Chapter 1

The Fourier Series

1.1 Background

The branch of mathematical understanding which we now call Fourier theory was born, like so many of the important results in modern mathematical physics, in the latter part of the 18th century. It seems first to have arisen as a problem in musical acoustics. Between about 1720 and 1740, the physicist Joseph Sauveur and the musical theorist Jean-Phillipe Rameau established the result that any sustained musical note, as for example from a bowed violin string, could be analyzed by the human ear into a fundamental and overtones; and that the frequencies of these overtones are harmonically related to the fundamental. What was not known was why this was so.

The answer was sought by mathematicians among the predictions of that enormously powerful analytical tool that Newton (and others) had invented — the calculus. The first to successfully model the vibrating string by a partial differential equation was Daniel Bernoulli. In 1748, he showed that, if the ends of a string were fixed at $x = 0$ and $x = l$, then a formal solution to its motion had the form

$$y(x,t) = \sum_n a_n \sin \left( \frac{n\pi x}{l} \right) \cos \left( \frac{n\pi ct}{l} \right)$$

(1.1)

He further asserted that this was the most general solution to the problem.

Criticism was immediate. Euler pointed out that this would mean that any function which was zero at $x = 0$ and $x = l$ must be expressible as a summation of sines. That seemed to him clearly absurd, since it would imply that all such functions must be odd and periodic: clearly not the case. The next fifty years were spent arguing back and forth about exactly what was meant by a mathematical function.

Eventually, in 1807, the answer was provided by Jean-Baptiste Joseph Fourier, just back from a disastrous tour of Egypt with his patron, Napoleon Buonaparte. In a paper on the solution of yet another differential equation, the heat conduction equation, he outlined what is now generally accepted as clear proof of the generality of Eq. 1.1. Even so, it took another hundred years for all the final details to be sorted out, and in that time it came to be realized that there were many other sets of functions in which expansions like Eq. 1.1 could be made. Hence in modern mathematics textbooks this whole branch of theory is usually dealt with as just one example of a wider range of techniques, gathered together under the general heading of Sturm-Liouville Theory.

But to physicists, the original form of Fourier theory, and its extension into the Fourier transform, retains its key importance. Most physical devices which receive energy from oscillating waves — the ear, the eye, acoustic transducers, radio antennas — usually respond sinusoidally. That is why it is important for a student of physics to develop a keen intuitive understanding of, by getting lots of practice with, Fourier analysis and synthesis.
1.2 The Fourier Theorem

A simple statement of the theorem is as follows.

*Any physical function that varies periodically with time with a frequency \( f \) can be expressed as a superposition of sinusoidal components of frequencies:*

\[ f, 2f, 3f, 4f, \ldots \text{ etc} \]

A quantitative statement of the same theorem is usually given in reverse form:

*If a periodic function of \( t \), with period \( \tau \), can be expressed as the following summation*

\[
f(t) = \frac{a_0}{2} + \sum_{n=1}^{\infty} a_n \cos(2\pi nt/\tau) + \sum_{n=1}^{\infty} b_n \sin(2\pi nt/\tau) \tag{1.2}\]

*then the coefficients in that summation may be calculated as follows:*

\[
a_n = \frac{2}{\tau} \int_0^\tau f(t) \cos(2\pi nt/\tau) \, dt \tag{1.3}
\]

\[
b_n = \frac{2}{\tau} \int_0^\tau f(t) \sin(2\pi nt/\tau) \, dt \tag{1.4}
\]

The mathematical investigation of this theorem then hinges on establishing what conditions need to be placed on the function \( f(t) \) for this assumption to be valid. A lot of that investigation is irrelevant in physics. Most every physically realizable periodic function does satisfy this theorem.

An alternative statement which is easily derivable from the first, and which is often more useful, is the following:

*If a periodic function of \( t \), with period \( \tau \), can be expressed as the following summation*

\[
f(t) = \frac{c_0}{2} + \sum_{n=1}^{\infty} c_n \cos(2\pi nt/\tau - \phi_n) \tag{1.5}\]

*then the (positive) coefficients \( c_n \) and the phases \( \phi_n \) may be calculated from the coefficients \( a_n \) and \( b_n \) defined by Eqs. 1.3 and 1.4 thus:*

\[
\begin{align*}
c_n &= \sqrt{a_n^2 + b_n^2} \\
\phi_n &= \arctan \left( \frac{b_n}{a_n} \right)
\end{align*} \tag{1.6}
\]

By convention, the quantities appearing in these statements of the theorem are referred to by special names, as follows.

- The sinusoidal functions which are added in Eq. 1.2 are known as **Fourier components**.
- The Fourier component with the same period as the original function is known as the **fundamental**. Those with higher frequencies are called **harmonics**.
- The coefficients \( a_n \) and \( b_n \), or \( c_n \) and \( \phi_n \), are known as **Fourier coefficients**.
- The complete list of the coefficients \( c_n \), with the phases \( \phi_n \), is called the **Fourier spectrum**.
- In physical applications, it is occasionally useful to talk about the energy which is carried by each sinusoidal component, and is proportional to its square. The complete list of the squares of the coefficients, \((c_n)^2\), appearing in Eq. 1.5, is called the **power spectrum**.
1.3 Mathematical concerns

The fact that Eqs. 1.3 and 1.4 are consistent with Eq. 1.2 relies on the following integral properties of the trigonometric functions (which are not difficult to prove).

\[
\int_0^{2\pi} \sin^2 nx \, dx = \pi \\
\int_0^{2\pi} \cos^2 nx \, dx = \pi 
\] (1.7)

\[
\int_0^{2\pi} \sin nx \sin mx \, dx = 0 \quad \text{for} \ m \neq n \\
\int_0^{2\pi} \cos nx \cos mx \, dx = 0 \quad \text{for} \ m \neq n \\
\int_0^{2\pi} \sin nx \cos mx \, dx = 0 \quad \text{for all} \ m, n
\] (1.8)

It is often useful to consider these expressions as equivalent to the relation between two vectors which are orthogonal to one another. This is expressed in the equation

\[
a \cdot b = 0 
\] (1.9)

The most intuitive way to appreciate this is to realize that, when these calculations are done in a computer, any function is in fact represented by an array of numbers, and each of these numbers might just as easily be thought of as the component of a vector along the different axes in a many-dimensional space. The integrations done in Eqs. 1.7 or 1.8 are calculated as a summation of terms, each of which consists of one “component” of a sin or cos “vector” by the corresponding “component” of another “vector”. So, in a formal sense, Eqs. 1.7 and 1.8 are exactly the same sort of result as Eq. 1.9. (The branch of mathematics which deals with this is called vector space theory.) Eqs. 1.7 and 1.8 are therefore known as orthogonality relations.

If you multiply Eq. 1.2 by \(\cos(2\pi nt/\tau)\), or by \(\sin(2\pi nt/\tau)\), and integrate from 0 to \(\tau\), it is very easy to establish that Eqs. 1.3 and 1.4 are valid. On the other hand, if you substitute Eqs. 1.3 and 1.4 into 1.2, then you find the result

\[
f(t) = \frac{2}{\tau} \int_0^\tau dt' f(t') \left\{ \frac{1}{2} + \sum_{n=1}^{\infty} \cos \left( \frac{2\pi n(t-t')}{\tau} \right) \right\} 
\] (1.10)

which leads to the conclusion that

\[
\left\{ \frac{1}{2} + \sum_{n=1}^{\infty} \cos \left( \frac{2\pi n(t-t')}{\tau} \right) \right\} = \frac{\tau}{2} \delta(t-t') \quad \text{for} \ (t-t') \leq \tau. 
\] (1.11)

(A definition of the impulse symbol, alternatively called the Dirac delta function, can be found in most textbooks on Quantum Mechanics.) Strictly speaking, the right hand side of this equation is a periodic collection of delta functions: but only one occurs inside the interval \(0 \leq t < \tau\). Eq. 1.11 is called the closure relation, and it is this property of the trigonometric functions that underlies the whole of Fourier Theory.

