Fast Fourier Transform

Miroslav Bonchev Bonchev

Spring 2011

Scientific Computing MTH739 - Project

Queen Mary, University of London

Contents

Introduction

A periodic function is a function which repeats itself in equal distance of its parameter, i.e. $f(x) = f(x + \lambda)$, where x is a variable form the domain of the function and λ is the period on which the function repeats. By nature periodic functions can be very complicated and have many different kinds of behaviour. On the other hand the functions $\sin(x)$ and $\cos(x)$ are both periodic and well understood. The idea behind **Fourier Analysis** is to represent any periodic function as a linear combination of some fundamental building blocks. The theory uses the $\sin(x)$ and the $\cos(x)$ functions as building blocks to represent any periodic function. The Fourier analysis has two parts:

- "Analysis" which works on a given periodic function and determines the exact building blocks that constitute it in a sense of a linear combination of constituents.
- "Synthesis" on the other hand produces a function from given constituents.

Usually the domain of the given function is the time domain, but not necessary as periodicity may be in many different kinds of domains, e.g. many types of groups have periodicity. The constituents of a function are referred to as being in the frequency domain.

It is clear however that not all functions are periodic. While the Fourier analysis works on such functions by replicating (cloning/tiling) the function where it is not defined the **Fourier Transform** takes the limiting of the function as its period tends to infinity. The Fourier transform has the two similar directions:

- Forward Fourier Transformation – finding the constituent parts of a non-periodic function;
- Inverse Fourier Transformation – synthesizing the function from given constituent parts.

The time/frequency domain interpretation of the function and its constituent parts are also valid for the Fourier transform. It is important to note that a function and its constituent parts are two faces of one and the same thing. This understanding and machinery allows many useful applications. For example a noisy signal can be converted to fundamental constituent blocks and then reassembled form only part of them filtering out the undesired noise constituent.

This project presents MATLAB function implementing an algorithm for Fast (Forward) Fourier Transformation.

Fourier Analysis

Fourier Series

Suppose $f(t)$ is periodic function with period 1, so $f(t) = f(t+1)$. The idea is to try to represent $f(t)$ as a linear combination of fundamental building blocks chosen to be $\sin(x)$ and $\cos(x)$ as simplest periodic functions. A particular linear combination might look like this:

$$f(t) = a.\sin(2\pi t) + b.\sin(4\pi t) + c.\sin(6\pi t) + \cdots$$

Generalizing the question we ask: Can we express the function in the form $f(t) = \sum_{k=1}^{n} A_k \sin(2\pi kt + \varphi_k)$, and if we can what are the coefficients and the phases?

Using $\sin(2\pi kt + \varphi_k) = \sin(2\pi kt)\cos(\varphi_k) + \cos(2\pi kt)\sin(\varphi_k) = a_k \cos(2\pi kt) + b_k \sin(2\pi kt)$ and the Euler formula $e^{ix} = \cos(x) + i\sin(x)$ so we have $\cos(\mathrm{x}) = \frac{e^{ix}+e^{-ix}}{2}$ and $\sin(\mathrm{x}) = \frac{e^{ix}-e^{-ix}}{2i}$ we rewrite the sum as follows:

$$\sum_{k=1}^{n} A_k \sin(2\pi kt + \varphi_k)$$

$$= \sum_{k=1}^{n} (a_k \cos(2\pi kt) + b_k \sin(2\pi kt))$$

$$= \sum_{k=1}^{n} \frac{1}{2}\left(a_k e^{2\pi ikt} + a_k e^{-2\pi ikt} - b_k i e^{2\pi ikt} + b_k i e^{-2\pi ikt}\right)$$

$$= \sum_{k=1}^{n} \frac{1}{2}\left((a_k - ib_k)e^{2\pi ikt} + (a_k + ib_k)e^{-2\pi ikt}\right)$$

$$= \sum_{k=-n}^{n} c_k\, e^{2\pi ikt}$$

Reformulating the question we now ask again: Can we write a generic periodic function $f(t)$ with period 1 as a linear combination:

$$f(t) = \sum_{k=-n}^{n} c_k\, e^{2\pi ikt}$$

where the coefficients c_k are $c_0 = \frac{1}{2}a_0$, $c_{-n} = \frac{a_n + ib_n}{2}$, and $c_n = \frac{a_n - ib_n}{2}$, and if "YES" how do we find the coefficients?

Suppose we can, then expressing for the $m-th$ coefficient from the above formula we have:

$$c_m e^{2\pi imt} = f(t) - \sum_{\substack{k=-n \\ k\neq m}}^{n} c_k\, e^{2\pi ikt}$$

$$c_m = f(t)e^{-2\pi imt} - \sum_{\substack{k=-n \\ k\neq m}}^{n} c_k\, e^{2\pi ikt} e^{-2\pi imt}$$

$$c_m = f(t)e^{-2\pi imt} - \sum_{\substack{k=-n \\ k\neq m}}^{n} c_k\, e^{2\pi i(k-m)t}$$

Now Integrate the both parts on one period of the function and we have:

$$\int_0^1 c_m dt = \int_0^1 f(t)e^{-2\pi imt}dt - \int_0^1 \sum_{\substack{k=-n \\ k\neq m}}^{n} c_k\, e^{2\pi i(k-m)t}dt$$

$$c_m = \int_0^1 f(t)e^{-2\pi imt}dt - \sum_{\substack{k=-n \\ k\neq m}}^{n} \int_0^1 c_k e^{2\pi i(k-m)t}dt$$

Since the solution of the integral is:

$\int_0^1 c_k e^{2\pi i(k-m)t}dt = \frac{c_k}{2\pi i(k-m)}\left[e^{2\pi i(k-m)t}\right]_0^1 = \begin{cases}1 & m = k \\ 0 & m \neq k'\end{cases}$ so

We have the sum equal to zero

$$\sum_{\substack{k=-n \\ k\neq m}}^{n} \int_0^1 c_k e^{2\pi i(k-m)t}dt = 0$$

and finally

$$c_m = \int_0^1 f(t)e^{-2\pi imt}dt\,.$$

c_m is called $m-th$ Fourier coefficient. Now we know what the coefficients look like, if the function can be represented as a sum of constituent sine/cosine waves.

Now, we define **Fourier Coefficient** of a periodic function $f(t)$ to be:

$$c_k = \hat{f}(k) = \int_0^1 f(t)e^{-2\pi ikt}dt\,.$$

We pose the same question as before, namely: Can we write $f(t) = \sum_{k=-n}^{n} \hat{f}(k)e^{2\pi ikt}$?

Suppose $f(t)$ is periodic but **not** smooth function. Since $\hat{f}(k)$ is a number and $e^{2\pi ikt}$ is smooth, it follows that $\sum_{k=-n}^{n} \hat{f}(k)e^{2\pi ikt}$ is also smooth, but $f(t)$ is by assumption non-smooth so contradiction. Since finite sum of smooth functions is always smooth the above equation is not valid. We need to extend the sum from $-\infty\ to\ +\infty$ as follows:

$$f(t) = \sum_{k=-\infty}^{\infty} \hat{f}(k)e^{2\pi ikt}$$

Next, we need to show that this sequence converges to the function. For the purpose of this project the relevant proofs will be omitted, however some general remarks aiming to explain what are the Fourier coefficients will be presented below.

A function $f(t)$ is said to be square integrable if:

$$\int_0^1 |f(t)|^2 dt < \infty$$

and is denoted as $f(t) \in L^2(\mathbb{R})$, W.L.O.G. we however will work only with $f(t) \in L^2([0,1])$.

Let $f(t)$ be a periodic function with period 1 and $f(t) \in L^2([0,1])$. Then by definition the Fourier coefficients as defined previously:

$$\hat{f}(k) = \int_0^1 f(t) e^{-2\pi i k t} dt,$$

are well defined by implication from the square integrability of $f(t)$ and we have:

$$\int_0^1 \left| f(t) - \sum_{k=-n}^{n} \hat{f}(k) e^{2\pi i k t} \right| dt \to 0, as\ n \to \infty,$$

which means that the sequence converges to the function in a sense of mean-square convergence.

We now define orthogonality via inner product. Let $f, g \in L^2([0,1])$. The inner product of f and g is a defined as:

$$\langle f, g \rangle = \int_0^1 f(t)\overline{g(t)} dt,$$

then f and g are orthogonal if $\langle f, g \rangle = 0$.

The norm of f is defined as:

$$\langle f, f \rangle = \|f\|^2 = \int_0^1 f(t)\overline{f(t)} dt = \int_0^1 |f(t)|^2 dt\,.$$

Remark: $\langle f, g \rangle = 0 \Leftrightarrow \|f\|^2 + \|g\|^2 = \|f + g\|^2$ is the Pythagorean theorem for $L^2([0,1])$.

We now note that:

$$\langle e^{2\pi i m t}, e^{2\pi i n t} \rangle = \begin{cases} 1, & m = n \\ 0, & m \neq n \end{cases},$$

and so the very important observation that:

$$\left\{ e^{2\pi i k t} : -\infty < k < \infty \right\}$$

forms an orthonormal basis for the function space the elements of which are the functions that we want to be able to represent with Fourier series.

We now explain the nature of the Fourier coefficients by computing the inner product of f and $e^{2\pi i k t}$:

$$\langle f, e^{2\pi int} \rangle = \int_0^1 f(t)\overline{e^{2\pi int}}dt = \int_0^1 f(t)e^{-2\pi int}dt = \hat{f}(n)\,,$$

but we know that the inner product of two vectors in a vector space gives the projection of the vectors, so we now understand the nature of the Fourier coefficients as projections of the function onto the axes of an orthonormal basis of an infinitely dimensional vector space $L^2([0,1])$.

In conclusion Fourier series analysis is concerned with periodic functions $f(t) \in L^2(\mathbb{R})$ in two modes:

- Analysis: $\hat{f}(k) = \int_0^1 f(t)e^{-2\pi ikt}dt$ – finds the Fourier coefficients and
- Synthesis: $f(t) = \sum_{k=-\infty}^{\infty} \hat{f}(k)e^{2\pi ikt}$ – synthesizes the function from given Fourier coefficients.

Fourier Transform

As with the Fourier series there are also two types of Fourier transformations. The Forward Fourier Transform is a generalization of the Fourier series analysis for a non-periodic function as a limiting case as its period tends to infinity. The Inverse Fourier Transform is a limiting case of Fourier series synthesis for a non-periodic function as its period tends to infinity.

Suppose $f(t)$ is a periodic function with period T. Ultimately we want to have $T \to \infty$. In this case from the Fourier series for a function with period T we have:

$$c_k = \frac{1}{T}\int_0^T f(t)e^{-2\pi i\left(\frac{k}{T}\right)t}dt = \frac{1}{T}\int_{-\frac{T}{2}}^{\frac{T}{2}} f(t)e^{-2\pi i\left(\frac{k}{T}\right)t}dt$$

$$f(t) = \sum_{k=-\infty}^{\infty} c_k e^{2\pi i\left(\frac{k}{T}\right)t}$$

We want to use $T \to \infty$ to pass from periodic to non-periodic, but we cannot to just take $T \to \infty$ for suppose $f(t)$ is a non-zero, non-periodic function s.t. $f(t) = 0$ for all $t < a$ and $b < t$. Then choose T such that $-\frac{T}{2} < a$ and $b < \frac{T}{2}$, and then solve the above integral:

$$\left|\int_{-\frac{T}{2}}^{\frac{T}{2}} f(t)e^{-2\pi i\left(\frac{k}{T}\right)t}dt\right| = \left|\int_a^b f(t)e^{-2\pi i\left(\frac{k}{T}\right)t}dt\right| \le \int_a^b |f(t)|\left|e^{-2\pi i\left(\frac{k}{T}\right)t}\right|dt = \int_a^b |f(t)|dt = Const,$$

so $|c_k| \le \frac{1}{T}Const \to 0\ as\ T \to \infty$, so contradiction. If the limit is taken directly we would force all Fourier coefficients to be zero and thus be unable to represent the function via the Fourier series/transform.

