Homework_4

February 18, 2019

1 Data sets for Homework 4

There are three data sets in this notebook. Each data set appears in two different ways: as a list of data points and with the independent and dependent variables given separately in two lists.

Note: You may add cells using the menu `Insert -> Insert Cell Above/Below`. You are welcome to copy commands that we have used in other notebooks into this one.

1.0.1 Problem 1

The following data set gives the number of people in Bangladesh living without electricity by year: `elec` gives a list of data points, `elecx` gives a list of years, and `elecy` gives a list of the number of people without electricity.

```
In [1]: elec = [[1990,97115487],
                [1991,93190282],
                [1992,96268982],
                [1993,95627988],
                [1994,95501823],
                [1995,94060109],
                [1996,93157160],
                [1997,96111201],
                [1998,91079158],
                [1999,89888449],
                [2000,89475245],
                [2001,87171255],
                [2002,85636155],
                [2003,83949183],
                [2004,83936648],
                [2005,79991525],
                [2006,71920671],
                [2007,78719467],
                [2008,72543169],
                [2009,69761553],
                [2010,68071508],
                [2011,62180414],
                [2012,60923202],
                [2013,60664947],
                [2014,59936385],
```

```
        [2015,51254335],
        [2016,39238736]]
elecx, elecy = map(vector, zip(*elec))

def ones(m):
    return vector([1] * m)

def projection(b, basis):
    return sum([b.dot_product(v)/v.dot_product(v)*v for v in basis])

def unit(v):
    return v/v.norm()

def vectors2matrix(vectors):
    return matrix(vectors).transpose()

def gs(basis):
    onbasis = []
    for b in basis:
        if len(onbasis) == 0: onbasis.append(b)
        else: onbasis.append(b-projection(b, onbasis))
    return map(unit, onbasis)

def QR(A):
    Q = vectors2matrix(gs(A.columns()))
    return Q, Q.T*A

def poly_regression(data, k):
    ind, dep = zip(*data)
    A = np.array([ [v**j for j in range(k+1)] for v in ind])
    A = matrix(A)
    B = A.T*A
    b = A.T*vector(dep)
    coefficients = B \ b
    return coefficients * vector([x^i for i in range(k+1)])

def plot_regression(data, k, color='blue'):
    x, y = zip(*data)
    f = poly_regression(data, k)
    return list_plot(data, color=color, size=20) + plot(f, min(x), max(x), color=color
```

In [2]:
```
A = vectors2matrix([ones(len(elecx)), elecx])
b = elecy
```

In [3]:
```
Q, R = QR(A)
```

In [4]:
```
y_int, b = n(R.inverse() * Q.T * b)
y_int, b
```

Out[4]: (3.88369074321062e9, -1.89923032173382e6)

```
In [5]: y_int/-(b)
```

Out[5]: 2044.87612627476

By the end of the year 2044 there will be 0 people in bangladesh without electricity

1.0.2 Problem 2

The following data set shows the relative risk of having an accident at a particular level of blood alcohol content. The list `risk` gives the data points, while `riskx` gives a list of blood alcohol levels and `risky` gives a list of relative risks.

Note: If you have a list `my_list`, you can create a new list whose entries are the natural log of the entries in `my_list` by saying `vector(np.log(my_list))`.

```
In [6]: import numpy as np
        risk = [[0, 1],
                [.01, 1.03],
                [.03, 1.06],
                [.05, 1.38],
                [.07, 2.09],
                [.09, 3.54],
                [.11, 6.41],
                [.13, 12.6],
                [.15, 22.1],
                [.17, 39.5],
                [.19, 65.32],
                [.21, 99.78]]
        riskx, risky = map(vector, zip(*risk))
```

```
In [7]: A = vectors2matrix([ones(len(riskx)), vector(riskx)])
        b = vector(np.log(risky))
```

```
In [8]: Q, R = QR(A)
        y_int, b = n(R.inverse() * Q.T * b)
        y_int, b
```

Out[8]: (-0.539980939336522, 23.8319617111994)

Part 2a. Our equation is now y = 23.83x - 0.54 where β_0 = -0.54 and β_1 = 23.83

```
In [9]: C = e ** y_int
        a = b
        C, a
```

Out[9]: (0.582759360048180, 23.8319617111994)

Part 2b. These are our logarithmic parameters where C = e^b = 0.5828 and a = m (in $y = mx + b$ form).

```
In [10]: np.log(1.50/C)/a
```

Out[10]: 0.039671348036800155

Part 2c. Our BAC for a risk of 1.50 would be 0.0396

1.0.3 Problem 3

The following data set gives the temperature in the earth's atmosphere at various altitudes. The list `temp` gives a list of data points while `tempx` gives a list of altitudes and `tempy` gives a list of temperatures.

