104283 - Introduction to Numerical Analysis
Spring 2023
Python Assignment 3

Name: Haoyi Yang
Student ID: 999009798

May 24, 2023

Least Squares Approximation

1. Question

Write the following function in Python:
$LS(xs, ys, degree)$
Where xs is a list of nodes and ys is a list of functional values for these nodes. Meaning, $ys[k] = f(xs[k])$, hence the length of these lists must be the same. Parameter degree specifies the degree of the least-squares polynomial to be computed. The function constructs the least squares polynomial and returns the following:

- An n-array with the coefficients of the polynomial (of length degree+1)
- The total error.

The coefficients are computed from a linear system that comes from the so-called normal equations. To solve this linear system, you may use a Python library function or any other method.
Test your function on the sample data set given in the file: *data_points.txt*.
Plot the given data points and the resulting polynomials with degrees 1, 2 and 3 on the same figure. Add a legend and appropriate title and axis labels to the figure.

2. Code, Explanation and Results

From the page 528 in textbook, we can get the normal equations of least squares polynomial and the least squares error, where n represents the degree of the least squares polynomial and m represents the number of data(xs or ys in this question, they are equal):

$$
\begin{aligned}
a_0 \sum_{i=1}^{m} x_i^0 + a_1 \sum_{i=1}^{m} x_i^1 + a_2 \sum_{i=1}^{m} x_i^2 + \cdots + a_n \sum_{i=1}^{m} x_i^n &= \sum_{i=1}^{m} y_i x_i^0, \\
a_0 \sum_{i=1}^{m} x_i^1 + a_1 \sum_{i=1}^{m} x_i^2 + a_2 \sum_{i=1}^{m} x_i^3 + \cdots + a_n \sum_{i=1}^{m} x_i^{n+1} &= \sum_{i=1}^{m} y_i x_i^1, \\
&\vdots \\
a_0 \sum_{i=1}^{m} x_i^n + a_1 \sum_{i=1}^{m} x_i^{n+1} + a_2 \sum_{i=1}^{m} x_i^{n+2} + \cdots + a_n \sum_{i=1}^{m} x_i^{2n} &= \sum_{i=1}^{m} y_i x_i^n.
\end{aligned}
$$

Figure 1: The normal equations of least squares polynomial

$$
E = \sum_{i=1}^{m} (y_i - P_n(x_i))^2
$$

Figure 2: The least squares error

Then using these equations, we can implement the Python function code:

```python
def LS(xs,ys,degree):
    A = []
    B = []
    tmp_list = []
    tmp_num1 = tmp_num2 = P = E = 0
    for i in range(degree+1):
        for j in range(degree+1):
            for k in range(0,len(xs)):
                tmp_num1 += xs[k] ** (j+i)
                tmp_num2 += ys[k] * (xs[k] ** (j + i))
            tmp_list.append(tmp_num1)
            tmp_num1 = 0
            if len(B) == degree+1:
                continue
            else:
                B.append(tmp_num2)
                tmp_num2 = 0
        A.append(tmp_list)
```

```
        tmp_list = []
    result = np.linalg.inv(np.array(A)).dot(np.array(B))
    # The above code finds the n-array with the coefficients of the polynomial.

    for m in range(len(result)):
        P += result[m] * (x**m) # This is the least square polynomial P(x).
    for n in range(len(xs)):
        E += (ys[n] - P.subs(x,xs[n]))**2 # This is the error.

    return(f"When degree = {degree}, the n-array with"
    f"\the coefficients of the polynomial is: {result}, and the error is {E}")
    # It is too long so I add "\" move it to a new line
    # You can see the original version in the code I attached.
```

Then we add the following code to get the data in the file: *data_point.txt*, and print the result for degrees 1, 2 and 3:

```
data = np.loadtxt('data_points.txt')
xs = data[:,0] # entire first column
ys = data[:,1] # entire second column

print(LS(xs,ys,1))
print(LS(xs,ys,2))
print(LS(xs,ys,3))
```