The make this work we scale up by T. We introduce the modified integral for Fourier coefficients and series:

$$\mathcal{F}f\left(\frac{k}{T}\right) = \int_{-\frac{T}{2}}^{\frac{T}{2}} f(t)e^{-2\pi i\left(\frac{k}{T}\right)t}dt$$

$$f(t) = \sum_{k=-\infty}^{\infty} \frac{1}{T} \mathcal{F}f\left(\frac{k}{T}\right) e^{2\pi i\left(\frac{k}{T}\right)t}$$

Letting $T \to \infty$ makes the spectrum "arbitrarily dense", note also that k is defined on $\pm\infty$ so we replace the $\frac{k}{T}$ with a continuous variable $s \in (-\infty, +\infty)$ and the sum with an integral. Thus we reach the expressions for the Fourier transform and series for non-periodic function $f(t)$ defined for $-\infty < t < \infty$:

$$\mathcal{F}f(s) = \int_{-\infty}^{\infty} f(t) e^{-2\pi i s t} dt$$

$$f(t) = \int_{-\infty}^{\infty} \mathcal{F}f(s) e^{2\pi i s t} ds$$

The Fourier transform - $\mathcal{F}f(s)$ analyses the constituents of the function, while the inverse Fourier transform $f(t)$ synthesizes the function from given constituents. The inverse Fourier transform can be also expressed in the form: $\mathcal{F}^{-1}g(t) = \int_{-\infty}^{\infty} g(s) e^{2\pi i s t} ds$. The following expressions for forward and inverse Fourier transform hold: $\mathcal{F}\mathcal{F}^{-1}g(t) = g(t)$ and $\mathcal{F}^{-1}\mathcal{F}f(s) = f(s)$.

Remark: Generally t is interpreted as time variable while s is understood as a frequency variable.

Remark: $\mathcal{F}f(s)$ and $f(t)$ two representation of one and the same thing.

Note: that these are not proofs but only motivational notes for the problem giving a loose guidance to the results. Also note that there are further proofs required to show that the integrals converge and converge to the correct values.

Discrete Fourier Transform (DFT)

The Fourier transform above are continuous transformations, however most signals in reality are measured at discrete moments in time thus building streams of discrete data that is then passed for processing. Hence we need to say a few words about the Discrete Fourier transformation.

By sampling of a "continuous-time" function $f(t)$ we produce a sequence $f(nT)$ for integers n and some time period T. This implies that there is lost information since the samples are separated in time. In fact the highest frequency that could be recovered as constituent to the original signal is half or less than the sampling frequency. The Discrete-time Fourier transformation is defined as:

$$S_T(f) = \sum_{k=-\infty}^{\infty} S\left(f - \frac{k}{T}\right) \equiv \sum_{k=-\infty}^{\infty} T.s(nT).e^{-2\pi i f n T}, where\ s[n] = T.s(nT).$$

When $s[n]$ is periodic with period N, $S_T(f)$ simplifies to:

$$S(k) = \sum_{n=0}^{N} s[n].e^{-2\pi i\left(\frac{k}{N}\right)n}$$

For all integer values of k. This sequence is sufficient to describe N-periodic functions, and non-periodic but finite non-zero-duration N functions.

Fourier Transform Numerical Solution

Fourier transform is the process of identifying the fundamental building constituents of a signal. Using the notes above it is easy to construct a numerical solution. However a direct implementation has $O(N^2)$ cycles. Therefore for a large chunk of data this may take long computational time defeating the objectives. Also note that the error is directly dependent on the sampling rate - more samples mean better approximation. This can be viewed in a different perspective: fewer samples are equivalent to adding constituents which are not present in the original signal – this is the same as injecting frequencies, which is the same as adding noise.

Therefore the objective in general is the have as higher as possible sampling rate, but this introduces a processing bottleneck which is in general serious problem as signals are usually processed "live". One solution of the problem is the Fast Fourier Transform.

Fast Fourier Transform - Numerical Solution

The Fast Fourier Transform algorithm decomposes the original sampled data of $N = N_1 N_2$ samples to smaller N_1 and N_2 chunks recursively and applies the Fourier transformation on them. This reduces the computational time to $O(NlogN)$ cycles. The code uses the radix-2 DIT version of the Cooley-Turkey algorithm. Radix-2 DIT computes the DFTs of the even and odd-indexed samples, and then combines those two results to produce the DFT of the whole sequence. This is then applied recursively. A simplified form presented here assumes that N is a power of two – this is often not an important restriction.

Firstly the Radix-2 DIT algorithm rearranges the DFT of the function into two parts (called butterfly). A sum over the even-numbered indices $n = 2m$ and a sum over the odd-numbered indices $n = 2m + 1$:

$$X_k = \sum_{m=0}^{\frac{N}{2}-1} x_{2m} e^{-i\left(\frac{2\pi 2m}{N}\right)k} + \sum_{m=0}^{\frac{N}{2}-1} x_{2m+1} e^{-i\left(\frac{2\pi(2m+1)}{N}\right)k}$$

Taking the common factor $e^{-\frac{2\pi i}{N}k}$ out of the second sum and naming the two sums to $E(ven)$ and $O(dd)$ and we obtain:

$$X_k = \sum_{m=0}^{\frac{N}{2}-1} x_{2m} e^{-i\left(\frac{2\pi m}{N/2}\right)k} + e^{-i\frac{2\pi}{N}k} \sum_{m=0}^{\frac{N}{2}-1} x_{2m+1} e^{-i\left(\frac{2\pi m}{N/2}\right)k} = E_k + e^{-i\frac{2\pi}{N}k} O_k$$

Since the resulting DFTs have a length $N/2$ we need compute only $N/2$ outputs, due the periodicity properties of the DFT the outputs for $0 \leq k < N/2$ is identical to $N/2 \leq k < N$. So $E_{k+N/2} = E_k$ and $O_{k+N/2} = O_k$. The phase $e^{-i\frac{2\pi}{N}k}$ flips the sign of the $O_{k+N/2}$ terms. Thus we calculated whole DFT:

$$X_k = \begin{cases} E_k + e^{-i\frac{2\pi}{N}k} O_k & k < \frac{N}{2} \\ E_{k-N/2} - e^{-i\frac{2\pi}{N}(k-N/2)} O_{k-N/2} & k \geq \frac{N}{2} \end{cases}$$

The algorithm speed acceleration is due reusing the results of intermediate computations. This algorithm is highly suitable for parallel processing further significantly increasing the performance gain.

