{var(y)}}$$

(Kent State University, 2018)

Which, when substituting the formula for covariance and variance for a more complete formula, is:

$$r = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}}$$

The fourth problem primarily used the same methods but added a new component in regression the data to a polynomial rather than a straight line. The form for this is given as:

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

Given that the problem requested that the problem be fitted to a parabola only a second order equation was used, the familiar linear form:

$$Ax = b$$

Can be obtained using the matrix system:

$$\begin{bmatrix} \sum x_i^0 & \sum x_i^1 & \ldots & \sum x_i^n \\ \sum x_i^1 & \sum x_i^2 & \ldots & \sum x_i^{n+1} \\ \vdots & \vdots & \ddots & \vdots \\ \sum x_i^n & \sum x_i^{n+1} & \ldots & \sum x_i^{n+n} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} \sum x_i^0 y_i \\ \sum x_i^1 y_i \\ \sum x_i^2 y_i \end{bmatrix}$$

From which point a solution introduces no additional methods, although the python script shown in the appendix was used for the exponential summations and to solve the linear system

Results:

Problem 1 (Heath 3.1)

For *n* = 1, 2 .. 5, fit a polynomial of degree *n* by least squares to the following data:

t|	0.0	1.0	2.0	3.0	4.0	5.0
y|	1.0	2.7	5.8	6.6	7.5	9.9

Make a plot of the original data points along with each resulting polynomial curve. Which polynomial captures the trend of the data better? Explain your assumptions in answering.

For the polynomial order 0, the expression coefficients are: [5.58333333]. This is not an accurate representation of the data at, as can be seen in figure 1.

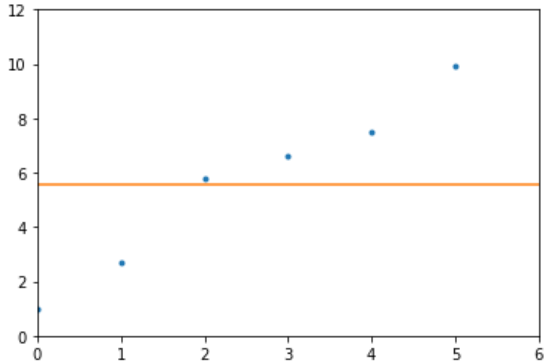

Figure 1

The first order polynomial expression, y = 1.70571429x + 1.31904762, fits the trend of the data fairly well, but does not minimize residuals, as seen in figure 2.

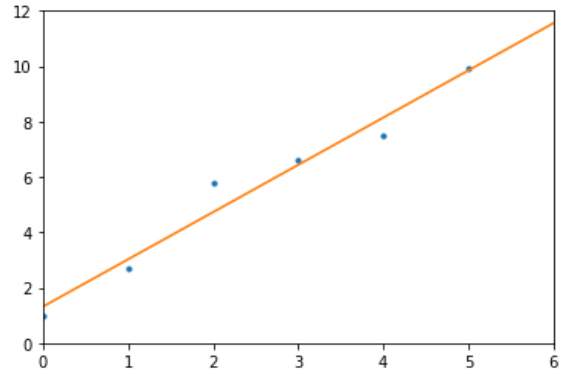

Figure 2

The second order polynomial does not offer significant improvement over the first. Its expression is y = -0.09464286x^2 + 2.17892857x + 1.00357143, and can be seen in figure 3.

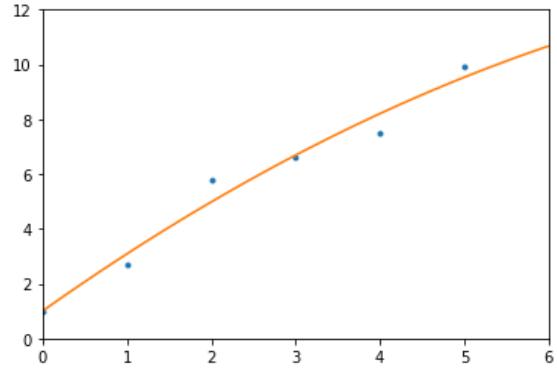

Figure 3

The third order polynomial expression, y = 0.0712963x^3 - 0.62936508x^2 + 3.15568783x 0.78968254, represents the data fairly well. It captures the trend , lowers residuals over prior equations, but does not introduce overly much extraneous wiggle, as seen in figure 4.

