Mathematical Foundations for Computer Vision and Machine Learning - Polynomial Approximation

ID : 20131790 Name : So-Hyun Kwon

October 12, 2017

1 Problem Definition

- Demonstrate that you understand the model fitting algorithm based on the polynomial with degree $n-1$
- Implement Polynomial Approximation Code by using Python3
- Report using LaTeX

2 Algorithm Description

2.1 What is the goal of algorithm?

- Given a set of data pairs $(x_1, y_1), (x_2, y_2), ..., (x_m, y_m)$.
- Aims : Find a model $(w_0, w_1, ..., w_{p-1})$ that yields $y_i \approx f(x_i)$.
- $f(x_i) = w_0 + w_1 x_i^1 + w_2 x_i^2 + ... + w_{p-1} x_i^{p-1}$.
- Demonstration
  Regularization : model with varing degree of the polynomial

2.2 Concept of QR factorization method

There are many algorithms that can calculate solution of polynomial approximation. But I used QR factorization method.

$$\begin{aligned}\hat{x} = (A^TA)^{-1}A^Tb &= ((QR)^T(QR))^{-1}(QR)^Tb \\ &= (R^TQ^TQR)^{-1}R^TQ^Tb \\ &= (R^TR)^{-1}R^TQ^Tb \\ &= (R^{-1}R^{-T}R^TQ^Tb \\ &= R^{-1}Q^Tb\end{aligned}$$

Algorithm

1. $Compute\ QR\ factorization\ A = QR\ (2mn^2\ flops\ if\ A\ is\ m \times n)$
2. $Matrix - vector\ product\ d = Q^Tb\ (2mn\ flops)$
3. $Solve\ Rx = d\ by\ back\ substitution\ (n^2\ flops)$

2.3 Process of algorithm

Let approximated $n-1$ polynomial equation is

$$y(x) = c_1 + c_2x + c_3x^2 + \cdots + c_nx^n - 1$$

number of m data points

$$(a_1, b_1), ..., (a_m, b_m)$$

Then, the solution of polynomial approximation

$$\hat{x} = \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_n \end{bmatrix}$$

(i) Set matrices

$$A = \begin{bmatrix} 1 & a_1 & a_1^2 & \dots & a_1^{n-1} \\ 1 & a_2 & a_2^2 & \dots & a_2^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & a_m & a_m^2 & \dots & a_m^{n-1} \end{bmatrix},\ b = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix}$$

(ii) The solution $\hat{x}$ is which minimizes $||A\hat{x} - b||^2$

(iii) Calculate QR Factorization by using modified $Gram - Schmidt$ algorithm

- QR factorization's goal is to make matrices Q, R, which satisfy $A = QR$

- $Q = \begin{bmatrix} q_1 & q_2 & \dots & q_n \end{bmatrix}$ which vectors $q_1, \dots, q_n$ are orthonormal

- $R = \begin{bmatrix} R_{11} & R_{12} & \dots & R_{1n} \\ 0 & R_{22} & \dots & R_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & R_{nn} \end{bmatrix}$

(iv) Method of modified $Gram - Schmidts$ algorithm

This is the cycle of calculation

$$\text{Let vector } v = a_n - q_1 R_{1n} - \dots - q_{n-1} R_{n-1n}$$

1. Calculate R_{nn}

$$\text{if } n = 1,\ R_{nn} = ||a_1||, \text{ else, } R_{nn} = ||v||$$

2. Calculate q_n

$$q_n = v / R_{nn}$$

3. Calculate R_{nj} which $(n < j)$

$$R_{nj} = q_n^T a_j$$

(v) Calculate $\hat{x}$

$$\hat{x} = R^{-1} Q^T b$$

3 Technical details about the code

3.1 Get Data

I get data points by making directly and using website: *https://www.kaggle.com/* I used these data

Swedish Crime Rates
Reported crimes in Sweden from 1950 to 2015
MGN · updated 8 months ago · crime

