

AARHUS UNIVERSITY

COMPUTER-SCIENCE

NUMERICAL LINEAR ALGEBRA

Handin 3

Author

Søren M. DAMSGAARD

Student number

202309814

February 24, 2026



Polynomials for days!

As a postface, epilogue or early sequel to this piece of paper, the Author has become increasingly aware that making things pretty witty unicorn glitty, while keeping a formal tone, is consuming a lot of time, and is contrived for any lack of formality the reader might encounter. The Author urges the reader to use the following link if any discomfort arises in reading the paper. [Emergency Witty Unicorn Hotline](#)

(a) Plotting like a Caveman

Author goes plot plot with python

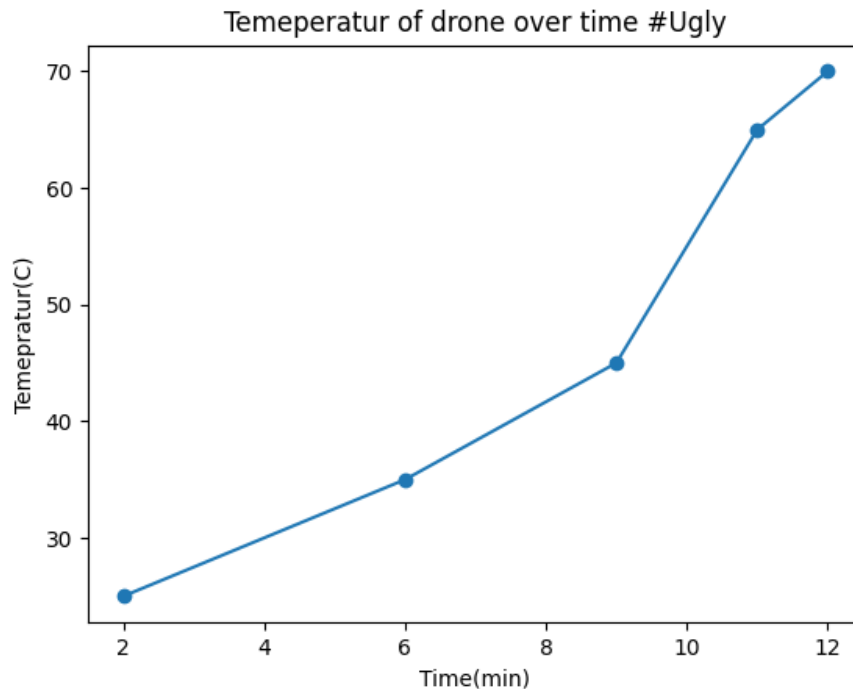


Figure 1: Plot of the data points

The code for the plot can be found in the appendix, LINE 5-7 AND 70-75

(b) System of equations, but make it polynomial

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

$$p_1(9) = a_1 + b_1 \cdot 9 + c_1 \cdot 9^2 = 45$$

$$p_1(11) = a_1 + b_1 \cdot 11 + c_1 \cdot 11^2 = 65$$

$$p_1(12) = a_1 + b_1 \cdot 12 + c_1 \cdot 12^2 = 70$$

This give the following augmented 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 the following operations:

$$R_2 \rightarrow R_2 - R_1$$

$$R_3 \rightarrow R_3 - R_1$$

$$R_2 \rightarrow \frac{1}{2}R_2$$

$$R_3 \rightarrow R_3 - 3R_2$$

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 $c_1 = -\frac{5}{3}$. The code for this can be found in the appendix, LINE 10-18. And then we plot!