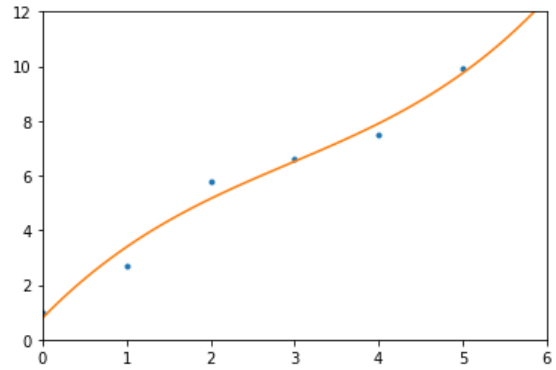

Figure 4

The fourth order polynomial expression, $y = 0.10625x^4 - 0.9912037x^3 + 2.63402778x^2 + 0.11997354x + 0.9718254$, very nearly mitigates residuals, but begins to wildly exaggerate the trend beyond the given data points.

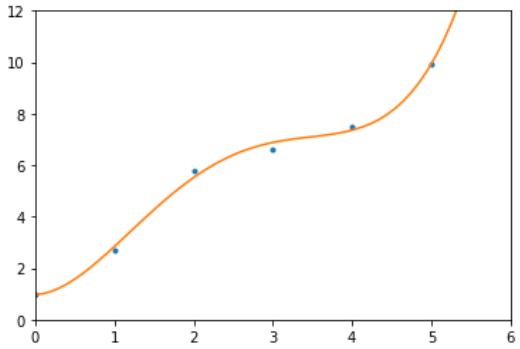

Figure 5

And, finally, the fifth order polynomial expression, $y = -0.05916667x^5 + 0.84583333x^4 - 4.2125x^3 + 8.30416667x^2 - 3.17833333x + 1$, perfectly aligns with the five data points, but does not adequately capture the trend represented. Even between points x=0 and x=1, the trend does not match, and for points where x>5, the apparent trend is positive while this formula is negative, as shown in figure 6. The wiggle introduced by this order polynomial makes the expression far less good as an expression to explain the relationship between x and y.

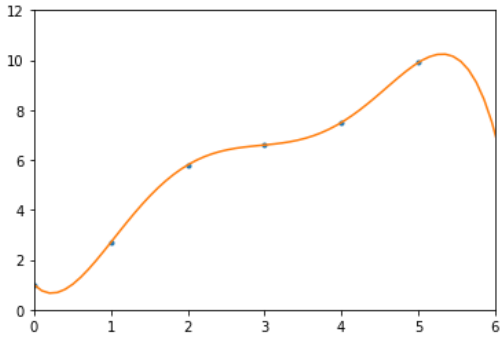

Figure 6

Problem 2 (Heath 3.2)

Given:

$x_1 = 2.95$ $x_2 = 1.74$

$x_3 = -1.45$ $x_4 = 1.32$

$x_1 - x_2 = 1.23$ $x_1 - x_3 = 4.45$

$x_1 - x_4 = 1.61$ $x_2 - x_3 = 3.21$

$x_2 - x_4 = 0.45$ $x_3 - x_4 = -2.75$

Set up a corresponding least squares problem $Ax \cong b$ and use a library routine or one of my own design to solve it for the best values. How do the calculated values compare to the direct measurements?

$$Ax = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & -1 & 0 & 0 \\ 1 & 0 & -1 & 0 \\ 1 & 0 & 0 & -1 \\ 0 & 1 & -1 & 0 \\ 0 & 1 & 0 & -1 \\ 0 & 0 & 1 & -1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} \cong \begin{bmatrix} 2.95 \\ 1.74 \\ -1.45 \\ 1.32 \\ 1.23 \\ 4.45 \\ 1.61 \\ 3.21 \\ 0.45 \\ -2.75 \end{bmatrix} = b$$

Solving this system with python yields the x^T vector [2.96, 1.746, -1.46, 1.314]. Compared to the original values, these are all relatively close, with variances of at most 1/100th.