Swedish central bank interest rate and inflation
Historic Swedish interest rate 1908-2001 and Swedish inflation consumer price
Christian Nygaard · updated a year ago · finance

3.2 Read Data

I read coordinate points by using "" or "*tab*" as token.

```python
def calculateQiRin(Qi, Rin):
    result = [[]for _ in range (len(Qi))]
    for rowindex in range(0,len(Qi)):
        result[rowindex].append(Qi[rowindex][0]*Rin)
    return result

def subColumns(column1, column2):
    result = [[] for _ in range(len(column1))]
    if(len(column1)==len(column2)):
        for index in range(0,len(column1)):
            result[index].append(column1[index][0]-column2[index][0])
    return result

def getColumn(matrix, index):
    column = [[]for _ in range(len(matrix))]
    for rowindex in range(0,len(matrix)):
        column[rowindex].append(matrix[rowindex][index-1])
    return column

def getRnn(RnnParameter):
    result = 0
    for rowindex in range(0,len(RnnParameter)):
        result += pow(RnnParameter[rowindex][0],2)
    result = math.sqrt(result)
    return result

def getGetRnnParameter(matrixA, matrixQ, matrixR, n):
    result = getColumn(matrixA, n)
    if(n == 1):
        return result
    else:
        for index in range(1,n):
            Qi = getColumn(matrixQ, index)
            Rin = matrixR[index-1][n-1]
            result = subColumns(result, calculateQiRin(Qi, Rin))
    return result

def getQn(parameter, Rnn):
    qn = [[]for _ in range(len(parameter))]
    for rowindex in range(0,len(parameter)):
        qn[rowindex].append(parameter[rowindex][0]/Rnn)
    return qn

def getRij(qi, aj):
    result = 0
    if(len(qi)==len(aj)):
        for index in range(0,len(qi)):
            result += qi[index][0]*aj[index][0]
    else:
        return -1
    return result

def addColumnInMatrix(matrix,column):
    for rowindex in range(0,len(matrix)):
        matrix[rowindex].append(column[rowindex][0])
```

3.3 Making Matrices

I made Matrices A and b

```
def makeMatrixA(dataCoordinates, degreeNum):
    matrix = [[]for _ in range(len(dataCoordinates))]
    for coordinate in dataCoordinates:
        for squared in range(0,degreeNum):
            matrix[dataCoordinates.index(coordinate)].append(pow(coordinate[0],squared))
    return matrix

def makeMatrixb(dataCoordinates, degreeNum):
    matrix = [[]for _ in range(len(dataCoordinates))]
    for coordinate in dataCoordinates:
        matrix[dataCoordinates.index(coordinate)].append(coordinate[1])
    return matrix
```

3.4 Calculation

I made each calculation functions which means Method of modified $Gram-Schmidts$ algorithm. (2.3)

```
def calculateQiRin(Qi, Rin):
    result = [[]for _ in range (len(Qi))]
    for rowindex in range(0,len(Qi)):
        result[rowindex].append(Qi[rowindex][0]*Rin)
    return result

def subColumns(column1, column2):
    result = [[] for _ in range(len(column1))]
    if(len(column1)==len(column2)):
        for index in range(0,len(column1)):
            result[index].append(column1[index][0]-column2[index][0])
    return result

def getColumn(matrix, index):
    column = [[]for _ in range(len(matrix))]
    for rowindex in range(0,len(matrix)):
        column[rowindex].append(matrix[rowindex][index-1])
    return column

def getRnn(RnnParameter):
    result = 0
    for rowindex in range(0,len(RnnParameter)):
        result += pow(RnnParameter[rowindex][0],2)
    result = math.sqrt(result)
    return result

def getGetRnnParameter(matrixA, matrixQ, matrixR, n):
    result = getColumn(matrixA, n)
    if(n == 1):
        return result
    else:
        for index in range(1,n):
            Qi = getColumn(matrixQ, index)
            Rin = matrixR[index-1][n-1]
            result = subColumns(result, calculateQiRin(Qi, Rin))
    return result

def getQn(parameter, Rnn):
    qn = [[]for _ in range(len(parameter))]
    for rowindex in range(0,len(parameter)):
        qn[rowindex].append(parameter[rowindex][0]/Rnn)
    return qn

def getRij(qi, aj):
    result = 0
    if(len(qi)==len(aj)):
        for index in range(0,len(qi)):
            result += qi[index][0]*aj[index][0]
    else:
        return -1
    return result

def addColumnInMatrix(matrix,column):
    for rowindex in range(0,len(matrix)):
        matrix[rowindex].append(column[rowindex][0])
```

3.5 cycle

I made cycle of calculation which calculate Matrix Q and R in QR factorization.(2.3)

```
def RniQnCycle(matrixA, matrixQ, matrixR, n):
    columnNum = len(matrixA[0])
    parameter = getGetRnnParameter(matrixA, matrixQ, matrixR, n+1)
    # add Rnn
    Rnn = getRnn(parameter)
    matrixR[n].append(Rnn)
    # add qn
    qn = getQn(parameter, Rnn)
    addColumnInMatrix(matrixQ, qn)
    # add Rni
    if(n==columnNum-1):
        return
    else:
        for i in range(n+2,columnNum+1):
            ai = getColumn(matrixA, i)
            Rni = getRij(qn,ai)
            matrixR[n].append(Rni)
    return
```

4 Algorithms of result

I tested algorithms with $data1.txt$, degree 3

Make matrix A and matrix b

```
MatrixA
[1.0, -5.0, 25.0]
[1.0, -4.0, 16.0]
[1.0, -2.0, 4.0]
[1.0, 0.0, 0.0]
[1.0, 1.0, 1.0]
[1.0, 3.0, 9.0]
[1.0, 4.0, 16.0]
[1.0, 6.0, 36.0]
[1.0, 8.0, 64.0]
[1.0, 10.0, 100.0]
```

```
Matrixb
[10.0]
[8.0]
[3.0]
[-1.0]
[3.0]
[4.0]
[5.0]
[8.0]
[5.0]
[3.0]
```

Result of matrix Q and matrix R

```
MatrixQ
[0.31622776601683794, -0.4713473836821528, 0.46615049951007415]
[0.31622776601683794, -0.4049604282339623, 0.26016124049902833]
[0.31622776601683794, -0.2721865173375812, -0.06123789211684974]
[0.31622776601683794, -0.13941260644120013, -0.261864510857776]
[0.31622776601683794, -0.07302565099300960, -0.31688812752513223]
[0.31622776601683794, 0.05974825990337148, -0.336355975453631]
[0.31622776601683794, 0.12613521535156202, -0.3008002067147735]
[0.31622776601683794, 0.2589091262479431, -0.13910928383084456]
[0.31622776601683794, 0.3916830371443242, 0.143354152928036]
[0.31622776601683794, 0.5244569480407053, 0.5465901035618684]
```

```
MatrixR
[3.1622776601683795, 6.640783086353597, 85.69772459056308]
[0, 15.063200191194433, 69.965212346848]
[0, 0, 66.24023747889547]
```

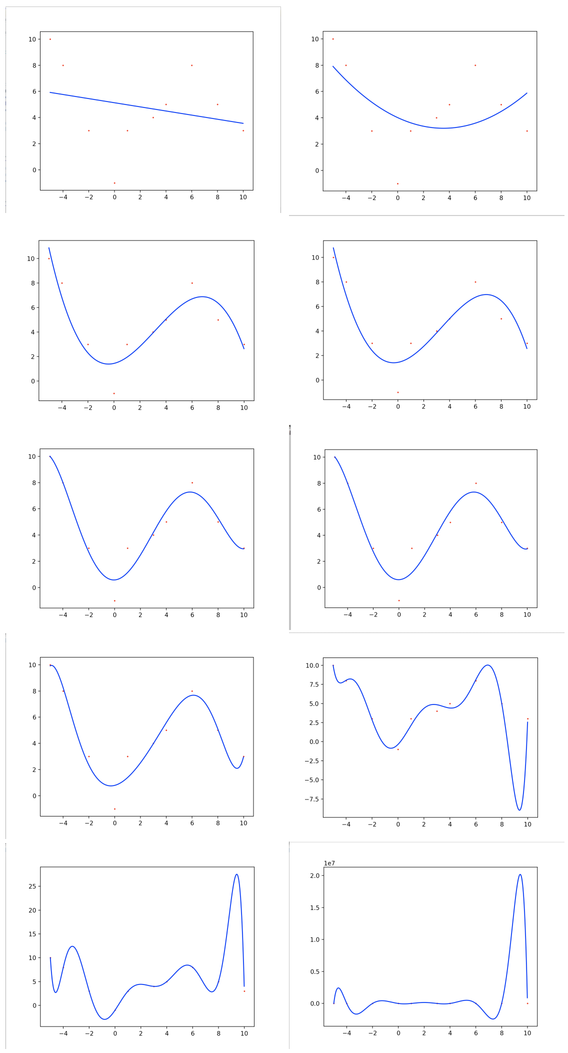
Result of $\hat{x}$

```
Result
[ 4.0146081]
[-0.4568079]
[ 0.06437965]
```

5 ScreenShots of result

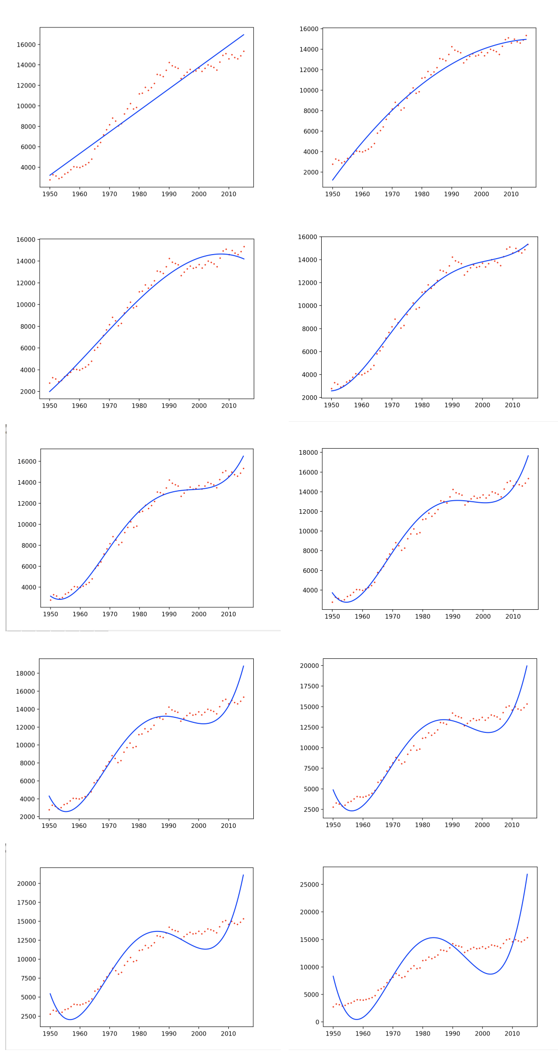
When I run this code with $data1.txt$ with variety of degrees, the result looks like below.

Degree : $2, 3, 4, 5, 6, 7, 8, 9, 10, 11$



When I run this code with *data2.txt* with variety of degrees, the result looks like below.

$$\text{Degree}: 2, 3, 4, 5, 6, 7, 8, 9, 10, 15$$



When I run this code with *data*3.*txt* with variety of degrees, the result looks like below.

Degree : $2, 4, 5, 7, 10, 12, 15$