And we get the output:
When degree = 1, the n-array with the coefficients of the polynomial is: [0.92951404, 0.52810205], and the error is 0.0245660611491392
When degree = 2, the n-array with the coefficients of the polynomial is: [1.01134099, -0.32569875, 1.14733031], and the error is 0.000945246293489820
When degree = 3, the n-array with the coefficients of the polynomial is: [1.00043981, -0.00154099, -0.01150567, 1.02102256], and the error is 0.000111237678230218
That is, the polynomials for different degrees are:
For degree = 1,
$P_1(x) = 0.92951404 + 0.52810205x$
For degree = 2,
$P_2(x) = 1.01134099 - 0.32569875x + 1.14733031x^2$
For degree = 3,
$P_3(x) = 1.00043981 - 0.00154099x - 0.01150567x^2 + 1.02102256x^3$

Then I plot the given data points and the resulting polynomials on the same figure. Here's the figure:

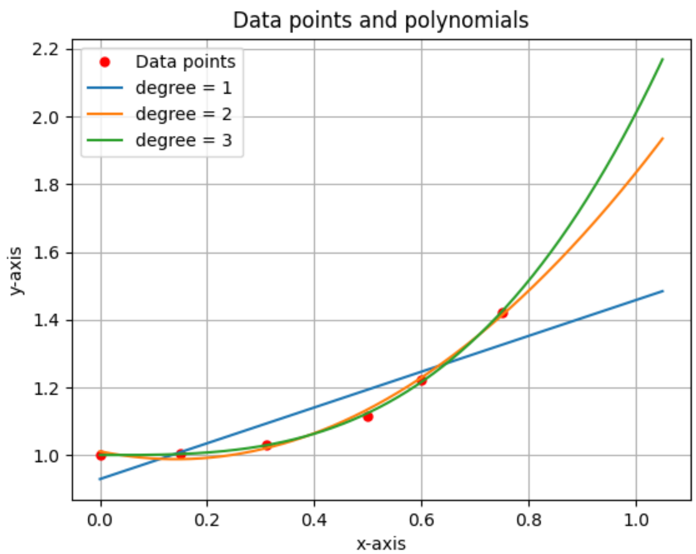

Figure 3: Data points and the resulting polynomials

And here's the Python code to implement the figure:

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

data = np.loadtxt('data_points.txt')
x_data = data[:,0] # entire first column
y_data = data[:,1] # entire second column

def p1(x):
    return 0.528102053515867 * x + 0.929514042729724
def p2(x):
    return 1.14733030545859 * x**2 - 0.325698750708621 * x + 1.01134099279321
def p3(x):
    return 1.02102256421131 * x**3 - 0.0115056745189577 * x**2\
           - 0.00154098604718911 * x + 1.00043980518403

x_values = np.linspace(0, 1.05, 500)
plt.plot(x_data, y_data,'r.', label="Data points",markersize=10)
plt.plot(x_values, p1(x_values), label="degree = 1")
plt.plot(x_values, p2(x_values), label="degree = 2")
plt.plot(x_values, p3(x_values), label="degree = 3")
plt.title("Data points and polynomials")
```

```python
plt.xlabel("x-axis")
plt.ylabel("y-axis")
plt.legend()
plt.grid()
plt.show()
```

Moore's Law Prediction

1. Question

Moore's law was formulated by computer scientist and Intel co-founder Gordon Moore (1929- 2023) in 1965. The law states that the number of transistors on microchips will roughly double every one and a half to two years. The figure below shows the number of transistors N in 13 microprocessors, and the year of their introduction. Notice the plot gives the number of transistors on a logarithmic scale.
The attached file moore.txt includes this raw data. Find the least squares linear model for the data. Use your model to predict the number of transistors in a microprocessor introduced in the year 2023. Plot the results (raw data and model) and compare your calculated model to Moore's law.

2. Code, Explanation and Results

We know that the least squares linear model is actually the situation when we set degree 1 in the first question's code we wrote. However, the output seems not what we want so I modify all the code according to the equation in the textbook, so that it can be used for least squares linear model and output the least square line:

The solution to this system of equations is

$$a_0 = \frac{\sum_{i=1}^{m} x_i^2 \sum_{i=1}^{m} y_i - \sum_{i=1}^{m} x_i y_i \sum_{i=1}^{m} x_i}{m\left(\sum_{i=1}^{m} x_i^2\right) - \left(\sum_{i=1}^{m} x_i\right)^2} \tag{8.1}$$

and

$$a_1 = \frac{m\sum_{i=1}^{m} x_i y_i - \sum_{i=1}^{m} x_i \sum_{i=1}^{m} y_i}{m\left(\sum_{i=1}^{m} x_i^2\right) - \left(\sum_{i=1}^{m} x_i\right)^2}. \tag{8.2}$$

Figure 4: The linear least squares equations

And here's the function code only for linear least square method:

```
import numpy as np
from sympy.abc import x
def Linear_LS(xs,ys):
    a_0 = a_1 = x_i = y_i = x_i2 = x_iy_i = 0
    for i in range(len(xs)):
        x_i2 += xs[i]**2
        y_i += ys[i]
        x_iy_i += xs[i] * ys[i]
        x_i += xs[i]
    a_0 = (x_i2 * y_i - x_iy_i * x_i) / (len(xs) * x_i2 - (x_i**2))
    a_1 = (len(xs) * x_iy_i - x_i * y_i) / (len(xs) * x_i2 - (x_i**2))
    P = a_0 + a_1 * x
    return(P)
```

Notice that we need to use logarithmic scale number of transistors when we import the data from *moore.txt*, otherwise the data won't fit a line. Then we add following codes to print result:

```
data = np.loadtxt('moore.txt')
xs = data[:,0]
ys = np.log10(data[:,1]) # In a logarithmic scale
print(Linear_LS(xs,ys))
print(10 ** Linear_LS(xs,ys).subs(x,2023)) # The real data is 10^P(2023)
```

And we get the output:
0.154018179843825*x - 300.290221658514
194337300412.816
That is, the least squares linear model for the data is:
$P(x) = 0.154018179843825x - 300.290221658514$
And the prediction of transistors by using this model is:
194337300412.816

Then we plot the raw data and the model. Here's the figure:

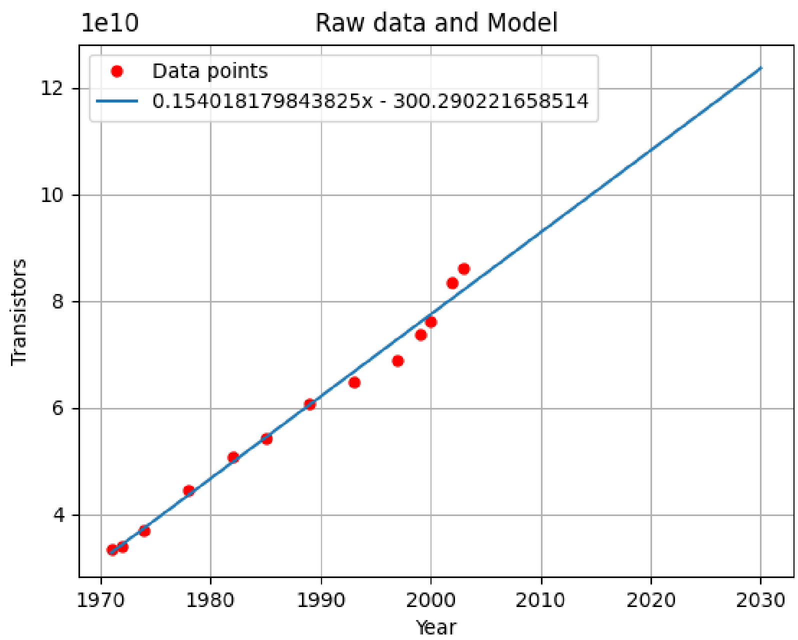

Figure 5: Raw data and Model

And here's the Python code to implement the figure:

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

data = np.loadtxt('moore.txt')
x_data = data[:,0]
y_data = np.log10(data[:,1])

def p(x):
    return 0.154018179843825*x - 300.290221658514

x_values = np.linspace(1971, 2030, 20000)

plt.title("1e10", loc="left")
plt.plot(x_data, y_data,'r.', label="Data points",markersize=10)
plt.plot(x_values, p(x_values), label="0.154018179843825x - 300.290221658514")
plt.xlabel("Year")
plt.ylabel("Transistors")
plt.title("Raw data and Model")
plt.legend()
```

```
plt.grid()
plt.show()
```

Now we need to compare my calculated model to Moore's law. Since the law states that the number of transistors on microchips will roughly double every one and a half to two years. So we can assume from 2003 - 2023, the number of transistors double about 10 times by Moore's law. That is,

$$\text{The number of transistors} \approx 410000000 * 2^{10} = 419840000000$$

As a result, we can see that the 2023 data predicted by our model differs from Moore's Law by more than two times ($4.9184 * 10^{11}$ and $1.9434 * 10^{11}$).