Aarhus university

Computer-Science

Numerical Linear Algebra

Handin 3

Author
Søren M. Damsgaard

Student number
202309814

February 24, 2026

Polynomials for days!

(a) Plotting like a Caveman

(b) System of equations, but make it polynomial

Settign up system of equation for p_1, p_2 and p_3, going through the last three data points:

$$p_1(9) = a_1 + b_1 \cdot 9 + c_1 \cdot 9^2 = 45 \tag{1}$$
$$p_1(11) = a_1 + b_1 \cdot 11 + c_1 \cdot 11^2 = 65 \tag{2}$$
$$p_1(12) = a_1 + b_1 \cdot 12 + c_1 \cdot 12^2 = 70 \tag{3}$$

This give the following augmened matrix

$$\left(\begin{array}{ccc|c} 1 & 9 & 81 & 45 \\ 1 & 11 & 121 & 65 \\ 1 & 12 & 144 & 70 \end{array}\right)$$

Now to reduce it to something useful, by doing to following operations:

$$R_2 \to R_2 - R_1$$
$$R_3 \to R_3 - R_1$$
$$R_2 \to \frac{1}{2}R_2$$
$$R_3 \to R_3 - 3R_2$$

This gives us the following matrix:

$$\left(\begin{array}{ccc|c} 1 & 9 & 81 & 45 \\ 0 & 1 & 20 & 10 \\ 0 & 0 & 3 & -5 \end{array}\right)$$

With this we can easily find a_1, b_1 and c_1 by back substitution, which gives us $a_1 = -210$, $b_1 = 43.3$ and $c_1 = -\frac{5}{3}$.

(c) Vandermonde, Vandermonde, Vandermonde

(d) Too many polynumials, not enough time

Setting up the system of equations for a new set of p_1 and p_2:

$$p_1(2) = a_1 + b_1 \cdot 2 + c_1 \cdot 2^2 + d_1 \cdot 2^3 = 25 \tag{4}$$
$$p_1(6) = a_1 + b_1 \cdot 6 + c_1 \cdot 6^2 + d_1 \cdot 6^3 = 35 \tag{5}$$
$$p_1(9) = a_1 + b_1 \cdot 9 + c_1 \cdot 9^2 + d_1 \cdot 9^3 = 45 \tag{6}$$

$$p_2(9) = a_2 + b_2 \cdot 9 + c_2 \cdot 9^2 + d_2 \cdot 9^3 = 45 \tag{7}$$
$$p_2(11) = a_2 + b_2 \cdot 11 + c_2 \cdot 11^2 + d_2 \cdot 11^3 = 65 \tag{8}$$
$$p_2(12) = a_2 + b_2 \cdot 12 + c_2 \cdot 12^2 + d_2 \cdot 12^3 = 70 \tag{9}$$

To find where $p_1'(9) = p_2'(9)$, find the derivates then set them equal to each other and then group them on the left side.

$$p_1'(9) = b_1 + 2c_1 \cdot 9 + 3d_1 \cdot 9^2$$
$$p_2'(9) = b_2 + 2c_2 \cdot 9 + 3d_2 \cdot 9^2$$
$$b_1 + 2c_1 \cdot 9 + 3d_1 \cdot 9^2 = b_2 + 2c_2 \cdot 9 + 3d_2 \cdot 9^2$$
$$b_1 + 2c_1 \cdot 9 + 3d_1 \cdot 9^2 - b_2 - 2c_2 \cdot 9 - 3d_2 \cdot 9^2 = 0$$

Then the system can be erected for $[a_1, b_1, c_1, d_1, a_2, b_2, c_2, d_2]$

$$\left(\begin{array}{cccccccc|c} 1 & 2 & 4 & 8 & 0 & 0 & 0 & 0 & 25 \\ 1 & 6 & 36 & 216 & 0 & 0 & 0 & 0 & 35 \\ 1 & 9 & 81 & 729 & 0 & 0 & 0 & 0 & 45 \\ 0 & 0 & 0 & 0 & 1 & 9 & 81 & 729 & 45 \\ 0 & 0 & 0 & 0 & 1 & 11 & 121 & 1331 & 65 \\ 0 & 0 & 0 & 0 & 1 & 12 & 144 & 1728 & 70 \\ 0 & 1 & 18 & 243 & 0 & -1 & -18 & -243 & 0 \end{array}\right)$$

(e) Sherlock Holmes and the One Solution

The above systems is consistent, since no equation contradicts another. Altough it is not unique since there are 7 equations with 8 unknowns, which means the system is underdetermined and has infinitely many solutions.