Problem 3 (Chapra Canale 17.3)

A: Use least squares regression to fit a straight line to:

X: 0, 2, 4, 6, 9, 11, 12, 15, 17, 19,

Y: 5, 6, 7, 6, 9 8, 7, 10, 12, 12,

Along with the slope and intercept, compute the standard error of the estimate and the correlation coefficient. Plot the data and the regression line.

x_i	y_i	$x_i y_i$	x_i^2	Var
0	5	0	0	0.022042
2	6	12	4	0.196714
4	7	28	16	0.545507
6	6	36	36	0.933842
9	9	81	81	0.953035
11	8	88	121	0.531011
12	7	84	144	4.331289
15	10	150	225	0.019206
17	12	204	289	1.337435
19	12	228	361	0.203884
Σ = 95	Σ = 82	Σ = 911	Σ = 1277	Σ = 9.073965287

Fitting the least squares using the same formulas as the prior problem:

$y = a_0 + a_1 x$

$a_1 = \frac{n\sum x_i y_i - \sum x_i \sum y_i}{n\sum x_i^2 - (\sum x_i)^2}$

$a_0 = \bar{y} - a_1\bar{x}$

Yields:

$a_1 = \frac{10(911)-95(82)}{10(1277)-95^2} = \frac{1320}{3745}$ = 0.35246996

$a_0 = 8.2 - 0.35246996\ (9.5)$ = 4.85153538

Thus, $y = 4.85153538 + 0.35246996x$

The standard error can be defined as:

$SE = \frac{\sigma}{\sqrt{n}} = \sqrt{\frac{S}{n}}$ where $S = \sum_{i=1}^{n}(y_i - a_0 - a_1x_i)^2$

Since we have little information regarding the sample, we will use n-1 in the SSE formula.

Thus:

$SE = \sqrt{\frac{9.073965287}{9}} = 1.00410077$

For the correlation coefficient we used Pearson's formula:

$$r = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}}$$

Thus: $r = \frac{132}{\sqrt{374.5}\sqrt{55.6}}$ = 0.91476728

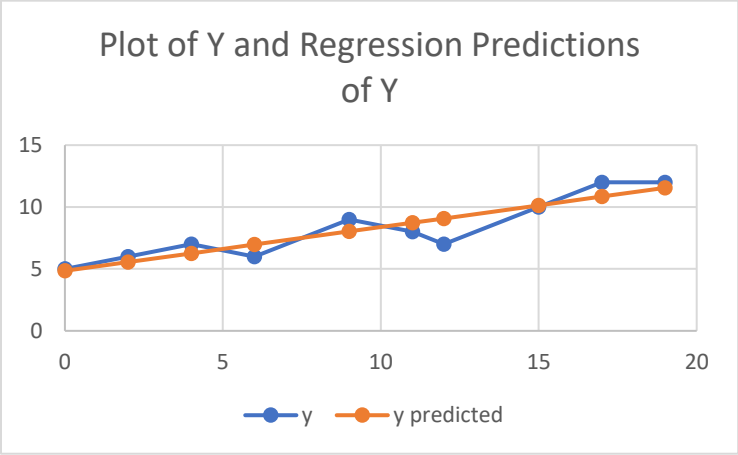

Figure 7

Figure 7 shows the relationship of the predicted values to the observed values. The error observed is the vertical distance between the observations and regressed line of the predicted values.

B: Then repeat the problem, but regress x versus y: that is, switch the variables. Interpret your results.

x_i	y_i	$x_i y_i$	x_i^2	sse
5	0	0	25	3.620944
6	2	12	36	5.184631
7	4	28	49	7.028221
6	6	36	36	2.968803
9	9	81	81	5.756547
8	11	88	64	3.899915
7	12	84	49	28.61095
10	15	150	100	1.504593
12	17	204	144	2.315214
12	19	228	144	0.228883
Σ = 82	Σ = 95	Σ = 911	Σ = 728	Σ = 61.11870504

Fitting the least squares using the same formulas as the prior problem:

$y = a_0 + a_1 x$

$a_1 = \frac{n\sum x_i y_i - \sum x_i \sum y_i}{n\sum x_i^2 - (\sum x_i)^2}$

$a_0 = \bar{y} - a_1 \bar{x}$

Yields:

$a_1 = \frac{10(911) - 95(82)}{10(728) - 82^2} = \frac{1320}{3745}$ = 2.37410072

$a_0 = 9.5 - 2.37410072(8.2)$ = -9.96762590

Thus, $y = -9.96762590 + 2.37410072x$

The standard error can be defined as:

$SE = \frac{\sigma}{\sqrt{n}} = \sqrt{\frac{S}{n}}$ where $S = \sum_{i=1}^{n}(y_i - a_0 - a_1 x_i)^2$

Since we have little information regarding the sample, we will use n-1 in the SSE formula.

Thus:

$SE = \sqrt{\frac{61.11870504}{9}} = 2.60594843$

For the correlation coefficient we used Pearson's formula:

$r = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}}$

However, one will note that due to the commutative property this is unchanged when X and Y are exchanged, thus the Pearson's coefficient remains the same: 0.91476728.

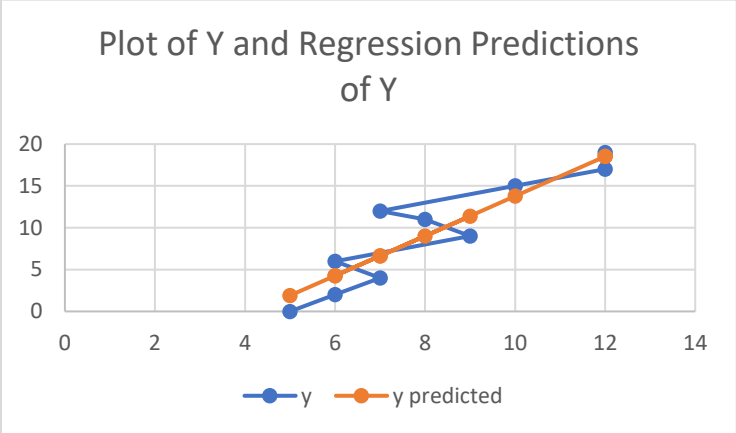

Figure 8

Figure 8 showed the results of reversing the values. The error is now the horizontal distance to the regressed line. It should be noted that the observed error is distorted due to the lack of scale between X and Y: the increased distance between points along the X axis exaggerates the visually apparent error in this model, although it does present with relatively greater error.

Reviewing the models, the error present in each represents the residual values observed between the regressed line and the observed values based on the axis being interpreted. This causes them to have different amounts of error and to create slightly different regressed lines when plotted along the same values, as shown in the following chart, figure 9.

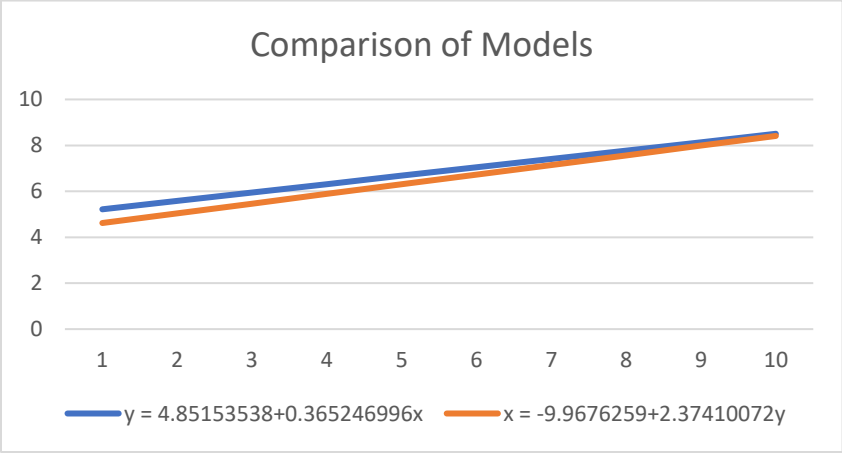

Figure 9

Problem 4 (Chapra Canale 17.6)

Use least-squares regression to fit a straight line to:

X: 1, 2, 3, 4, 5, 6, 7, 8, 9,

Y: 1, 1.5, 2, 3, 4, 5, 8, 10, 13,

A: Along with the slope and intercept, compute the standard error of the estimate and the correlation coefficient. Plot the data and the straight line. Assess the fit.

x_i	y_i	$x_i y_i$	x_i^2	Var
1	1	1	1	2.41975309
2	1.5	3	4	0.35667438
3	2	6	9	0.13040123
4	3	12	16	0.67148920
5	4	20	25	1.63271605
6	5	30	36	3.01408179
7	8	56	49	0.03780864
8	10	80	64	0.12056327
9	13	117	81	3.56790123
Σ = 45	Σ = 47.5	Σ = 325	Σ = 285	Σ = 11.98138889

Fitting the least-squares regression to a straight line:

$y = a_0 + a_1 x$

$a_1 = \frac{n\sum x_i y_i - \sum x_i \sum y_i}{n\sum x_i^2 - (\sum x_i)^2}$

$a_0 = \bar{y} - a_1 \bar{x}$

Yields:

$a_1 = \frac{9(325) - 45(47.5)}{9(285) - 45^2} = \frac{787.5}{540} = 1.45833333$

$a_0 = 9.5 - 2.37410072(8.2) = -2.01388889$

Thus, $y = -2.01388889 + 1.45833333\text{x}$

Standard Error:

$SE = \frac{\sigma}{\sqrt{n}} = \sqrt{\frac{S}{n}}$ where $S = \sum_{i=1}^{n}(y_i - a_0 - a_1 x_i)^2$

Since we have little information regarding the sample, we will use n-1 in the SSE formula.