MFFT.m

```
************************************************************
function Result = MFFT( Samples )

% Make sure column vectors are treated as row vectors.
if 1 == size( Samples, 2 )
    Samples = Samples';
end

N = size( Samples, 2);

if floor( log2( N ) ) ~= log2( N )
    error( 'The number of samples must be on the powers of 2.' );
end

if 1 ~= size( Samples, 1 )
    error( 'Supplied data is not a column or row vector.' );
end

Module = 1;
Phase  = 2;

% The results are saved in a two column vector.
% The Module column contains the module of each Fourier coefficient.
% The Phase column contains the phase of each frequency for Fourier coefficient.
% [*] Result = zeros( N/2, 2 ); % [use this line to return spectrum with no duplication]
Result = zeros( N, 2 );

real = zeros( N, 1 );
imag = zeros( N, 1 );

% Rearrange the data in a butterfly order.
for n=1:((log(N)/log(2))-1)

    power=2^((log(N)/log(2))-n+1);

    for k=1:power:N
        par   = k - 1;
        nopar = k + power/2 - 1;
        last  = k + power - 1;

        for q=k:last
            if 1 == rem( q, 2)
                par = par + 1;
                real(par) = Samples( q );
            else
                nopar = nopar + 1;
                real(nopar) = Samples( q );
            end
        end
    end

    Samples = real;
end

% Do the Discrete Fourier Transformation
p=(log(N))/(log(2));

for n=1:p
    power2 = 2^(n-1);
    powerN = 2^n;

    for k=1:power2
        Wc = cos((k-1)*2*pi/powerN);
        Ws = sin((k-1)*2*pi/powerN);
```

```
        for q=k:powerN:N,
            bbreal = real(q);
            bbimag = imag(q);

            CBr =  real(q+power2)*Wc + imag(q+power2)*Ws;
            CBi = -real(q+power2)*Ws + imag(q+power2)*Wc;

            real(q) = bbreal + CBr;
            imag(q) = bbimag + CBi;

            real(q+power2) = bbreal - CBr;
            imag(q+power2) = bbimag - CBi;
        end
    end
end

% Convert the data from the Complex to the Amplitude-Phase domain.
% [*] for n=1:N/2    % [use this line to return spectrum with no duplication]
for n=1:N
    Result( n, Module ) = 2*sqrt(real(n)^2+imag(n)^2)/N;

    if 0 < real(n)
        Result( n, Phase ) = 180/pi*(pi/2+atan(imag(n)/real(n)));
    else
        Result( n, Phase ) = 180/pi*(6/4*pi+atan(imag(n)/real(n)));
    end
end

end
```

Since the theoretical part of the project documents the mechanisms behind the Fourier analysis and transform the code is fairly self-explanatory. The Function MFFT.m performs Fast Fourier Transform on a sample of real data with size on the powers of 2. The function returns a 2 column vector containing a pair for each Fourier coefficient. The first leaf of each entry (row) is the amplitude of the constituent frequency and the second leaf holds the phase for that frequency.

Note: To improve performance use the two commented lines marked as [*] to return the spectrum with no mirrored (duplicated) constituents instead of the similar line with code immediately under them.

The MFFT function structure is as follow:

1. The function ensures that the input data format and properties are compatible.

2. The data is ordered in butterfly order.

3. The Fourier coefficients are computed. Each coefficient is represented by a pair of real numbers for real and imaginary part. Another implementation could have used complex numbers. Overall there is no significant difference as the same data manipulation would have been done implicitly by the environment. In this way it is explicit for clarity and completeness. This would also allow easier use of rational numbers instead of floating for improved accuracy.

4. The complex Real-Imaginary data is converted to Amplitude-Phase representation and returned to the caller.

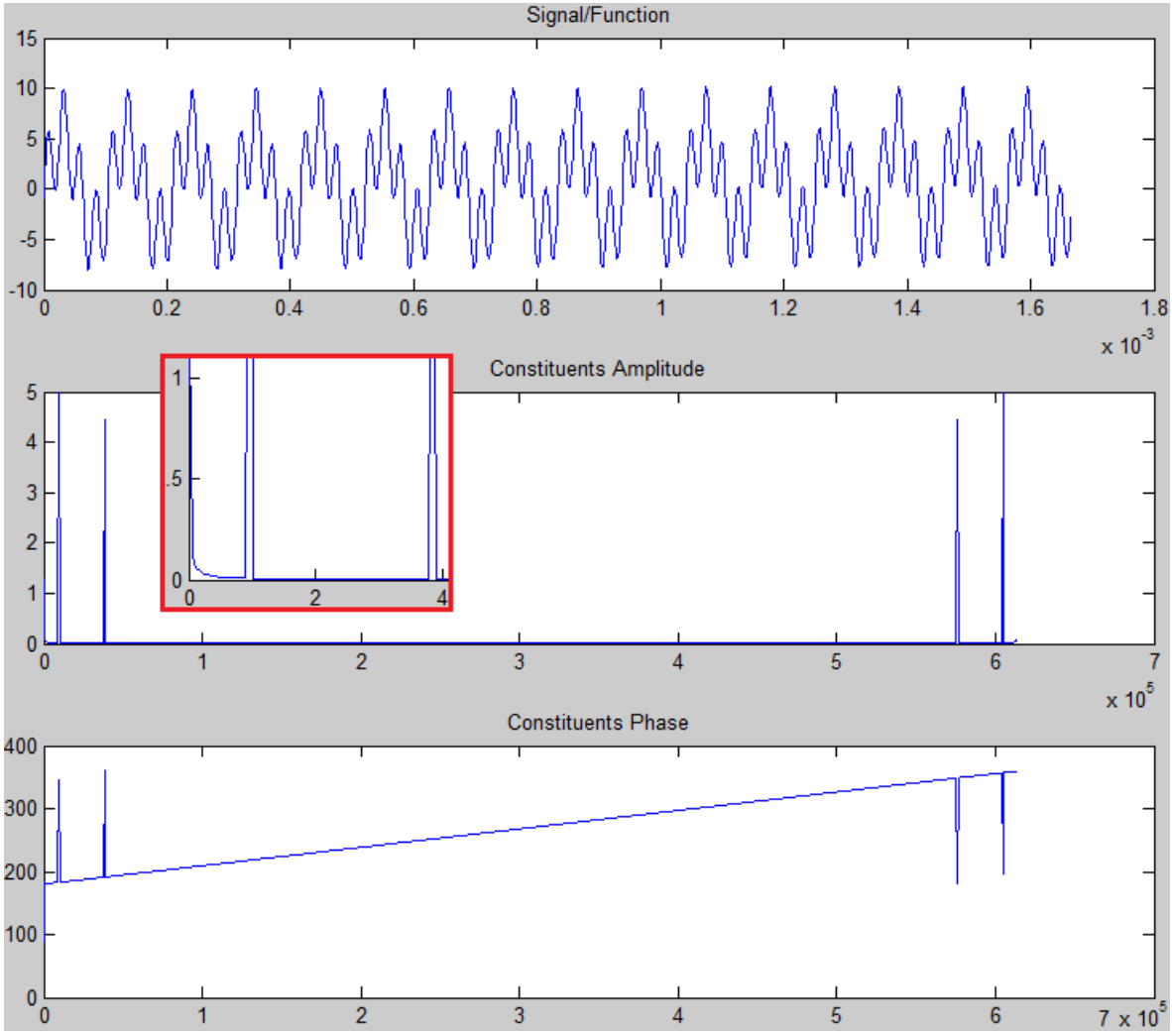
Test Cases

This section presents four testing scripts.

Sine waves test – Case A

The following script generates a signal composited from several sine waves. The signal is then passed to the Fast Fourier Function and the results are plotted.

```
clc
clear;

SamplingFrequency = 2*307200;
SamplingCount     = 4*512;
delta             = 1/SamplingFrequency;

f2 = SamplingFrequency/SamplingCount*(0:(SamplingCount-1));
t  = 0:delta:(SamplingCount-1)*delta;

x = 4.45321*sin(2*38400*pi*t) + 5*sin(2*9600*pi*t - pi/12) + sin(2*pi*50*t + pi/7);

result = MFFT( x );

subplot(3, 1, 1 ); plot( t,x), title('Signal/Function');
subplot(3, 1, 2 ); plot( f2, result(:,1)), title('Constituents Amplitude');
subplot(3, 1, 3 ); plot( f2, result(:,2)), title('Constituents Phase');
```

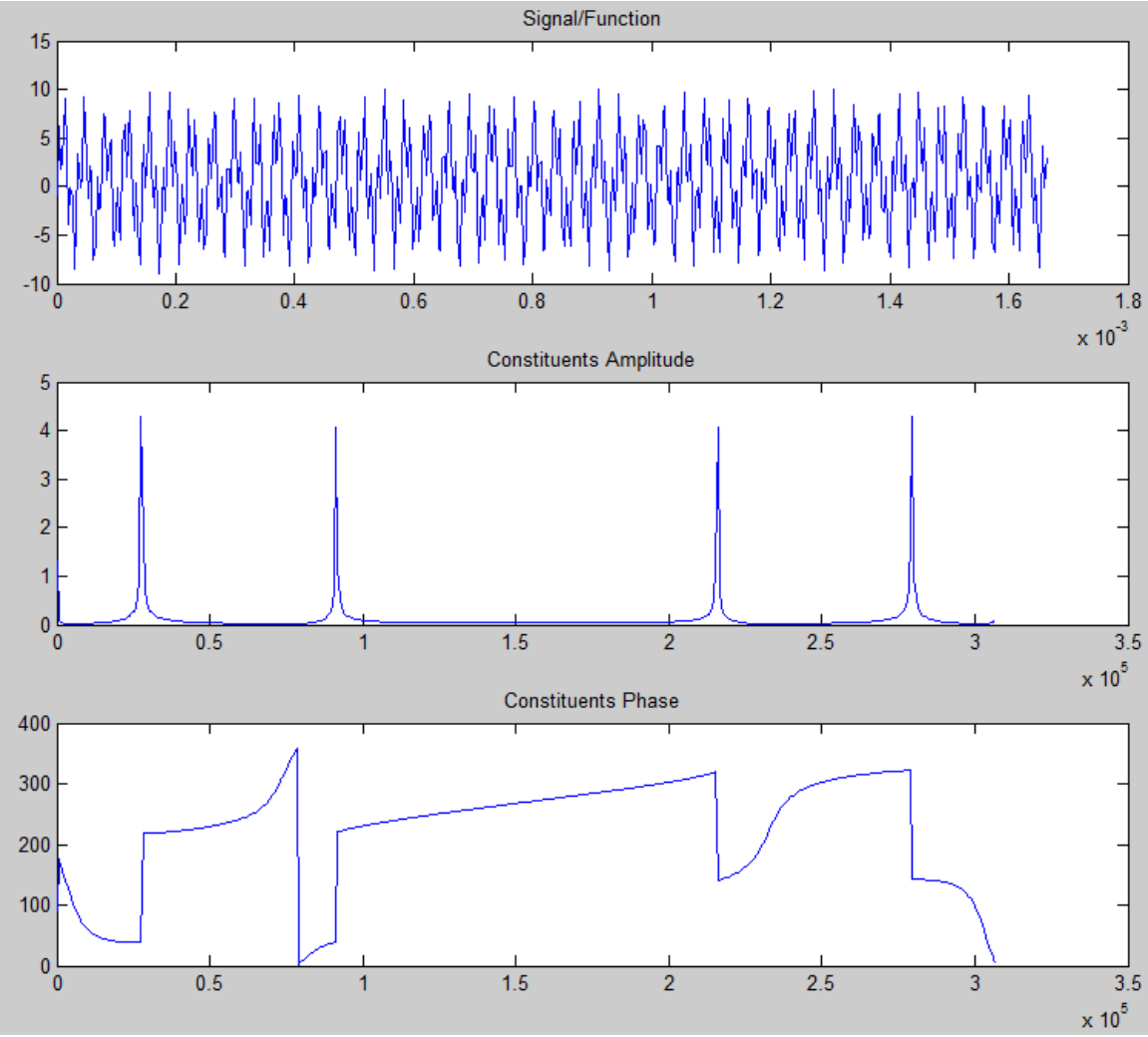


We see the amplitudes of the three constituent signals are present and have the correct values. The phases are also present and correct. The second half of the returned frequencies simply mirrors the first part and can be ignored.