(f) Math Tariffs

The assignment recommends imposing the condition $p_1'(0.0) = 0$, which will be used.
This conditions impies the slope at $t = 0$ is 0, which means the drone starts with no heat increase, which is a resonable assumption for a drone that hasn't started flying yet.
If we look at the derivative of p_1 and set $x = 0$ we get:

$$p_1'(0) = b_1 + 2c_1 \cdot 0 + 3d_1 \cdot 0^2 = b_1 \tag{10}$$

This means that b_1 must be 0, which reduces the number of unknowns from 8 to 7. This means that the system of equations now have and equal number of equations and uknowns making it unique. Note that consistency of the system was argued for in the previous section.

(g) Judicial Bias

Appendix

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


# ..................a
time = np.array([2.0, 6.0, 9.0, 11.0, 12.0])
temp = np.array([25.0, 35.0, 45.0, 65.0, 70.0])

# --------------------b
# System of equations
A = np.array([[1,9,81, 45],
                [1,11,121, 65],
                [1,12,144, 70]])
# Row operations
A[1,:] -= A[0,:]       # R_2 -> R_2 - R_1
A[2,:] -= A[0,:]       # R_3 -> R_3 - R_1
A[1,:] = (1/2)*A[1,:] # R_2 -> 1/2*R_2
A[2,:] -= 3*A[1,:]     # R_3 -> R_3 - 3R_2
tB = np.linspace(2.0, 12.0, 100)

y_valsB = (-210)+43.3*tB -(5/3)*tB**2 # 't' is the x values



# ----------c - polynomial interpolation

a = np.vander(time, len(time)) # The lenght is equal to 5 which
    corresponds to the exercise, which gives us 4 degrees (0,1,2,3,4)

# Augment and solve, This gives us the coefficients [a,b,c] see page 214
    in notes
coefficientsC = np.linalg.solve(a, temp)

# give it some more 'doug' to work with
tC = np.linspace(2.0, 12.0, 100)

# This gives us p(x)
y_valsC = np.vander(tC, len(time)) @ coefficientsC



# -----------d One solution to rule them all
# setup a a system of equaitons for both p_1 and p_2 and for checking the
    slope, take the derivative of them both,
#subtract the derivates and set them to 0

# -----------e ALAKAZAM!
# time to construct big ass, bitch ass, matrix...fml
iCri_X = np.zeros((8,8))
```

```
iCri_Res = np.array([25,35,45,45,65,70,0,0])

iCri_X[0,:] =[1,2,4,8,0,0,0,0]
iCri_X[1,:] =[1,6,36,216,0,0,0,0]
iCri_X[2,:] =[1,9,81,729,0,0,0,0]
iCri_X[3,:] =[0,0,0,0,1,9,81,729]
iCri_X[4,:] =[0,0,0,0,1,11,121,1331]
iCri_X[5,:] =[0,0,0,0,1,12,144,1728]
iCri_X[6,:] =[0,1,18,243,0,-1,-18,-243]
iCri_X[7,1] =1 # The constraint b_1 = 0
#No need to vander before solve since our shiz is already a polynomial
   matrix
coefficients = np.linalg.solve(iCri_X, iCri_Res)

# splitting since we need to differenciate between p_1 and p_2
t1 = np.linspace(2,9,50) # p_1
t2 = np.linspace(9,12,50) # p_2


y1 = np.vander(t1, 4, increasing=True) @ coefficients[:4]
y2 = np.vander(t2, 4, increasing=True) @ coefficients[4:]


#-------------PLotting-------------#
# Exercise a
figA, drone = plt.subplots()
drone.plot(time, temp, 'o-')
drone.set_xlabel('Time(min)')
drone.set_ylabel('Temepratur(C)')
drone.set_title('Temeperatur of drone over time #Ugly')

# Exercise b
figb, b = plt.subplots()
b.plot(time, temp, 'o', label = 'Data pointies' )
b.plot(tB,y_valsB, label = 'The last 3 poly points' )
b.set_xlabel('Time(min)')
b.set_ylabel('Temepratur(C)')
b.set_title('Polynomial passing through the last 3 data points')

# Exercise c
figB, vander = plt.subplots()
vander.plot(time, temp, 'o', label = 'Data pointies')
vander.plot(tC, y_valsC, label = '4 Degrees of Poly' )
vander.set_xlabel('Time(min)')
vander.set_ylabel('Temepratur(C)')
vander.set_title('High Polynomials')


# Exercise f
figC, big = plt.subplots()
big.plot(t1, y1, label='p1(x)')
big.plot(t2, y2, label='p2(x)')
big.plot(time, temp, 'o', label = 'Data pointies')
big.set_xlabel('Time(min)')
```

```
100 big.set_ylabel('Temepratur(C)')
101 big.set_title('Splicing polynomials and fitting them together ')
102
103
104
105
106 plt.show()
```

Listing 1: Python code for handin 3