Thus:

$SE = \sqrt{\frac{11.98138889}{8}} = 1.22226168$

For the correlation coefficient we used Pearson's formula:

$$r = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}}$$

Thus: $r = \frac{87.5}{\sqrt{60}\sqrt{139.555}}$ = 0.95622229

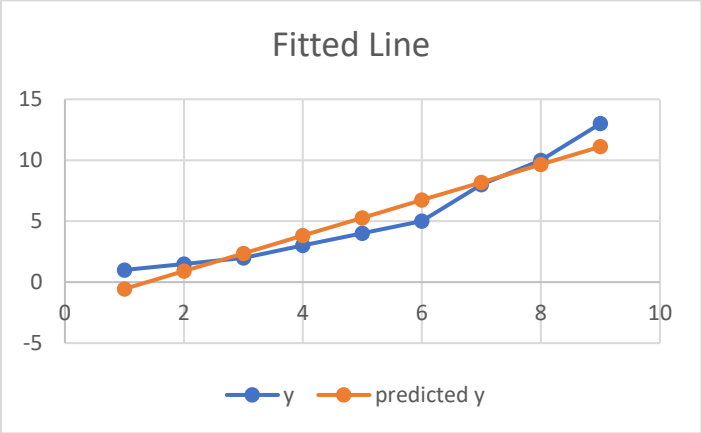

Figure 10

The fit shown in figure 10 is deficient, as the observed line is clearly curved whereas the linear regression line is straight. However, in this particular sample, the relevant range is small enough to create a relatively high correlation between X and Y.

B: Recompute A, but use polynomial regression to fit a parabola to the data. Compare the results to those of A. Along with the slope and intercept, compute the standard error of the estimate and the correlation coefficient. Plot the data and the straight line. Assess the fit.

A parabola will be fit by polynomial equation order $y = a_2x^2 + a_1x + a_0$

Using a brief python script to obtain the values used renders the coefficients and intercept such that:

$$y = 0.19101732\,x^2 + -0.45183983\,x + 1.48809524$$

Computing the standard error of this equation with

$$SE = \frac{\sigma}{\sqrt{n}} = \sqrt{\frac{S}{n}} \text{ where } S = \sum_{i=1}^{n}(y_i - a_0 - a_1x_i - a_2x_i^2)^2$$

Again, using n-1 to account for working with a sample of unknown proportion, yields

$$SE = \sqrt{\frac{0.71320346}{8}} = 0.29858070$$

This is relatively small compared to the least square linear regression standard error of 0.95622229. The improvement in the model is evident in a chart of the predicted dependent

variable values as well, as shown in figure 11. The polynomial regression, shown in grey, nearly covers entirely the plot of the observed data, as a contrast to the orange plot of the straight line linear regression.

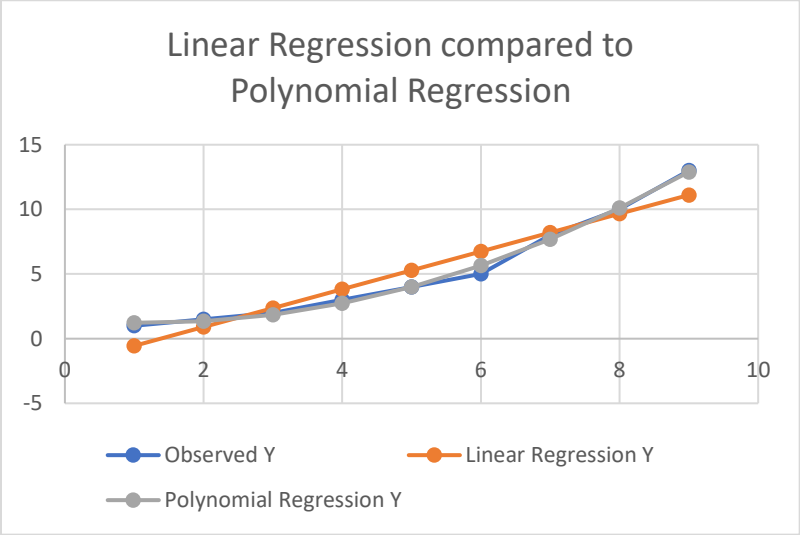

Figure 11

Bibliography

Higham, N. J. (2009). *Cholesky Factorization.* Retrieved from Manchester School of Mathematics: http://www.maths.manchester.ac.uk/~higham/papers/high09c.pdf

