AMS 326 Exam 1 - Mehadi Chowdhury (115112722)

The code alongside the report can be found on the following public repo on github: https://github.com/EmceeCiao/AMS_326_Exam1

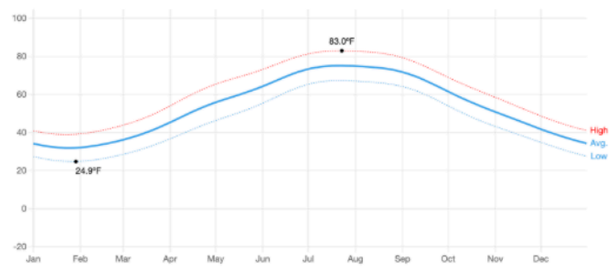
I have also zipped up the files with the submission for convenience if needed. There will be both regular python files and this Jupyter notebook provided.

Background For Problem 1

Problem T1.1 (100 Points) The monthly temperature averages, along with min and max, in Stony Brook are tabulated and graphed here. To simplify the project, we assume each month has precisely 31 days and the average for a given month can be considered as the temperature of the 16th day.

Note: You need to do reasonably amount of original coding for solving this problem.

	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
High	40°F	41°F	48°F	60°F	69°F	77°F	83°F	82°F	75°F	64°F	54°F	45°F
Average	33°F	34°F	40°F	51°F	60°F	69°F	75°F	74°F	67°F	56°F	47°F	38°F
Low	26°F	26°F	32°F	42°F	51°F	61°F	67°F	66°F	60°F	49°F	39°F	31°F



Thus, I start a table of 12 data points (only showing data for four months, you do the rest):

Month	Jan	Feb	Mar	Apr	...
t (Day from Jan 1)	16+31*0	16+31*1	16+31*2	16+31*2	...
Average Temp (F)	33	34	40	51	...

Please do (60 points for (1), 20 points for each of (2) and (3)):
(1) Fit the given 12 data points in a polynomial,

$$P_3(t) = a_0 + a_1 t + a_2 t^2 + a_3 t^3$$

(2) Calculate the temperatures on June 4 and Dec. 25 with your fit.
(3) Calculate the day(s) when the temperature reaches 64.89, again, with your fit.

Problem 1 - Task 1 Description

Given 12 data points of the average temperatures on the 16th day for each month in a year at StonyBrook fit the data points in the cubic polynomial $P_3(t) = a_0 + a_1 t + a_2 t^2 + a_3 t^3$

Algorithm Description

The algorithm I will be using to create the fit for this cubic polynomial will be Least Square Error curve fitting. The formula and reasoning is as follows:

Given an inconsistent system of form Ax = b, we can solve $A^T A\bar{x} = A^T b$ for the least squares solution. In this case since we are trying to do a cubic fit of the form $a_0 + a_1 t + a_2 t^2 + a_3 t^3$, our Ax = b will be the system of equations created by plugging in our points into the cubic function we were given.

Specifically we are passing in $1, t, t^2, t^3$ to create matrix A as this matrix multiplied by our coefficients represented by x should lead to the y values of our points, otherwise represented by b here. We can now solve our inconsistent system here for what the coefficients of this function will be as we can isolate for $\bar{x}$ giving us the values for a_0, a_1, a_2, a_3. With this we now know the fit function and can compute it at a specific point.

The code below will be an adaptation of my quadratic fit code I wrote for hw 1!

Code

In [26]:
```
import numpy as np

def cubic_fit(t_values, y_values, point_value):
    """
    Function to both create the cubit fit function and print out the results achiev

    Input:
    t_values: Array of t values passed in for the points (A point is of the form (t
    y_values: Array of y values passed in for the points of the form (A point is of
    point_value: t value we are evaluating our cubic fit for

    Output:
    prints the cubic fit equation and the value of computing the cubic fit for the
    """

    t_values = np.array(t_values)
    y_values = np.array(y_values)

    n = len(t_values)

    A = np.vstack([np.ones(n), t_values, t_values**2, t_values**3]).T

    # Need to create A^TA and A^Tb and then solve these

    A_TA = np.dot(A.T, A)

    A_Ty = np.dot(A.T, y_values)

    #Solves for the coefficients
    coeffs = np.linalg.solve(A_TA, A_Ty)

    a_0 = coeffs[0]
    a_1 = coeffs[1]
    a_2 = coeffs[2]
    a_3 = coeffs[3]

    print(f"P_3(t) = {a_0} + {a_1}t + {a_2}t^2 + {a_3}t^3\n")
    # ans = a_0 + (a_1*point_value) + (a_2 * point_value**2) + (a_3 * point_value**
    # print(f"P_3(t = {point_value}) = {ans}")
    return coeffs

# For loop ran to see what values should be inputted into the array
# for i in range(1, 13):
#     print(16 + (31 * i))

t_values = np.array([16, 47, 78, 109, 140, 171, 202, 233, 264, 295, 326, 357])
y_values = np.array([33, 34, 40, 51, 60, 69, 75, 74, 67, 56, 47, 38])


coeffs = cubic_fit(t_values, y_values, 6)
```

```
P_3(t) = 25.941780752899096 + 0.1871394692380645t + 0.0009576875408839653t^2 + -3.99
0504477228924e-06t^3
```

Results

The fit to a cubic polynomial for the 12 datapoints was:
P_3(t) = 25.941780752899096 + 0.1871394692380645t + 0.0009576875408839653t^2 + -3.990504477228924e-06t^3

Performance

The perfomance of this algorithm is the same as in our homework. The time complexity of this code generically is around O(m * n^2) where we are given the m * n matrix A. This is because multiplying and m * n matrix by it's transpose which is an n * m matrix will require

(m * n * n) operations, which will dominate the time complexity. However in this case talking about the code above, we can simplify the time complexity a bit more, since n is a constant 4 here due to the cubic function only having 4 terms, we can say the time complexity of the code is O(m * 16). The space complexity of this code is O(m * n) as that's the amount of space needed to store these matrices.

Problem 1 - Task 2

Calculate the temperatures of June 4th and Dec 25th with our fit found in part 1.

Algorithm/Solution Description

Using the coefficients for the cubic fit I returned in part 1, I'll be calculating june 4th and dec 25, by representing them as a whole number calculated by finding the number of days since Jan 1. I found on geeks for geeks (https://www.geeksforgeeks.org/numpy-polyval-in-python/library) that the np.polyval when given coefficients will construct the polynomial for us to evaluate a point on, rather than having us construct it each time, making the code a bit cleaner. However the underlying implementation is still using Least Square Error Curve fitting.

Code

In [ ]:
```
import numpy as np

def cubic_fit(t_values, y_values, point_value):
    """
    Function to both create the cubic fit function and print out the results achiev

    Input:
    t_values: Array of t values passed in for the points (A point is of the form (t
    y_values: Array of y values passed in for the points of the form (A point is of
    point_value: t value we are evaluating our fit for

    Output:
    prints the  fit equation and the value of computing the fit for the t value pas
    """

    t_values = np.array(t_values)
    y_values = np.array(y_values)

    n = len(t_values)

    A = np.vstack([np.ones(n), t_values, t_values**2, t_values**3]).T

    # Need to create A^TA and A^Tb and then solve these

    A_TA = np.dot(A.T, A)

    A_Ty = np.dot(A.T, y_values)

    #Solves for the coefficients
    coeffs = np.linalg.solve(A_TA, A_Ty)

    a_0 = coeffs[0]
    a_1 = coeffs[1]
    a_2 = coeffs[2]
    a_3 = coeffs[3]

    # print(f"P_3(t) = {a_0} + {a_1}t + {a_2}t^2 + {a_3}t^3\n")
    # ans = a_0 + (a_1*point_value) + (a_2 * point_value**2) + (a_3 * point_value**
    # print(f"P_3(t = {point_value}) = {ans}")
    return coeffs

# For loop ran to see what values should be inputted into the array
# for i in range(1, 13):
#     print(16 + (31 * i))
```

```
t_values = np.array([16, 47, 78, 109, 140, 171, 202, 233, 264, 295, 326, 357])
y_values = np.array([33, 34, 40, 51, 60, 69, 75, 74, 67, 56, 47, 38])


coeffs = cubic_fit(t_values, y_values, 6)
june_4th = (5 * 31) + 4
dec_25th = (11 * 31) + 25
temp_4th = np.polyval(coeffs[::-1], june_4th)
temp_25th = np.polyval(coeffs[::-1], dec_25th)
print("Temperature of June 4th: ", temp_4th)
print("Temperature of Dec 25th: ", temp_25th)
```

```
Temperature of June 4th:  63.86770803631579
Temperature of Dec 25th:  27.07678022356906
```

Results

Using the cubic function I calculated in part 1, I got the following results for the temp on June 4th and Dec 25th:

Temperature of June 4th: 63.86770803631579
Temperature of Dec 25th: 27.07678022356906

Performance

Again the performance is still the same as in part 1 as it's using the underlying math to solve for the values: The time complexity of this code generically is around O(m * n^2) where we are given the m * n matrix A. This is because multipling and m * n matrix by it's transpose which is an n * m matrix will require (m * n * n) operations, which will dominate the time complexity. However in this case talking about the code above, we can simplify the time complexity a bit more, since n is a constant 4 here due to the cubic function only having 4 terms, we can say the time complexity of the code is O(m * 16). The space complexity of this code is O(m * n) as that's the amount of space needed to store these matrices.

But if we are strictly considering already having the function then calculating the value at this point is O(1) and the space is O(1) as we are just storing the function and values.

Problem 1 - Task 3

Calculate the days using your fit that the temperature reaches 64.89.

Algorithm/Solution Description

Using our cubic fit function and the nice np.polyval function from numpy, we can iterate through all days in the year and find the days where the function reaches 64.89. In this case our year will be a bit different as we are assuming that there are 31 days in each month for simplicity sake. Additionally the tolerance we will have to set will have to be a bit higher than we usually provide as I will be providing the days as whole numbers, so I've decided to print days that are within less than 0.25 of the desired temp as these would likely have reached the desired temp at some point.

Code

In [ ]:
```
import numpy as np

def cubic_fit(t_values, y_values, point_value):
    """
    Function to both create the cubit fit function and print out the results
    achieved when evaluating the function at a specific point_value

    Input:
    t_values: Array of t values passed in for the points
              (A point is of the form (t, y))
    y_values: Array of y values passed in for the points of the form
              (A point is of the form (t, y))
```

```
    point_value: t value we are evaluating our quadratic fit for

    Output:
    prints the quadratic fit equation and the value of computing the quadratic fit
    """

    t_values = np.array(t_values)
    y_values = np.array(y_values)

    n = len(t_values)

    A = np.vstack([np.ones(n), t_values, t_values**2, t_values**3]).T

    # Need to create A^TA and A^Tb and then solve these

    A_TA = np.dot(A.T, A)

    A_Ty = np.dot(A.T, y_values)

    #Solves for the coefficients
    coeffs = np.linalg.solve(A_TA, A_Ty)

    a_0 = coeffs[0]
    a_1 = coeffs[1]
    a_2 = coeffs[2]
    a_3 = coeffs[3]

    # print(f"P_3(t) = {a_0} + {a_1}t + {a_2}t^2 + {a_3}t^3\n")
    # ans = a_0 + (a_1*point_value) + (a_2 * point_value**2) + (a_3 * point_value**
    # print(f"P_3(t = {point_value}) = {ans}")
    return coeffs

# For loop ran to see what values should be inputted into the array
# for i in range(1, 13):
#     print(16 + (31 * i))

t_values = np.array([16, 47, 78, 109, 140, 171, 202, 233, 264, 295, 326, 357])
y_values = np.array([33, 34, 40, 51, 60, 69, 75, 74, 67, 56, 47, 38])


coeffs = cubic_fit(t_values, y_values, 6)
total_days = 31 * 12

days = []
for i in range (1, total_days + 1):
    temp_estimate = np.polyval(coeffs[::-1], i)
    if (abs(temp_estimate - 64.89)) < 0.25:
        print(f"Temperature of Day {i}: {temp_estimate}, is less than 0.25 away 64.
        days.append(i)
print("\n")
for i in days:
    print(f"Day {i} reaches within 0.25 of 64.89")
```

```
Temperature of Day 164: 64.78872602675074, is less than 0.25 away from our target 6
4.89
Temperature of Day 165: 64.96699155297371, is less than 0.25 away from our target 6
4.89
Temperature of Day 284: 64.92492764527968, is less than 0.25 away from our target 6
4.89
Temperature of Day 285: 64.68801303761553, is less than 0.25 away from our target 6
4.89


Day 164 reaches within 0.25 of 64.89
Day 165 reaches within 0.25 of 64.89
Day 284 reaches within 0.25 of 64.89
Day 285 reaches within 0.25 of 64.89
```

Results

I got the following days that approximately reach 64.89 as their temperature is around 0.25 off from the desired:

Day 164 reaches within 0.25 of 64.89
Day 165 reaches within 0.25 of 64.89
Day 284 reaches within 0.25 of 64.89
Day 285 reaches within 0.25 of 64.89

Performance

Again the dominating time complexity of O(m * n * n) is at play here as that's the time complexity for us to create the cubic polynomial fitting, where the matrix being created is of size m * n and the space complexity with it is O(m * n) as that's necessary to store the matix. However if we only care about the performance after computing this, then the overall time complexity for this program is O(d) where d is the number of days in our year which is 372 under the assumption that each month has 31 days and it's space complexity is minimal as we only need an array large enough to store our answers.