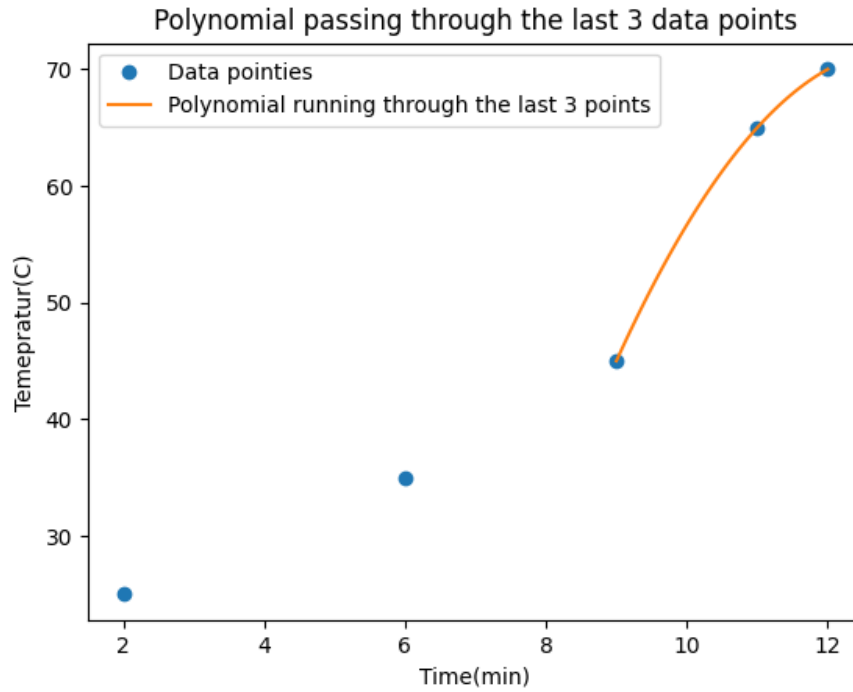


Figure 2: Plot of the polynomial running through the last three data points

the code for the plot can be found in the appendix, LINE 19-21 AND 76-83

(c) Vandermonde, Vandermonde, Vandermonde

Okay here the Author gets a bit unsure, since the use of the Vandermonde matrix is used but isn't explained throughly until chapter 16 in the notes, which hasn't been covered yet. The Vandermonde matrix essentially takes the x values and creates a matrix where each row i is the x values raised to the power of i , augment this with the y values and solve for the coefficients via `NP.VANDER()` and `NP.LINALG.SOLVE()` respectively, see Appendix LINE 25-36.

This should give the y values that allows the plotting of the polynomial of 4 degrees through all points, which is shown below.

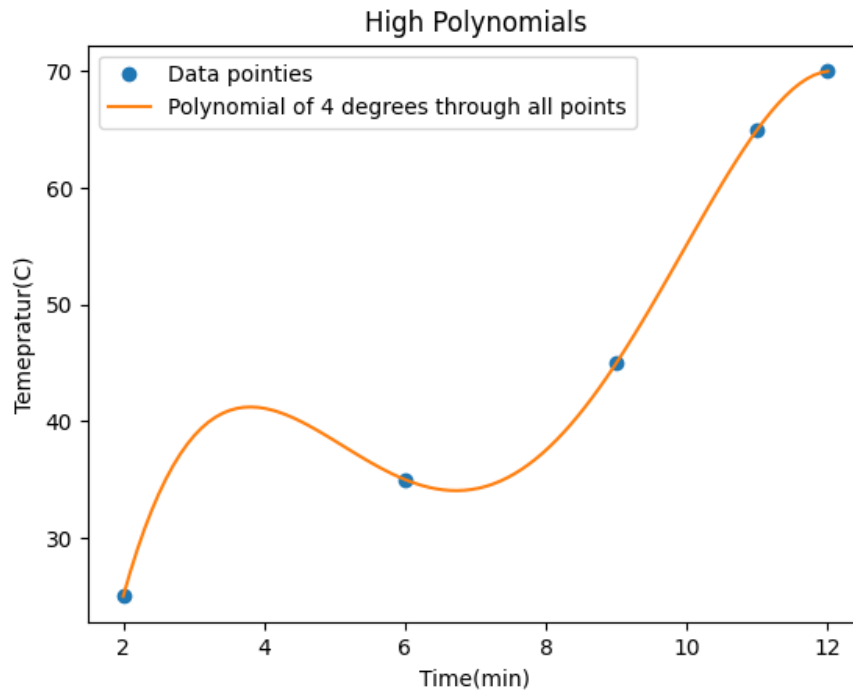


Figure 3: Plot of the polynomial of 4 degrees through all points

For plotting see Appendix LINE 84-91

(d) Too many polynomials, not enough time

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

$$\begin{aligned}p_1(2) &= a_1 + b_1 \cdot 2 + c_1 \cdot 2^2 + d_1 \cdot 2^3 = 25 \\p_1(6) &= a_1 + b_1 \cdot 6 + c_1 \cdot 6^2 + d_1 \cdot 6^3 = 35 \\p_1(9) &= a_1 + b_1 \cdot 9 + c_1 \cdot 9^2 + d_1 \cdot 9^3 = 45\end{aligned}$$

$$\begin{aligned}
p_2(9) &= a_2 + b_2 \cdot 9 + c_2 \cdot 9^2 + d_2 \cdot 9^3 = 45 \\
p_2(11) &= a_2 + b_2 \cdot 11 + c_2 \cdot 11^2 + d_2 \cdot 11^3 = 65 \\
p_2(12) &= a_2 + b_2 \cdot 12 + c_2 \cdot 12^2 + d_2 \cdot 12^3 = 70
\end{aligned}$$