Sine waves test – Case B

The following script continues the testing by changing the sampling rate, the number of samples and the frequencies of the input signal to demonstrate some of the properties of the Fourier transform.

```
clc
clear;

SamplingFrequency = 1*307200;
SamplingCount     = 1*512;
delta             = 1/SamplingFrequency;

f2 = SamplingFrequency/SamplingCount*(0:(SamplingCount-1));
t  = 0:delta:(SamplingCount-1)*delta;

x = 4.45321*sin(2*91333*pi*t) + 5*sin(2*27777*pi*t - pi/12) + sin(2*pi*50*t +
pi/7);

result = MFFT( x );

subplot(3, 1, 1 ); plot( t,x), title('Signal/Function');
subplot(3, 1, 2 ); plot( f2, result(:,1)), title('Constituents Amplitude');
subplot(3, 1, 3 ); plot( f2, result(:,2)), title('Constituents Phase');
```

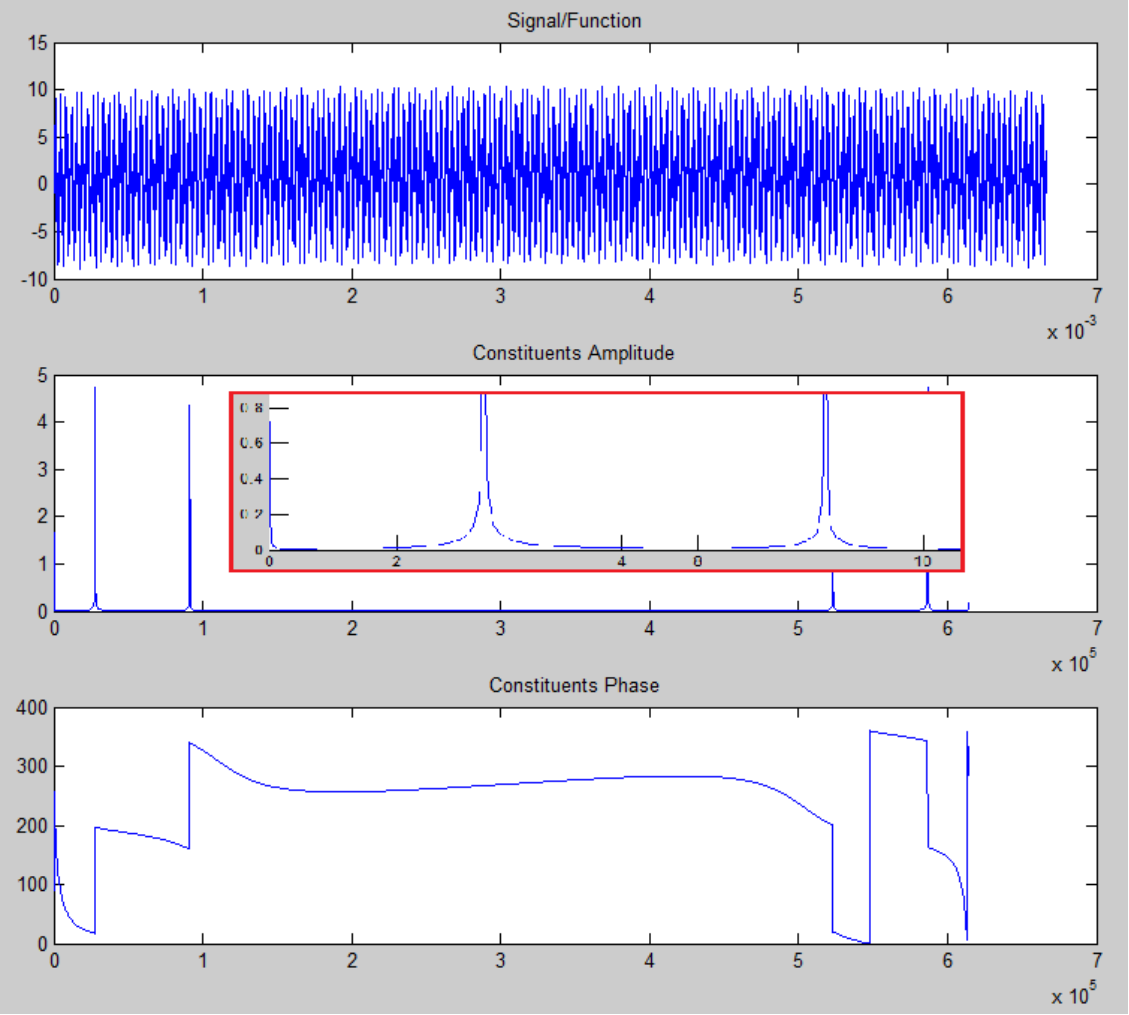


This case is similar to the previous one but notice that we have worse recognition. The amplitudes are of the spikes are smaller than what they are expected and we have many frequencies with small amplitudes which were not part of the original signal. More comments on this will be added in the discussion section.

