# -\*- coding: utf-8 -\*-

"""

Spyder Editor

This is a temporary script file.

"""

import numpy as np

import matplotlib.pyplot as plt

class Polynomial:

def \_\_init\_\_(self, coefficients):

'''

This class implements a polynomial function.

It takes input the coefficients of a n-order polynomial as a list

and given m number of inputs X, it returns the corresponding Y.

usage:

p = Polynomial([a\_0, a\_1, ..., a\_n])

X = np.linspace(-3, 3, 50, endpoint=True)

F = p(X)

plt.plot(X,F)

'''

self.coefficients = coefficients

def \_\_repr\_\_(self):

'''

method to return the canonical string representation

of a polynomial.

'''

return "Polynomial" + str(self.coefficients)

def \_\_call\_\_(self, x):

'''

This method calculates the ordinate (Y-axis) given the

abscissa (X-axis) of a particular point on the polynomial.

input: x can be a number or a numpy array

'''

# Initialize return value

res = 0

# Calculate the y\_i given x\_i

# remember: y\_i = f(x\_i)

# y\_i = a\_0 + a\_1 \* x\_i ^ 1 + a\_2 \* x\_i ^ 2 + ... + a\_n \* x\_i ^ n

#

# use a for loop to calculate res

# your loop might look like this

for index, coeff in enumerate(self.coefficients):

res += coeff \* x\*\* index

return res

def degree(self):

return len(self.coefficients)

def \_\_str\_\_(self):

res = ""

res += str(self.coefficients[0])

for i in range(1, len(self.coefficients)):

coeff = self.coefficients[i]

if coeff < 0:

res += " - " + str(-coeff) + "x^" + str(i)

else:

res += " + " + str(coeff) + "x^" + str(i)

return res

def get\_coefficients(x, y):

'''

Given a set of Xs and Ys, this function tries to form a

Vandermonde matrix. It then uses np.linalg.pinv to find the

inverse of the matrix to calculate the coefficients of the

n-order polynomial to fit the given data points.

usage:

x = np.array([0,1,2])

y = np.array([0,1,4])

coefficients = get\_coefficients(x, y) # array([-0., 0., 1.])

# y = f(x) = 0 + 0 + x^2

'''

# Get length

length = len(x)

# Create (len X len) matrix

coeff = np.zeros((length, length))

# Populate the matrix

# a\_0 + a\_1 \* x\_0 + a\_2 \* x\_0^2 + .. a\_n \* x\_0^n = y\_0

# a\_0 + a\_1 \* x\_1 + a\_2 \* x\_1^2 + .. a\_n \* x\_1^n = y\_1

# a\_0 + a\_1 \* x\_2 + a\_2 \* x\_2^2 + .. a\_n \* x\_2^n = y\_2

# .. .. ..

# a\_0 + a\_1 \* x\_(n-1) + a\_2 \* x\_(n-1)^2 + .. a\_n \* x\_(n-1)^n = y\_(n-1)

# |1 x\_0 x\_0^2 ... x\_0^n | |a\_0| | y\_0 |

# |1 x\_1 x\_1^2 ... x\_1^n | |a\_1| | y\_1 |

# |1 x\_2 x\_2^2 ... x\_2^n | |a\_2| = | y\_2 |

# | .. .. | |...| | ... |

# |1 x\_(n-1) x\_(n-1\_^2 ... x(n-1)^n | |a\_n| |y\_(n-1)|

for i in range(length):

for j in range(length):

coeff[i][j] = x[i]\*\*j

#print(coeff)

# Invert the matrix

inverse = np.linalg.pinv(coeff)

# calculate the coefficients

coefficients = np.dot(inverse, y)

#print(coefficients)

return coefficients

x = np.array([-3.,-2.,-1.,0.,1.,3])

y = np.array([-80.,-13.,6.,1.,5., 16])

coefficients = get\_coefficients(x, y)

p = Polynomial(list(coefficients))

X = np.linspace(-3, 3, 50, endpoint=True)

F = p(X)

plt.plot(X,F)

plt.plot(x,y, 'ro')