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

$$\begin{aligned}
p_1'(9) &= b_1 + 2c_1 \cdot 9 + 3d_1 \cdot 9^2 \\
p_2'(9) &= b_2 + 2c_2 \cdot 9 + 3d_2 \cdot 9^2 \\
b_1 + 2c_1 \cdot 9 + 3d_1 \cdot 9^2 &= b_2 + 2c_2 \cdot 9 + 3d_2 \cdot 9^2 \\
b_1 + 2c_1 \cdot 9 + 3d_1 \cdot 9^2 - b_2 - 2c_2 \cdot 9 - 3d_2 \cdot 9^2 &= 0
\end{aligned}$$

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

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

(e) Sherlock Holmes and the One Solution

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

The Author does not want to argue for consistency, since it makes the Author sad.

(f) Math Tariffs

The assignment recommends imposing the condition $p_1'(0.0) = 0$, which will be used.

This conditions implies 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:

$$p_1'(0) = b_1 + 2c_1 \cdot 0 + 3d_1 \cdot 0^2 = b_1 \quad (1)$$

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 an equal number of equations and unknowns making it unique, according to chapter 6.4 in the notes. Note that consistency of the system was argued for in the previous section. The resulting augmented matrix is the same as the

one in (d) but with an extra row for the new condition, see Appendix LINE 44-56 for the code:

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

With this it is now possible to solve it and plot it, see Appendix LINE 57-66 AND 94-102 for the code and plotting of Figure 4 respectively.

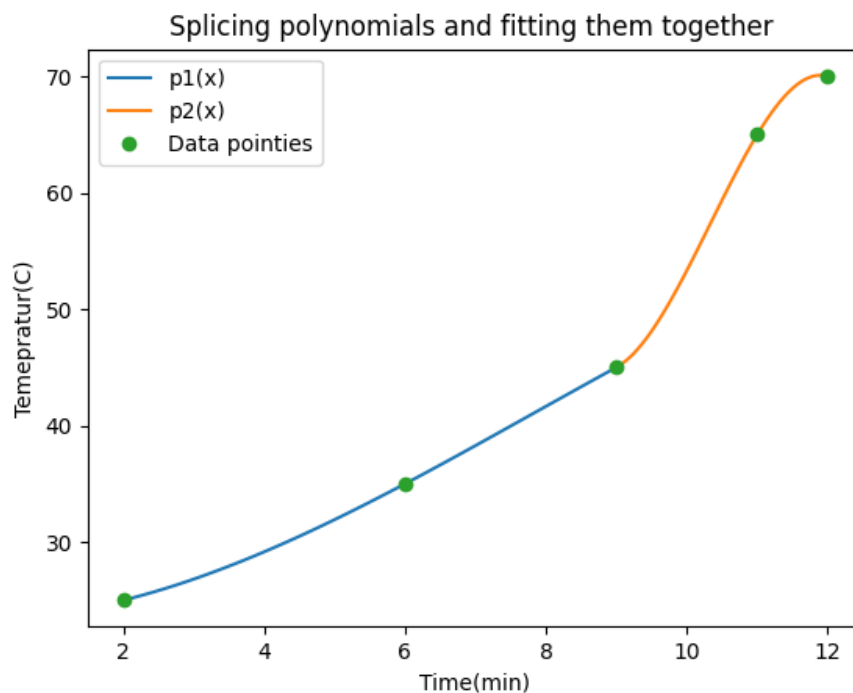


Figure 4: Plot of the spliced polynomials with the new condition

(g) Judicial Bias

Appendix

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3
4
5 # .....a
6 time = np.array([2.0, 6.0, 9.0, 11.0, 12.0])
```