Sine waves test – Case C

The following script continues the testing by changing only the sampling rate and number of samples to demonstrate some of the properties of the Fourier transform.

```
clc
clear;

SamplingFrequency = 2*307200;
SamplingCount     = 8*512;
delta             = 1/SamplingFrequency;

f2 = SamplingFrequency/SamplingCount*(0:(SamplingCount-1));
t  = 0:delta:(SamplingCount-1)*delta;

x = 4.45321*sin(2*91333*pi*t) + 5*sin(2*27777*pi*t - pi/12) + sin(2*pi*50*t +
pi/7);

result = MFFT( x );

subplot(3, 1, 1 ); plot( t,x), title('Signal/Function');
subplot(3, 1, 2 ); plot( f2, result(:,1)), title('Constituents Amplitude');
subplot(3, 1, 3 ); plot( f2, result(:,2)), title('Constituents Phase');
```

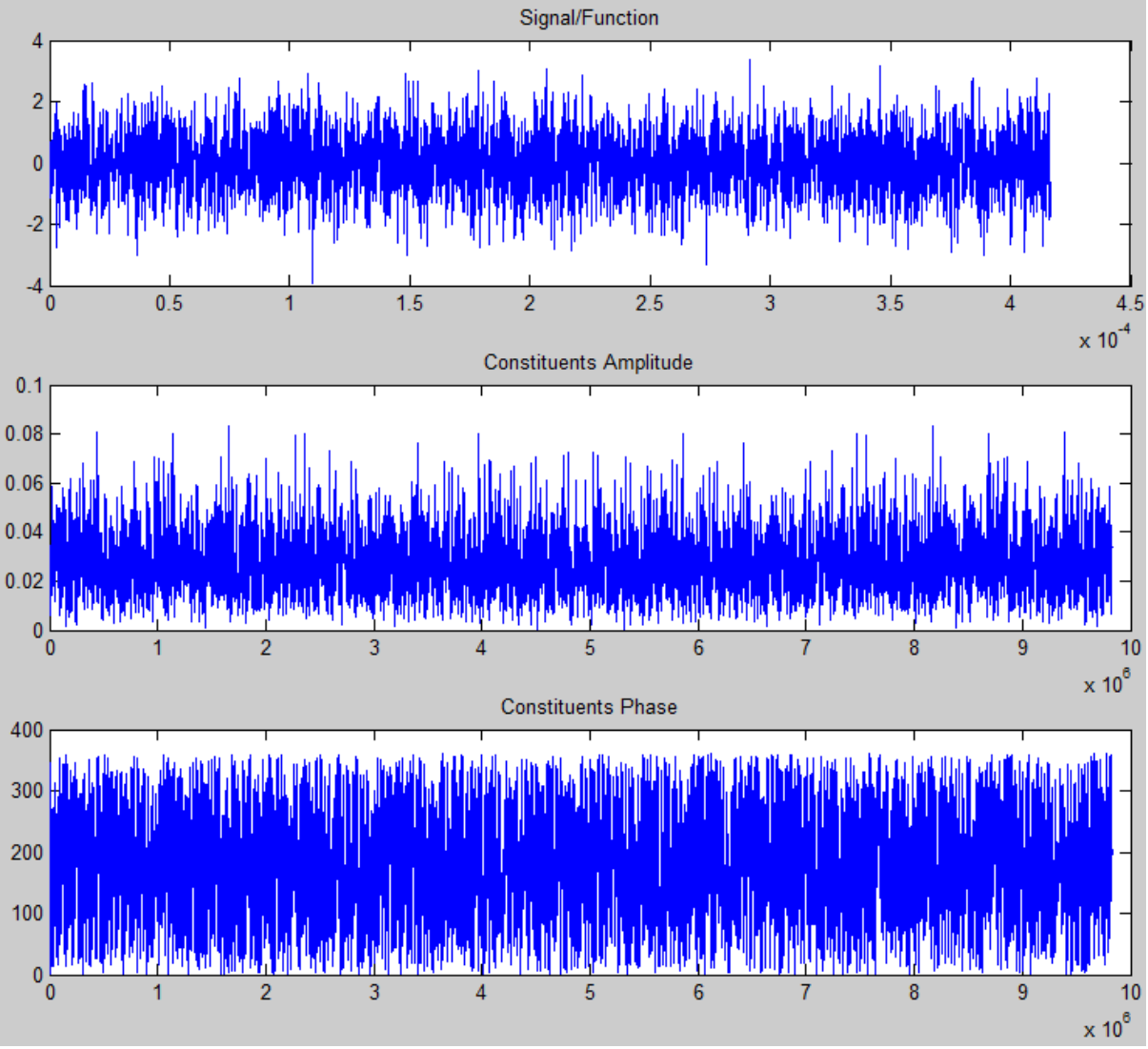


This case uses the same input signal as the previous case but utilizes doubled sampling frequency and eight times more samples which means that there are more recognizable frequencies. As a result we see the spikes are closer to the correct amplitudes and also that there are much lesser number of “injected” not present in the original signal frequencies. The amplitudes of the erroneous injected frequencies are also smaller than in the previous case.

White noise test

The following script continues the testing of the Fourier transform and algorithm implementation with white noise.

```
clc
clear;

SamplingFrequency = 32*307200;
SamplingCount     = 8*512;
delta             = 1/SamplingFrequency;

f2 = SamplingFrequency/SamplingCount*(0:(SamplingCount-1));
t  = 0:delta:(SamplingCount-1)*delta;

x = randn( size(t, 2), 1);

result = MFFT( x );

subplot(3, 1, 1 ); plot( t,x), title('Signal/Function');
subplot(3, 1, 2 ); plot( f2, result(:,1)), title('Constituents Amplitude');
subplot(3, 1, 3 ); plot( f2, result(:,2)), title('Constituents Phase');
```



The white noise is “white” because it is composited from frequencies from the entire spectrum. The graph demonstrates that this precisely is the result produced by the Fourier analysis. We see random distribution of frequencies and phases along the entire recognizable spectrum, which is what we expect.

Discussion

Clearly the Fourier transform is a very powerful tool. In this chapter we will discuss the accuracy, limitations, complexity and applications of the Fourier Transform.

Accuracy

There are multiple aspects of accuracy which will be considered here starting with the simpler and easier to deal with.

Computational Accuracy

The computational accuracy refers to the accumulative rounding errors which are building up when working with floating point numbers. Floating point numbers $\mathfrak{E}$ represent real numbers in computers. In digital computers however numbers are always discrete and so the floating point numbers are not complete. In fact the floating point numbers are imprecise, with gaps between them, where the gaps are smaller towards the zero and larger towards the ends of the interval covered by them. For this reason whenever a computation with floating point numbers is performed an error may occur rounding the number to one of the nearest numbers element of $\mathfrak{E}$. Thus when using floating point numbers the fast Fourier transform is not only faster but also more accurate than the ordinary Fourier transform because there is lesser number of computations and hence lesser number of accumulative rounding errors.

