hw6

jn2294@nyu.edu

April 2022

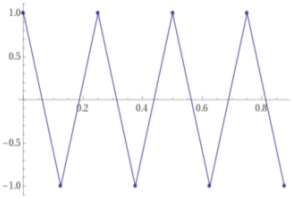
Problem 6.1

Description

Problem 6.1: Given the following 8 data points

t	0	$\frac{1}{8}$	$\frac{2}{8}$	$\frac{3}{8}$	$\frac{4}{8}$	$\frac{5}{8}$	$\frac{6}{8}$	$\frac{7}{8}$
x	1	-1	1	-1	1	-1	1	-1

which can be plotted as (although we do not know the values in-between the points):



Please find the trigonometric interpolating function (and plot your function with these 8 points shown).

Figure 1: Description for problem 3.1

Algorithm

For an even integer n, let $t_j = c + j(d-c)/n$ for $j = 0, \ldots, n-1$, and let $x = (x_0, \ldots, x_{n-1})$ denote a vector of n real numbers. Define $\vec{a} + \vec{b}i = F_n x$, where F_n is the Discrete Fourier Transform. Then the function

$$P_n(t) = \frac{a_0}{\sqrt{n}} + \frac{2}{\sqrt{n}} \sum_{k=1}^{n/2-1} \left(a_k \cos \frac{2k\pi(t-c)}{d-c} - b_k \sin \frac{2k\pi(t-c)}{d-c} \right) + \frac{a_{n/2}}{\sqrt{n}} \cos \frac{n\pi(t-c)}{d-c} \tag{10.19}$$

satisfies $P_n(t_j) = x_j$ for $j = 0, \ldots, n-1$. ■

Figure 2: Algorithm for problem 6.1

Code

```
def formatInterpolation(self,a,b,c,d):
    # Utilize the DFT THM
    n = a.shape[0]
    if n % 2 == 0:
        coeff = []
        for k in range(1,n//2):
            coeff.extend([a[k],-b[k]])

        period_func = []

        for k in range(1,n//2):
            period_func.extend([sympy.cos(2*sympy.pi*k*(t-c)/(d-c)),sympy.sin(2*sympy.pi*k*(t-
                c)/(d-c))])

        middle = sum([coeff[i]* period_func[i] for i in range(len(coeff))])

        func = a[0] + 2*middle + a[n//2]*sympy.cos(n*sympy.pi*(t-c)/(d-c))
    return t,func
```

Results

- **Trigonometric Interpolation Plot:**

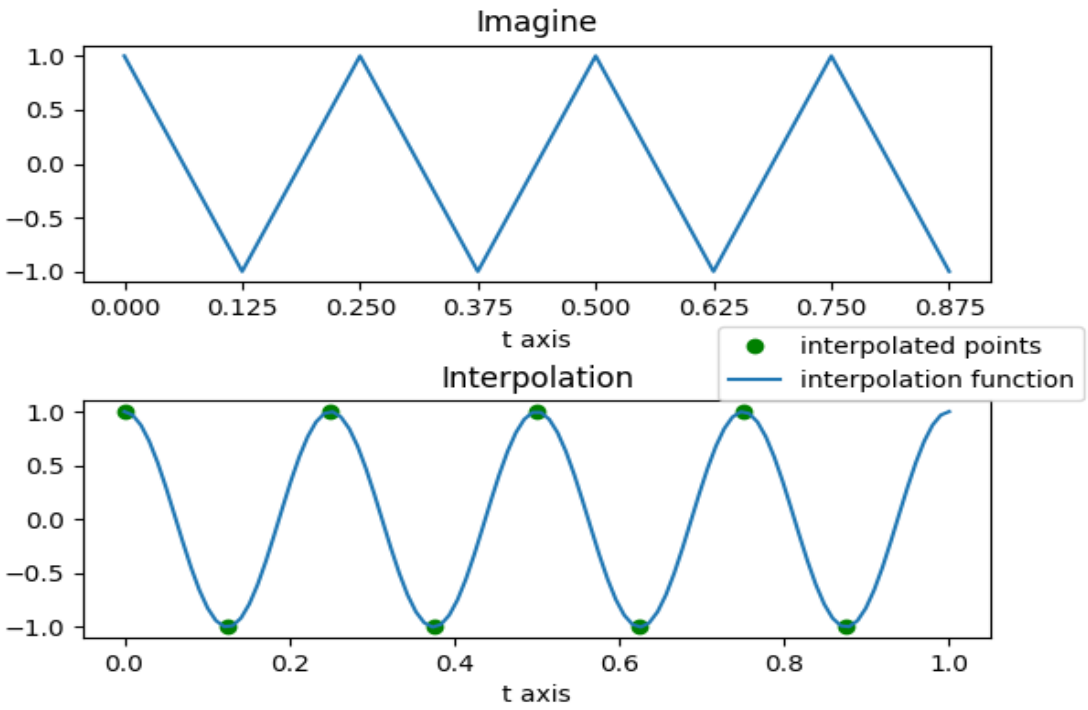


Figure 3: Speed test result for 6.1

- **Interpolation Function:**
  $7.85046229341887e^{-17} * sin(2\pi t) + 2.35513868802566e^{-16} * sin(4\pi t) + 8.63550852276076e^{-16} * sin(6\pi t) - 1.17756934401283e^{-16} * cos(2\pi t) - 1.96261557335472e^{-16} * cos(4\pi t) - 3.14018491736755e^{-16} * cos(6\pi t) + 1.0 * cos(8\pi t)$

Performance

No performance analysis is performed.

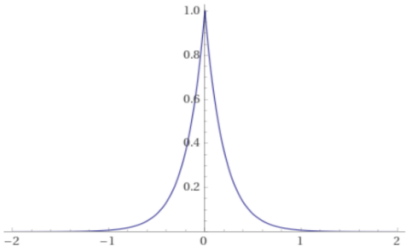
Problem 6.2

Description

Problem 6.2: Given the following function

$$f(t) = e^{-\frac{|t|}{T}}$$

where T is some positive number and, for convenience, please set it as $T = 0.1984$. The function looks like



Evenly select $N = 1024 = 2^{10}$, in $t_i \in [-2, 2]$, to evaluate $f(t_i)$ where $i = 0, 1, 2 \ldots, N-1$.

(1) Perform DFT on this vector of N dimensions whose elements are $f(t_0), f(t_1), f(t_2), \ldots$
(2) Perform FFT on the same vector as in (1).
(3) Plot your vector in the Fourier space for both cases.

Figure 4: Description for problem 6.1

Algorithm

$$F_{2n} = \begin{bmatrix} I & D \\ I & -D \end{bmatrix} \begin{bmatrix} F_n & 0 \\ 0 & F_n \end{bmatrix} P$$

Figure 5: Algorithm For FFT in Matrix Form

This algorithm utilizes the matrix representation of the FFT, which is discovered by Gaussian$P$ is the transpose matrix that rearranges the input vector into two subparts(one with even indices and the other with odd indices).F_n is the fourier matrix bu definition. I is the identity matrix of size n, and D is the diagonal matrix with w^n on the diagonal. The following is the code for FFT Transformation.

Code

```
def computeFn(self,n):
    """
    We will use the matrix form of multiplication
    """
    if n == 1:
        return np.array([[1]])

    # 1. Get the D matrix of size n = 2^m/2
    Dn_2 = np.diag(np.power(self.w_ft,np.arange(0,n//2)))

    # 2. Get the transpose matrix to align the odd and even terms
    Pn_2 = np.zeros((n, n))
    Pn_2[np.arange(0, n // 2), np.arange(0, n, 2)] = 1
    Pn_2[np.arange(n // 2, n), np.arange(1, n, 2)] = 1

    # 3. Get the identity matrix of size n
    In_2 = np.eye(n // 2)

    # 4. Compute the Fn
    Fn_2 = self.computeFn(n//2)

    Fn = np.block([[In_2,Dn_2],[In_2,-Dn_2]]) @ np.block([[Fn_2,np.zeros((n//2,n//2))],[np.
        zeros((n//2,n//2)),Fn_2]]) @ Pn_2

    return Fn
```

We utilize the recursive programming, just as what we have done for Strassen Algorithm, but this time we don't have to pass and copy the whole matrix into the recursive function(huge drain on the main memory and affects computing speed). Therefore, we can see stark contrast between DFT and FFT.

Results

- **The vector plotted in the Fourier Space**

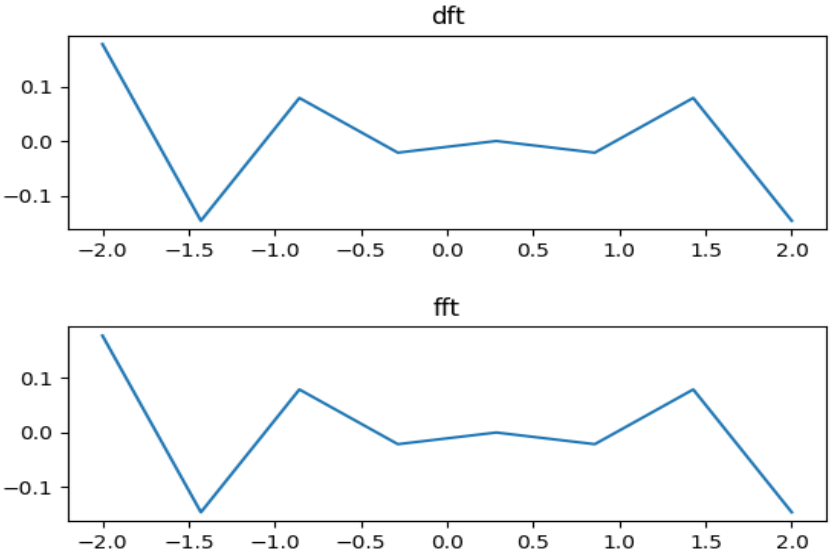


Figure 6: Plot Result For Problem 6.2

Performance

```
%timeit performDFT(ft,n)
666 ms ± 43.3 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
%timeit performFFT(ft,n)
195 ms ± 5.67 ms per loop (mean ± std. dev. of 7 runs, 10 loops each)
```

Figure 7: Speed test result for 6.2

As expected, the Fast Fourier Transformation computes much faster than the original Discrete Fourier Transformation.

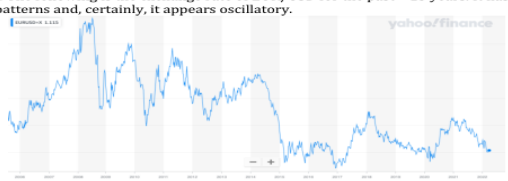
```
def performDFT(self,x,n):
    DFT = np.zeros((n,n),dtype = np.complex)

    for i in range(n):
        for j in range(n):
            DFT[i, j] = self.w_ft ** (i * j)

    DFT = (1 / np.sqrt(n)) * DFT
    DFT_x =  np.dot(DFT, x)

    return DFT_x, DFT
```

```
def performFFT(self,x,n):
    Fn = (1/np.sqrt(n)) * self.computeFn(n)
    return Fn @ x, Fn
```

Problem 6.3

Description

Problem 6.3: The following is the exchange rate of EUR/USD for the past ~15 years. It has some interesting patterns and, certainly, it appears oscillatory.



At the following web site, you can find the daily rates for an entire past year:
https://finance.yahoo.com/quote/EURUSD%3DX/history?p=EURUSD%3DX

Please glean the exchanges for the last day of the months in the following table:

Time "t"	Last day of month	EUR/USD "x"	
0	04/2021		
1	05		
2	06		
3	07		
4	08		
5	09		
6	10		
7	11		
8	12		
9	01/2022		
10	02		
11	03		

Find the trigonometric interpolating function for the rates in these 12 months.

Figure 8: Description for problem 6.1

Algorithm

For an even integer n, let $t_j = c + j(d-c)/n$ for $j = 0, \ldots, n-1$, and let $x = (x_0, \ldots, x_{n-1})$ denote a vector of n real numbers. Define $\vec{a} + \vec{b}i = F_n x$, where F_n is the Discrete Fourier Transform. Then the function

$$P_n(t) = \frac{a_0}{\sqrt{n}} + \frac{2}{\sqrt{n}} \sum_{k=1}^{n/2-1} \left(a_k \cos \frac{2k\pi(t-c)}{d-c} - b_k \sin \frac{2k\pi(t-c)}{d-c} \right) + \frac{a_{n/2}}{\sqrt{n}} \cos \frac{n\pi(t-c)}{d-c} \tag{10.19}$$

satisfies $P_n(t_j) = x_j$ for $j = 0, \ldots, n-1$. ■

Figure 9: Algorithm for problem 6.3

Code

```
def problem3(self):
    data = pd.read_csv(self.csv_path)
    data["Date"] = pd.to_datetime(data.Date)
    data.set_index("Date",inplace=True)

    # The last data entry every month(Adjusted Close Price)
    sampled_data = data[["Adj Close"]].resample("M").last()


    # Plot the dataset
    plt.plot(sampled_data)
    plt.xticks(sampled_data.index,rotation=45)
    plt.show()


    # Perform trigonometric interpolation function
    periodic_points = sampled_data["Adj Close"].to_numpy()[:-1]
    n = len(periodic_points)

    self.problem1(periodic_points,0,12,n)
```

Here we utilize the same method of trigonometric interpolation as in problem1 since we have even number of points in both scenarios.

Results

- **Gleaned Data:**

Date	Adj Close
2021-04-30	1.212709
2021-05-31	1.219007
2021-06-30	1.190193
2021-07-31	1.189300
2021-08-31	1.179690
2021-09-30	1.160160
2021-10-31	1.168361
2021-11-30	1.129344
2021-12-31	1.132503
2022-01-31	1.115237
2022-02-28	1.118105
2022-03-31	1.116184
2022-04-30	1.086602

Figure 10: Gleaned Data

- **Trigonometric Interpolation Plot:**

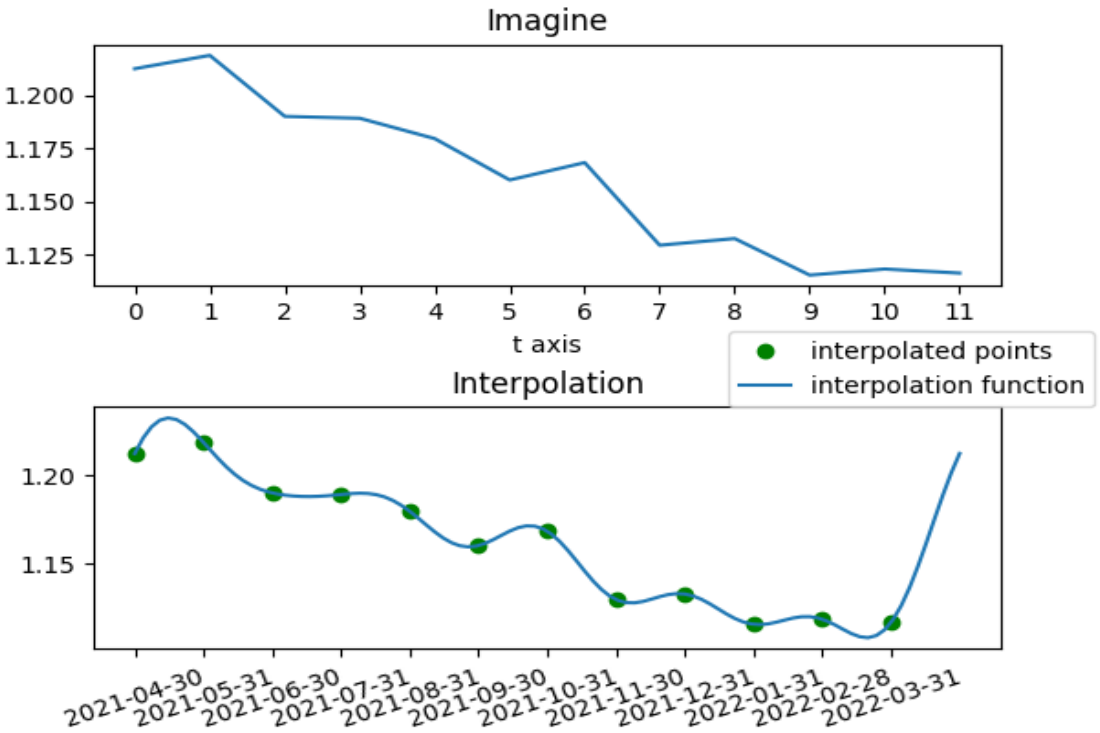


Figure 11: Gleaned Data

- **Interpolation Function:**
$0.0406962800060649 * sin(\pi t/6) + 0.0139874649716572 * sin(\pi t/3) + 0.0099293333333336 * sin(\pi t/2) + 0.00679916544511191 * sin(2\pi t/3) + 0.00626455332726901 * sin(5\pi t/6) + 0.0136611004371162 * cos(\pi t/6) + 0.0131058333333329 * cos(\pi t/3) + 0.00804049999999957 * cos(\pi t/2) + 0.010502333333333 * cos(2\pi t/3) + 0.000472399562883156 * cos(5\pi t/6) + 0.00602741666666651 * cos(\pi t) + 1.16089941666667$

Performance

No performance analysis is performed.