```

7 temp = np.array([25.0, 35.0, 45.0, 65.0, 70.0])
8
9 # -----b
10 # System of equations
11 A = np.array([[1,9,81, 45],
12               [1,11,121, 65],
13               [1,12,144, 70]])
14 # Row operations
15 A[1,:] -= A[0,:] # R_2 -> R_2 - R_1
16 A[2,:] -= A[0,:] # R_3 -> R_3 - R_1
17 A[1,:] = (1/2)*A[1,:] # R_2 -> 1/2*R_2
18 A[2,:] -= 3*A[1,:] # R_3 -> R_3 - 3R_2
19 tB = np.linspace(9.0, 12.0, 100)
20
21 y_valsB = (-210)+(130/3)*tB -(5/3)*tB**2 # 't' is the x values
22
23
24
25 # -----c - polynomial interpolation
26
27 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)
28
29 # Augment and solve, This gives us the coefficients [a,b,c] see page 214
    in notes
30 coefficientsC = np.linalg.solve(a, temp)
31
32 # give it some more 'doug' to work with
33 tC = np.linspace(2.0, 12.0, 100)
34
35 # This gives us p(x)
36 y_valsC = np.vander(tC, len(time)) @ coefficientsC
37
38
39
40 # -----d One solution to rule them all
41 # setup a a system of equaitons for both p_1 and p_2 and for checking the
    slope, take the derivative of them both,
42 #subtract the derivates and set them to 0
43
44 # -----f ALAKAZAM!
45 # time to construct big ass, bitch ass, matrix...fml
46 iCri_X = np.zeros((8,8))
47 iCri_Res = np.array([25,35,45,45,65,70,0,0])
48
49 iCri_X[0,:] =[1,2,4,8,0,0,0,0]
50 iCri_X[1,:] =[1,6,36,216,0,0,0,0]
51 iCri_X[2,:] =[1,9,81,729,0,0,0,0]
52 iCri_X[3,:] =[0,0,0,0,1,9,81,729]
53 iCri_X[4,:] =[0,0,0,0,1,11,121,1331]
54 iCri_X[5,:] =[0,0,0,0,1,12,144,1728]
55 iCri_X[6,:] =[0,1,18,243,0,-1,-18,-243]
56 iCri_X[7,1] =1 # The constraint b_1 = 0

```

```

57 #No need to vander before solve since our shiz is already a polynomial
    matrix
58 coefficients = np.linalg.solve(iCri_X, iCri_Res)
59
60 # splitting since we need to differenciate between p_1 and p_2
61 t1 = np.linspace(2,9,50) # p_1
62 t2 = np.linspace(9,12,50) # p_2
63
64
65 y1 = np.vander(t1, 4, increasing=True) @ coefficients[:4]
66 y2 = np.vander(t2, 4, increasing=True) @ coefficients[4:]
67
68
69 #-----Plotting-----#
70 # Exercise a
71 figA, drone = plt.subplots()
72 drone.plot(time, temp, 'o-')
73 drone.set_xlabel('Time(min)')
74 drone.set_ylabel('Temepratur(C)')
75 drone.set_title('Temepratur of drone over time #Ugly')
76 # Exercise b
77 figb, b = plt.subplots()
78 b.plot(time, temp, 'o', label = 'Data pointies' )
79 b.plot(tB,y_valsB, label = 'Polynomial running through the last 3 points'
    )
80 b.set_xlabel('Time(min)')
81 b.set_ylabel('Temepratur(C)')
82 b.set_title('Polynomial passing through the last 3 data points')
83 b.legend()
84 # Exercise c
85 figB, vander = plt.subplots()
86 vander.plot(time, temp, 'o', label = 'Data pointies')
87 vander.plot(tC, y_valsC, label = 'Polynomial of 4 degrees through all
    points' )
88 vander.set_xlabel('Time(min)')
89 vander.set_ylabel('Temepratur(C)')
90 vander.set_title('High Polynomials')
91 vander.legend()
92
93
94 # Exercise f
95 figC, big = plt.subplots()
96 big.plot(t1, y1, label='p1(x)')
97 big.plot(t2, y2, label='p2(x)')
98 big.plot(time, temp, 'o', label = 'Data pointies')
99 big.set_xlabel('Time(min)')
100 big.set_ylabel('Temepratur(C)')
101 big.set_title('Splicing polynomials and fitting them together ')
102 big.legend()
103
104
105

```



```
106 plt.show()
```

Listing 1: Python code for handin 3