On the other hand as we will see shortly there is methodological accuracy related to the discrete Fourier analysis. It requires larger data samples for better methodological accuracy. This however would build up the computational error due the properties of the floating point numbers. The best approach to resolve the computational accumulative error problem is to not use floating point numbers at all. Instead one could use rational numbers. One good implementation of Rational numbers is available at this location: http://www.mbbsoftware.com/Software/ProperNumbersLibrary/Default.aspx. The MFFT.m fast Fourier transform function above can be easily translated to C++, and using the suggested implementation of an arbitrary precise rational numbers will completely eliminate any computational errors. It must be noted that on machines equipped with floating point coprocessor this change will introduce some small computational overhead in comparison to the floating point implementation.

Methodological Accuracy

The three sine tests above were designed to help demonstrate and understand this type of error and the possible methods to minimize or eliminate it.

In Case A, notice the sharp spikes that recognize exactly the constituents in the input signal with the correct amplitude and phase. If one looks at the numerical data they will see that there is exactly one constituent representing each of the source waves and which have largely correct amplitude and phase. The reason for this is the carefully selected frequencies. The mechanism is as follows: the sampling frequency is 307200 samples per second and the number of samples is 2048. This means that we can recognize 2048 / 2 = 1024 frequencies (the division by two is to account for the mirroring) from 0 to 153600 Hz. Now 307200/1024 = 300 so we can recognize exactly frequencies multiple of 300Hz. Now 38400/300=128, and 9600/300=32 so the 9600Hz and 38400Hz are exactly recognizable, and this is the reason that these constituents were so precisely determined. If we look at the result data these constituents have indices respectively 33 and 129 since MATLAB counts arrays from 1. Now the third constituent in the emitted wave is 50Hz, which is not exactly recognizable as 50 it is not multiple of 300, so we expect not exact recognition. Looking into the graph we see that there are many frequencies around 50Hz which we did not emit. To resolve this and have the 50Hz constituent correctly recognized we need to change the sampling frequency and number of samples taken in a way so that 50 becomes also exactly recognizable.

The frequencies in the input signal in Case B were purposefully chosen to not be exactly recognizable and thus expecting that the FFT will produce a cluster of constituents around the correct frequencies. Since the exact frequency is not recognizable the transform indeed produced these clusters of constituents whose phases are such that they cancel in a way to give best approximation. Although this is very good it is of course much better to recognize the constituents exactly. However this is not necessarily possible, as the constituents might be such that "all-recognizable" sampling rate might be two high and physically impossible with the current technology.

The frequencies in the input signal in Case C are the same as these in Case B, but if one looks at the graphs they will note that the clusters are smaller and the recognition is better. This is due the fact of using higher frequency and larger number of samples. Hence if the studied signals are unknown one may want to have highest possible sampling frequency and use the largest possible sample. These of course depend on the particular circumstances. Another solution could be interleaved operation, sampling the signal with different frequencies multiple times and choosing the best recognition.

Physical Errors

Usually signals come from some physical environment and are converted to digital values using Analogue to Digital Converter (ADC). The ADCs are generally and comparatively speaking complex and slow devices. The higher the precision the slower they become. They are limited in precision up to approximately 24 bit presently and several MHz (largely varying) sampling rates. This induces the third type of error from discretization of the input signal. Generally there is not much that can be done except using the best available equipment. In certain cases if there is a prior knowledge of the signal one could minimize the error by applying numerical techniques on the input signal before supplied to the FFT function.

Quality of Recognition

It can be shown that signals containing more constituents generally have worst recognition. This property is used in many applications using the recognition error as indicator and information source.

Limitations

The limitations are mostly the physical capabilities of the computing systems being used.

Complexity

There are varying levels of complexity. FFT is simple once understood, however Fourier analysis could be difficult to understand and particularly when lacking proper perspective.

Applications

The applications of the Fourier analysis and the Fast Fourier Transform in particular are in physics, partial differential equations, number theory, combinatorics, probability theory, statistics, option pricing, cryptography, numerical analysis, oceanography, optics, diffraction, geometry, signal processing, imaging, acoustics, recognition, detection and many other areas.

Conclusion

If one looks at the Fourier analysis from philosophical point of view he or she might conclude that it is nothing but a trivial theory since there is nothing unusual and unexpected in it. After all everything that we know about is composed from small comparatively simple building blocks. For example a star, a building, a tree or the most complicated DNA molecule are all composed from the same less than a hundred chemical elements. So why would it be so amazing that any signal is composed from some basic building blocks and that we are able to find exactly what constituents are involved using the Fourier analysis?! - Well, it is so incredible because of the enormous power that it gives us. The applications of the Fourier analysis and the Fast Fourier Transform in particular are very many and diverse. Fourier analysis is used in physics, partial differential equations, number theory, combinatorics, probability theory, statistics, option pricing, cryptography, numerical analysis, oceanography, optics, diffraction, geometry, signal processing, imaging, acoustics and other areas. While in some of these areas its presence may be moderate, in others it is extremely important and for some fields even fundamental. One such example is the field of signal processing. For example to filter a particular constituent from a signal using analogue filter is difficult, expensive and unreliable. To filter the noise using the Fourier transform however is easy, cheap and reliable for as long as the required processing power is available. If the processing resource is not available the filtering could be still done offline for undefined amount of time depending on the size of the data and the processing power of the computing system. In signal and image processing Fourier analysis is also heavily used for data compression, synthesis, many different types of recognition and many other uses. It must be noted that one of the reasons for the successful application of the Fourier analysis are the Fast Fourier Transform algorithms in forward and inverse direction which make the use of Fourier analysis possible at much lower price.

Fourier analysis is a very large subject with deep connections to many areas in science. Due the limited size and scope of this project I was only able to scratch the surface of Fourier analysis, thus proofs were omitted and applications were restricted to only a few examples.

Bibliography

1. The theoretical part of this project is mostly based on the excellent lectures given by Professor Brad Osgood at Stanford University.
2. Signals and Systems, Alan V. Oppenheim, Alan S. Willsky, Ian T. Young, Prentice-Hall, Inc., 1983
3. Theory of signals, George D. Nenov, Technique, 1990 /Bulgarian/
4. Wikipedia – Fourier Analysis article