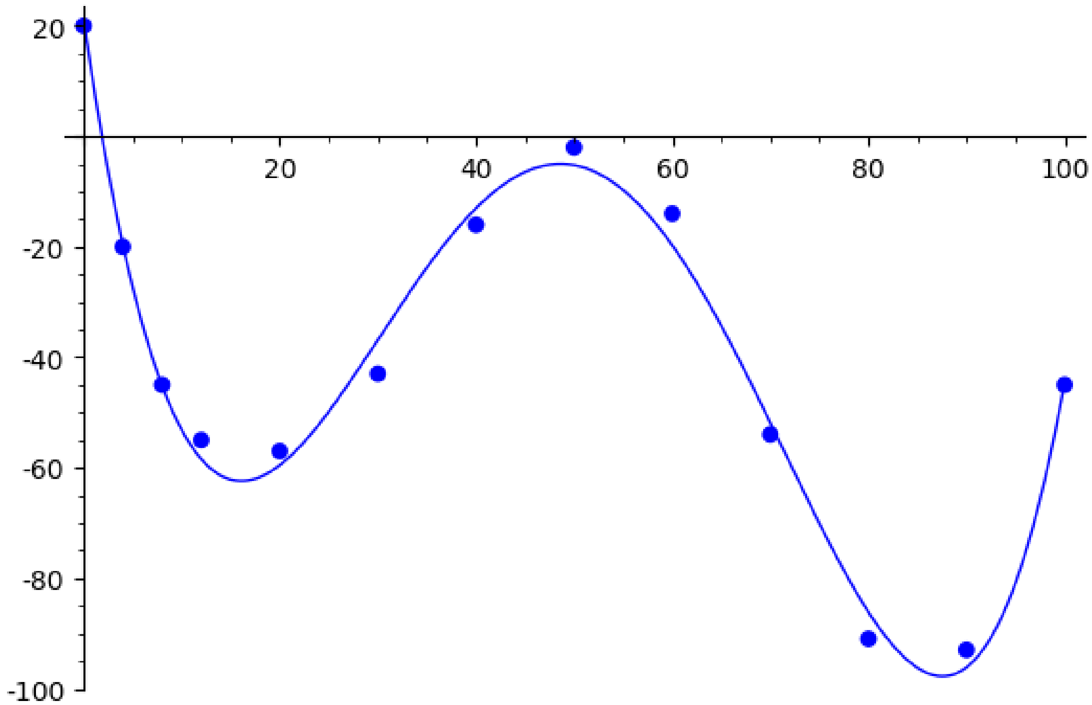
```
In [11]: def power_of(vector, power):
             new_vec = []
             for val in vector:
                 new_vec.append(val ** power)
             return new_vec
```

```
In [12]: temp = [[0, 20],
                 [4, -20],
                 [8, -45],
                 [12, -55],
                 [20, -57],
                 [30, -43],
                 [40, -16],
                 [50, -2],
                 [60, -14],
                 [70, -54],
                 [80, -91],
                 [90, -93],
                 [100, -45]]
         tempx, tempy = map(vector, zip(*temp))
```

```
In [13]: tempx_2 = power_of(tempx, 2)
         tempx_3 = power_of(tempx, 3)
         tempx_4 = power_of(tempx, 4)
         A = vectors2matrix([ones(len(tempx)), tempx, vector(tempx_2), vector(tempx_3), vector
         b = tempy
         Q, R = QR(A)
         x_hat = R.inverse() * Q.T * b
         v0, v1, v2, v3, v4 = n(x_hat)
```

```
In [14]: var('t')
         list_plot(temp, color='blue', size=40) + plot((21.310 - 12.41*t + 0.585*t^2 - 0.0092*
         list_plot(temp, color='blue', size=40) + plot((v0 + v1*t + v2*t**2 + v3*t**3 + v4*t**4
```

Out[14]:



Part 3a. because of the almose sine-wave look of this data, we opted to a quartic polynomial to allow for enough peaks and valleys to adaquately fit all of the data

Part 3b. We can see here that the predicted temperaure at an altitude of $55km$ is -9.73

In [0]: