Umeå Universitet
Institutionen för matematik och matematisk statistik October 20, 2022

Partial Differential Equation

Theodor Jonsson (thjo0148@student.umu.se)
Simon Edvardsson (sied0020@student.umu.se)
Mikkel Lindberg (jomi0065@student.umu.se)

Mathematical Methods of Physics
Teacher: Mats Bodin

Theodor Jonsson (thjo0148@student.umu.se)
Simon Edvardsson (sied0020@student.umu.se)
Mikkel Lindberg (jomi0065@student.umu.se) October 20, 2022

1 Introduction

1.1 Computer lab signification

The computer lab as a whole revolves around numerical methods used to computed discrete Fourier transforms and extract the coefficients. The assignment consisted of 4 different tasks which tests speed and accuracy.

1.2 Theory

All theory used to complete the four assignments are present in the computer lab description.

1.3 Matlab

The code used for all the tasks can be found in the appendix A.

2 Tasks

2.1 Task 1

The first task asks to complete a Discrete Fourier Transform and Inverse Discrete Fourier Transform to transform a signal from time domain to frequency domain and back. Using the predetermined value of $\mathbf{y} = (0, 3, -2\sqrt{3}, 6, 2\sqrt{3}, 3)$ and applying our implemented **mydft.m** to transform the signal to frequencies results in the following vector:

$$\mathbf{z} = (2, -\tfrac{1}{2} + i, \tfrac{1}{2} - i, -2, \tfrac{1}{2} + i, -\tfrac{1}{2} - i) = \mathbf{mydft}(\mathbf{y}).$$

Applying the inverse DFT on the same input vector $\mathbf{y}$ yields:

$$\mathbf{w} = (12, -3 - 6i, 3 + 6i, -12, 3 - 6i, -3 + 6i) = \mathbf{myidft}(\mathbf{y}).$$

2.2 Task 2a

The goal of task 2a is to compute the Fourier coefficients for the given function:

$$f(x) = 3 - 2 * cos(15x) + 4 * sin(20x). \tag{1}$$

By inspection the resulting exact coefficients should be; $a_0 = 3, a_{15} = -2, b_{20} = 4$. When computing the Fourier coefficients using **myfouriercoeff.m** the following non zero coefficients are generated:

Table 1: Computed Fourier coefficients using $N = 2^5$ data points.

n	a_n	b_n
0	3.0	0.0
12	0.0	-4.0
15	-2.0	0.0

Table 2: Computed Fourier coefficients using $N = 2^8$ data points.

n	a_n	b_n
0	3.0	0.0
15	-2.0	0.0
20	0.0	4.0

As displayed in tables above the values of b_n is faulty when using $N = 2^5$ individual data points in the interval. This is because there are not enough provided points to generate 20 Fourier sine coefficients. When using N data points a total of $\frac{N}{2} - 1$ Fourier cosine coefficients along with $\frac{N}{2} - 1$ Fourier sine coefficients and an additional constant term a_0 will be generated. And since $\frac{2^5}{2} - 1 = 15 \ngeq 20$ there will be no $20^{\mathbf{th}}$ cosine term.

2.3 Task 2b

Using the same function as in section 2.2 to compute the Fourier coefficients of the more intricate function:

$$g(x) = |cos(x)| \; x \in [0, 2\pi]. \tag{2}$$

Given that the Fourier coefficients can be expressed:

$$a_0 = \frac{2}{\pi}, \; a_{2n} = -\frac{4}{\pi}\frac{(-1)^n}{4n^2 - 1}, \; a_{2n+1} = 0, \; b_n = 0 \tag{3}$$

Plotting the computed coefficients alongside the exact Fourier coefficients results in the following figure:

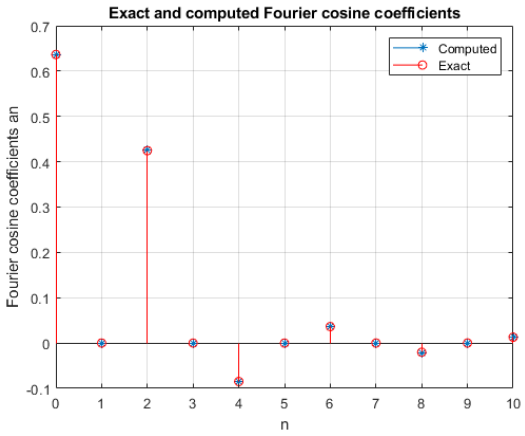

Figure 1: A plot of the computed Fourier coefficients and exact coefficients using equation (3) and 2^6 data points on the given interval.

2.4 Task 3a

The function given in the assignment is represented as:

$$h(x) = \begin{cases} 1 & 0 < x < \pi \\ -1 & \pi < x < 2\pi \\ 0 & x = 0 \text{ or } x = \pi \end{cases} \tag{4}$$

The function can be exactly represented by an infinite Fourier series but approximated with a finite number of Fourier coefficients. Using $N = 2^8$ data points to approximate M coefficients along a_0 results in the following plot:

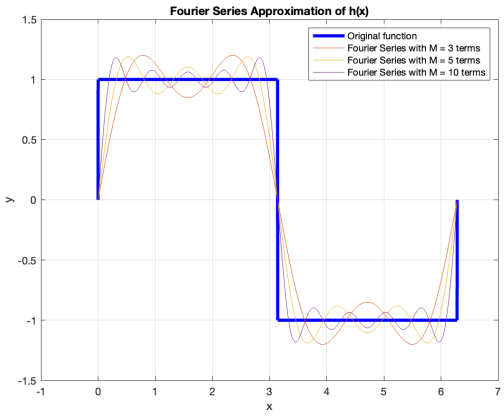

Figure 2: A plot of the computed Fourier series approximation of equation (4) using 2^8 data points on the given interval.

Using visual approximation, the overshoot of the Fourier series approximation above $h(x) \equiv 1$ is slightly below 0.2 for $M \in \{3, 5, 10\}$. On the interval $x \in (0, \pi)$.

2.5 Task 3b

The task revolves around evaluating the overshoot of Fourier series approximation as M increases. Using $M \in \{1, 2..., 100\}$ coefficients to approximate $h(x)$ and $N = 2^{11}$ data points in the interval results in the following error, $E(M)$, development as $M \to 100$.

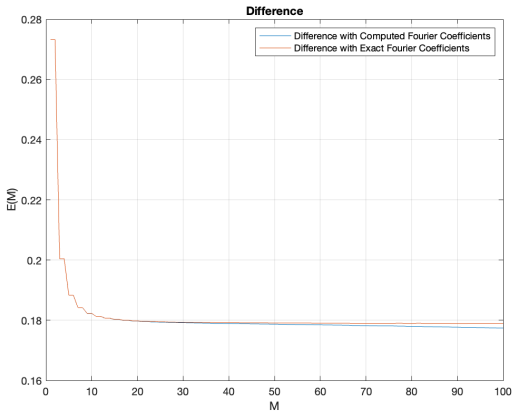

Figure 3: Figure of the difference between the Fourier series approximation.

Using the figure above it could be assumed that the difference between the Fourier series and the actual function $h(x)$ using tapers of and approaches $E(M) \approx 0.18$ as $M \to \infty$. Therefore there does not exist an M such that for $\forall m \geq M$ then $|f_m(x) - f(x)| < \epsilon$ for any $\epsilon > 0$. This means that the Fourier series does not satisfy the definition for uniform convergence.

2.6 Task 4a

The function **mydft** compared to the command FFT is slower and the time to run the function is significantly more affected by the size of the vector. The time for computing the function **mydft** gets exponential longer as the vector grows larger. While the value returned by the command FFT has a negligible change in time no matter the size of the vector.

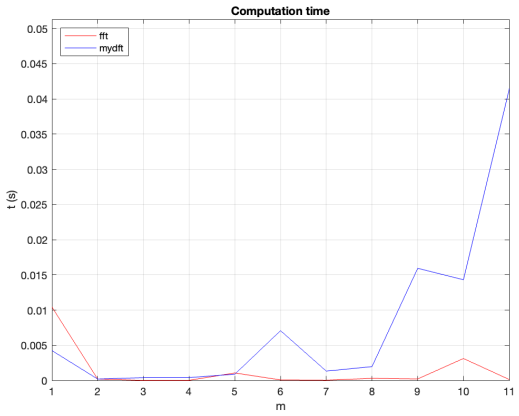

Figure 4: Figure of the computation time required for the Fast Fourier Transform (FFT) and the Discrete Fourier transform implemented in Task 1a under section **??**.

The figure above displays the performance difference of the two methods, as FFT is significantly faster and seems to only require $\mathcal{O}(n)$ time complexity relative to **mydft**.

2.7 Task 4b

When changing the threshold for the frequency, to ignore smaller frequencies, one can observe that the size of the compress file gets significant smaller. This is because the function compresses a bit of the data, and the output will then be smaller than the given input data.

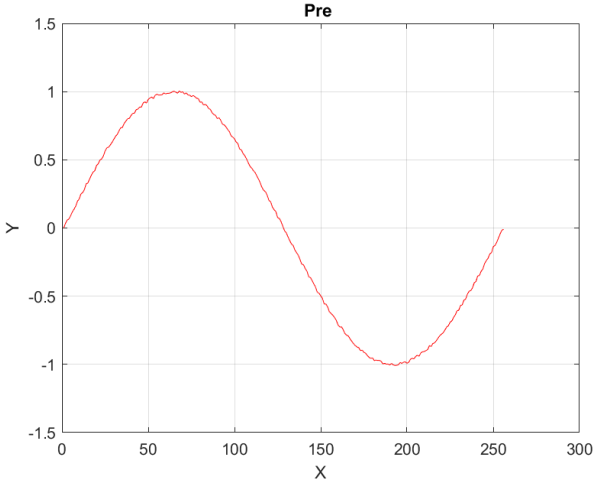

Figure 5: Figure of the unfiltered data.

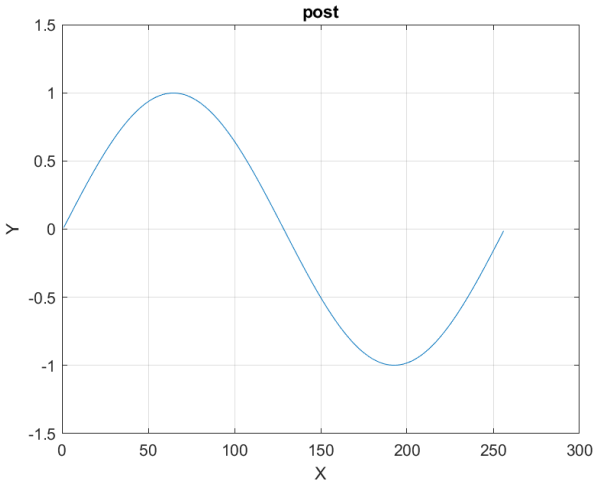

Figure 6: Figure of the filtered data.

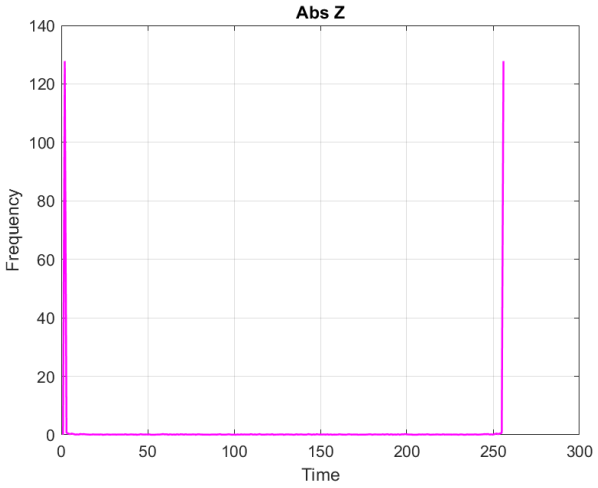

Figure 7: Figure of the absolute value of the Fourier transformed data of filtre.data. (file from lab description)

By comparing the two figures 5 and 6 the jaggedness of the unfiltered is removed and the figure comes out as a smooth curve as seen in figure 6. The value of w_{cut} which decides what frequencies should be cut or not is decided by plotting the absolute value of the Fourier transformed data from the filtre.data as seen in 7. A suitable w_{cut} is then chosen to be 10% of the maximum value of the transformed data to be sure to cut away all the low frequencies while keeping the high.

2.8 Task 4c

In task 4c a sound file is played chosen by a few different sounds in Matlab. The data of the sound is then transformed using FFT. The signal is then compressed using the following formula.

$$W(k) = \begin{cases} Y(k) & \text{if } |Y(k)| > \omega_{\mathrm{r}} \cdot \max_k |Y(k)|, \\ 0 & \text{else.} \end{cases}$$

Where $Y(k)$ is the Fourier transform of the given signal vector y, and ω_{r} is a specified relative threshold of the maximum value of Y. The sound quality was then compared before and after the compression for a few different ω_{r}

Table 3: Train sound

ω_{r}	compRatio	Quality
$\omega_{0.9}$	276 01	A single tone
$\omega_{0.8}$	193 21	Almost all sound gone.
$\omega_{0.5}$	772.84	Still recognisable but quite muffled
$\omega_{0.2}$	127.95	Almost the same as the original

Table 4: Splat sound

ω_r	compRatio	Quality
$\omega_{0.9}$	750 12	No sound
$\omega_{0.8}$	272 77	No sound
$\omega_{0.5}$	43.549	Not recognisable but sounds like a cricket
$\omega_{0.2}$	5.1939	First half is the same but the other half quite muffled

Even though the ω_r is the same, the compRatio differs from the train and splat sound. As can be seen from table (4) and (3) for example $\omega_{0.8}$, the sound splat has a much higher compRatio then for the train sound. But considering there is still sound coming from the train it must have frequencies with a higher amplitude.

A Appendix

A.1 mydft.m

```
function z = mydft(y)
    % Compute the DFT of a vector y of length N
    % z n = 1/N sum {j=0}{N-1}y j exp(-2 pi j n/N)

    % Matrix form is faster!
    N = length(y);
    j_s = 0:N-1;
    x_js = 2*pi*j_s/N;
    n_s = 0:N-1;
    omega_term = exp(-1i*x_js'*n_s)
    % An matrix of omega terms. x_js'*n_s results in NxN matrix
    z = y*omega_term/N; % (1,N)x(N,N) => (1,N)
    % Matrix multiplication with y results in a vector of size 1xN.
    % Division by N gives the desired result.
end
```

A.2 myidft.m

```
function w = myidft(z)
    %MYIDFT Inverse discrete Fourier transform.
    N = length(z);
    % Using the provided formula for the inverse DFT. (see mydft.m)
    w = N*conj(mydft(conj(z)));
end
```

A.3 myfourier.m

```
function y = myfourier(a0,a,b,x,period)
    % Calculate the values of the Fourier series given by the coefficients a0, a, b
    % at the points x.
    % a0 - is the constant term, a and b are vectors of the Fourier coefficients.
    % x - is a vector of the points at which to evaluate the Fourier series.
    % period - is the period of the Fourier series.

    % Calculate the different y values for each term in the Fourier series.
    N = length(a);
    n_s = 1:N;
    K = 2*pi/period;
    % And sum them up.
    y = a0 + sum(a.*cos(K*x'*n_s)+b.*sin(K*x'*n_s),2);
end
```

A.4 myfouriercoeff.m

```
function [a0,a,b] = myfouriercoeff(z)
    % z: complex Fourier transform

    % a0: the coefficient a0
    % a: vector with the Fourier coefficients an
    % b: vector with the Fourier coefficients bn
    N = length(z);% Compute the length of vector z
    c = z(2:N); % remove first element z(1) from z-vector
    a0 = z(1);
    rev_c = flip(c); % reverse the order of the elements in c
    % Adding the elements in c and rev_c results in adding the
    % last element of c to the first element of rev_c

    a = real(c+rev_c); % real part of the sum
    a = a(1:round(N/2)-1); % Only keep the N/2 - 1 elements
    b = real(1i*(c-rev_c));
    b = b(1:round(N/2)-1);
end
```

A.5 plotfouriercoeffs.m

```
function plotfouriercoeffs(a0,a,b,a0exact,aexact,bexact)
    % plotfouriercoeffs(a0,a,b)
    % plot the Fourier coefficients a0,a,b along with the exact values
    % a is the cosine coefficients
    % b is the sine coefficients

    % Above you calculate the discrete Fourier transform z
    % and the exact Fourier cosine coefficients a0 =a0exakt and (an) =aexact
    if length(aexact)>1
        figure()
        t_s = 1:length(a)+1;
        t_s = t_s-1; % Start t-index at 0 since we have a0
        stem(t_s,[a0 a],"*") % plot computed Fourier cosine coefficients
        hold on
        t_s = 1:length(aexact)+1;
        t_s = t_s-1; % Start t-index at 0 since we have a0
        stem(t_s,[a0exact aexact],"or")% plot exact Fourier cosine coefficients
        title("Exact and computed Fourier cosine coefficients")
        legend("Computed","Exact");
        xlabel("n"); ylabel("Fourier cosine coefficients an");
        xlim([0 lenght(a)])
        grid on;
        hold off
    end
    if length(bexact)>1
        figure()
        stem(bexact,"or")% plot exact Fourier sine coefficients
        hold on;
        stem(b,"*")% plot computed Fourier sine coefficients
        title("Exact and computed Fourier sine coefficients")
        legend("Computed","Exact");
        xlabel("n"); ylabel("Fourier sine coefficients bn");
        xlim([1 length(b)])
        hold off
    end
end
```

A.6 task2a.m

```
function [a0,a,b] = task2a()
    N = 2^8; % or 2^5.
    j = 0:N-1;
    x = 2*pi*j/N;
    y_s = 3-2*cos(15*x)+4*sin(20*x); % The specified function.
    z = mydft(y_s); % Use the function mydft to calculate the DFT of y_s.
    % Use the function myfouriercoeff to calculate the Fourier coefficients.
    [a0,a,b] = myfouriercoeff(z)
end
```

A.7 task2b.m

```
function [a0,a,b] = task2b()
    close all
    N = 2^6; % Number of points in the sequence
    j = 0:N-1;
    x = 2*pi*j/N;
    y_s = abs(cos(x)); % The function to be approximated
    z = mydft(y_s);
    % The Fourier coefficients of the function provided the DFT
    [a0,a,b] = myfouriercoeff(z);
    a0exact = 2/pi; % The exact value of a0
    n_s = 1:length(a);
    aexact = [];
    for n = n_s
        aexact(2*n) = (4/pi)*((-1)^(n+1))/((4*n^2)-1);
    end
    plotfouriercoeffs(a0,a,0,a0exact,aexact,0) % Plot the Fourier coefficients
end
```

A.8 task3a.m

```
function task3a()
    close all
    N = 2^8;
    period = 2*pi;
    x = generate_xs(N,period); % generate x values
    y = h(x);
    z = mydft(y);
    [a0,a,b] = myfouriercoeff(z);
    t = linspace(0,period,5000);
    M = [3,5,10]; % the number of terms M must be less than N/2.
    plot(t,h(t),'b-',"DisplayName",'Original function',"LineWidth",3)
    xlim([-1,7])
    hold on
    grid on
    for m = M
        % The partial sum of the Fourier series with M+1 terms
        out = myfourier(a0,a(1:m),b(1:m),t,period);
        pl = plot(t,out,"DisplayName","Fourier Series with M = " + (m)+" terms");
        legend()
    end
    xlabel('x')
    ylabel('y')
    title('Fourier Series Approximation of h(x)')
    function y = h(x)
        % Discontious function used in task 3.
        y = zeros(size(x));
        y((x>0&x<pi)) = 1;
        y((x>pi&x<2*pi)) = -1;
        y((x==0|x==pi)) = 0;
    end
end
```

A.9 task3b.m

```
function [a0,a,b] = task3b()
    % Task 3b Check overshoot at discontinuity
    close all
    N = 2^11;
    period = 2*pi;
    x = generate_xs(N,period);
    y = h(x);
    z = mydft(y);
    [a0,a,b] = myfouriercoeff(z);
    t = linspace(0,period,5000); % Values to plot
    M= [3,5,10];
    % the number of terms M must be less than N/2 where N = 2m points
    figure(1)
    plot(t,h(t),'b-',"DisplayName",'h(x)',"LineWidth",3)
    xlabel("x")
    ylabel("y")
    hold on
    grid on
    a0exact = 0;
    % Plot to check the convergence of the Fourier series
    for m = M
        % The partial sum of the Fourier series with M+1 terms
        out = myfourier(a0,a(1:m),b(1:m),t,period);
        pl = plot(t,out,"DisplayName","Fourier Series with M = "+m+" terms");
        legend("Location","northeast")
    end
    title("Fourier Series Approximation of h(x)")
    hold off

    figure(2)
    M = 1:100;
    diff = zeros(1,length(M));
    diff_exact = zeros(1,length(M));
    t = linspace(0+eps,pi-eps,5000); % Fine detailed interval for smoother plots
    a0exact = 0;
    for m = M
        % Calculate the fourier representation values using m + 1 fourier coeffients
        out_m = myfourier(a0,a(1:m),b(1:m),t,period);
        % Compute the exact fourier coefficients given the number of coeffients m to use.
        aexact = zeros(1,m);
        bexact = (2./(pi*(1:m))).*(1-(-1).^(1:m));
        out_exact = myfourier(a0exact,aexact,bexact,t,period);
        diff(m) = max_diff(out_m);
        diff_exact(m) = max_diff(out_exact);
    end
    plot(M,diff,"DisplayName","Difference with Computed Fourier Coefficients");
    hold on
    grid on
    plot(M,diff_exact,"DisplayName","Difference with Exact Fourier Coefficients");
    legend()
    xlim([0 max(M)])
    title("Difference")
    xlabel("M")
```

```
    ylabel("E(M)")
    function y = h(x)
        % Discontious function used in task 3.
        y = zeros(size(x));
        y((x>0&x<pi)) = 1;
        y((x>pi&x<2*pi)) = -1;
        y((x==0|x==pi)) = 0;
    end
    function diff = max_diff(y1)
        % Calculate the maximum difference between two vectors
        diff = max(y1-1);
        diff = max(diff,0); % Only consider overshoots,i.e. positive differences
    end
end
```

A.10 task4a.m

```
function [fast_time,my_time]= task4a()
    % Speed testing of mydft.m
    close all
    M = 1:11;
    my_time = [];
    fast_time = [];
    period = 2*pi;
    for m = M
        y = generate_xs(2^m,period); % Only to generate values.
        tic; % Begin speed testing of mydft
        z = mydft(y);
        t = toc; % Finish speed testing of mydft
        my_time = [my_time;t];

        tic;
        z = fft(y);
        t = toc;
        fast_time = [fast_time;t];
    end

    plot(M,fast_time,"r",M,my_time,"b") % Plot the results
    grid on
    legend("fft","mydft","Location","northwest")
    xlabel("m")
    ylabel("t (s)") % Time in seconds
    title("Computation time")
    hold off
end
```

A.11 task4b.m

```
clear all
close all
fid = fopen('filtre.data','r'); % open filtre.data
Y = fscanf(fid,'%f',[1 inf]); % read filtre.data
fclose(fid);
figure(1), plot(Y); % plot filtre.data
legend("Pre")
grid on
xl  = xlim; % Save limits to use the same for the filtered data
yl = ylim;
Z=fft(Y); % transform data to frequency domain
figure(2)
wcut=0.1*max(abs(Z));% cut-off
N=length(Z);
Z(abs(Z)<wcut) = 0;
Y = ifft(Z); % transform back to time domain
plot(Y,"r"); % plot filtered data in seperate figure
grid on
xlim(xl) % Resize the plot to the same size as the original for comparison
ylim(yl)
legend("Post")
hold off
```

A.12 task4a.m

```
clear all
load('splat');% load sound, e.g:  splat ,  gong  ,  handel  ,  train
sound(y,Fs); % play sound
Y = mydft(y).*length(y);% fft from Matlab
W=Y;
M=max(abs(Y));
omg_r = 0.7; % specify the relative threshold value
N = length(y);

W(abs(Y)>omg_r*M) = 0;
% compressed signal and original signal expressed as a sparse array
WS = sparse(W);
YS = sparse(Y);
% compare size before-after
before = whos('YS');
after = whos('WS');
comprRatio = before.bytes/after.bytes
% play back the sound
pause(5);% delay in order to finish playing original sound
disp('Play compressed signal');
w= real(ifft(full(WS)));
sound(w, Fs)
```