Kent State University. (2018, 9 26). *Pearson Correlation*. Retrieved from Kent State University Libraries: https://libguides.library.kent.edu/SPSS/PearsonCorr

PYTHON APPENDIX:

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

t_array = np.array([0.0,1.0,2.0,3.0,4.0,5.0])
y_array = np.array([1.0,2.7,5.8,6.6,7.5,9.9])

def fit_plot(i):
    z = np.polyfit(t_array,y_array,i)
    print("polynomial order",i)
    print("expression coefficients:",z)
    p = np.poly1d(z)

    xp = np.linspace(0,10,100)
    plt.plot(t_array, y_array,'.',xp,p(xp),'-',label=i)
    plt.ylim(0,12)
    plt.xlim(0,6)
#    plt.legend()
    plt.show

#for i in range(0,6):
#    fit_plot(i)

fit_plot(5)
#%%

a = np.array([[1,0,0,0],[0,1,0,0],[0,0,1,0],[0,0,0,1],[1,-1,0,0],\
           [1,0,-1,0],[1,0,0,-1],[0,1,-1,0],[0,1,0,-1],[0,0,1,-1]])
b = np.array([2.95,1.74,-1.45,1.32,1.23,4.45,1.61,3.21,0.45,-2.75])

x_lstsq = np.linalg.lstsq(a,b,rcond=None)[0]
print(x_lstsq)

#%%
a = np.array([[1,10],[1,15],[1,20]])
b = np.array([[11.6],[11.85],[12.25]])
x = np.linalg.lstsq(a,b, rcond=None)
print(x)

#%%

a = np.array([[1,0],[1,1],[1,3]])
aT = a.transpose()
b = np.array([[1],[2],[3]])
ata = np.dot(aT,a)
atb = np.dot(aT,b)
L = np.linalg.cholesky(ata)
```

```
strings = ("A","A Transpose","A Transpose * A","A Transpose *
B","Cholesky Lower")
variables = (a,aT,ata,atb,L)
for i in range(0,len(strings)):
    print(strings[i])
    print(variables[i])
    print()

#%%

# Polynomial Regression
# Finding summations of x^n and yx^n:

x = [1,2,3,4,5,6,7,8,9]
y = [1, 1.5, 2, 3, 4, 5, 8, 10, 13]

x_power = {'sumx0':0,'sumx':0,'sumx2':0,'sumx3':0,'sumx4':0}
xy_power = {'sumx0y':0,'sumx1y':0,'sumx2y':0}

def sumpowerx(sum_,n):
    for i in x:
        sum_+=i**n
#    print(sum_)
    return(sum_)

def sumpowerxy(sum_,n):
    for i in x:
#        print(i)
        sum_ += i**n * y[i-1]
    return(sum_)

for k,v in x_power.items():
    n = list(x_power.keys()).index(k)
    x_power[k] = sumpowerx(v,n)
for k,v in xy_power.items():
    n = list(xy_power.keys()).index(k)
    xy_power[k] = sumpowerxy(v,n)

print(x_power)
print(xy_power)

#%% Solving the linear system:
a = np.matrix([[9,45,285],[45,285,2025],[285,2025,15333]])
b = np.matrix([[47.5],[325],[2438]])
x = np.linalg.solve(a,b)
print(x)
```

ALGORITHM APPENDIX:

Set p_i =1, i=1:n
Do for k = 1:n
    Find s such at $a_{ss} = max_{k<=i<=n} a_{ij}$
    Swap rows and columns of k and s of A
        And swap p_k and p_s
    a_{kk} = Sqrt(a_{kk})
    Do for j = k + 1:n
        $a_{kj}=a_{kj}/a_{kk}$
    End for
    Do for j = k + 1:n
        Do for i = k + 1:j
            $a_{ij} = a_{ij} - a_{ki}a_{kj}$
        End for
    End for
End for
Set P to the matrix whose jth column is the p_ith column of I.
(Higham, 2009)