As mentioned earlier, such a property is not restricted to trigonometric functions. There are many sets of functions — Bessel functions, Hermite functions, etc. — which obey a similar expression. In each case members of the set are identified by satisfying the same boundary conditions. In the present example all the Fourier components are periodic, of period \(\tau\). In the case of the Bessel functions, for example, all the members of the set go to zero at \(r = 0\) and \(r = \infty\).

This property we are talking about here means that any function which has the same boundary conditions as the functions of the set can be expressed as a linear combination of members of that set. The property is called completeness and the set is said to be complete.
1.4 Structure of the program

The program that accompanies this chapter is written in Pascal, in the development environment Borland Delphi. It has all the user interface in place but several of the critical procedures are not yet written. These are left as exercises, so that you will become familiar with the important items in the program, having written them yourself. When you have finished all the exercises the program should run in the following manner.

Running the completed program

When you run the program, you are presented with buttons which direct your choice of action. The may choose sequences of buttons as follows.

Fourier Analysis This puts up a form which displays the default version of the function that you are going to Fourier analyse — referred to hereafter as the primary function. The default primary function is a simple sine wave.

Choose Function This allows you to change the primary function to one of several pre-determined shapes. The choices are: the default sine wave, the result of the latest synthesis process if you have already used that part of the program, a sawtooth function and a square pulse. (Note that at the start of the exercise, the necessary code for the last two choices has not yet been written.)

Some of these shapes require you to enter one parameter for their complete specification (for example the square pulse needs to have specified the fraction of the cycle for which it has its maximum value). This parameter can be entered into the box showing.

Find one component This allows you to enter a positive integer (n) to specify the number of the Fourier component you want to calculate. When you have done this, the functions \( f(x) \times \cos nx \) and \( f(x) \times \sin nx \) are plotted. They are integrated and the value of the integral (the area under each curve) is displayed. A vertical bar, whose height is represents the amplitude of the corresponding Fourier coefficient, and another representing its phase, are added to the spectrum which appears in the lower window.

Find all components This will carry out this procedure for all Fourier components, up to a specified maximum number, and display the complete spectrum.

Fourier Synthesis This puts up a form which displays the default version of the spectrum from which you will synthesize a periodic function. It is referred to as the given spectrum. The default given spectrum has constant amplitude and zero phase for all components.

Choose Spectrum This allows you to change the given spectrum to one of several pre-determined spectrums. The choices are: the default constant amplitude, the result of the latest analysis process if you have already used that part of the program, and two others initially unspecified. (Again, the necessary code for the last two choices has not previously been written.)

Again also, the spectrums may require you to enter a parameter for their complete specification. This parameter can be entered into the box showing.

Add one component This allows you to enter a positive integer to calculate the Fourier component corresponding to that number (with the appropriate amplitude and phase) and draw it. It is also added to a resultant synthesized function, drawn in the uppermost window. Note that the program guards against the same component being added to the resultant more than once. When each has been added in, the corresponding bar in the spectrum is coloured gray, and calling that number subsequently will not result in any change to the synthesized function.

Add all components This will carry out this procedure for all Fourier components, up to a specified maximum number, and display the final synthesized function.
Programming features

Any Pascal/Delphi program is made up of several units. Each unit contains code to do the various actions that have to be performed while the program is running. All the code relating to the physics calculations you want to do is isolated into one unit, called \texttt{FThm\_Phys}. A skeleton of this unit is laid out at the end of this chapter (see section 1.9). You will need to be aware of the following program specifics in order to do the following exercises easily.

- The period of the oscillation is represented by a global variable called \texttt{tau} of type \texttt{double}. This is set in the \texttt{CONST} list to be 1 second.

  A global variable called \texttt{t} is used to represent the time. It is an array of numbers (which is given the type name of \texttt{DataVector}). The number of elements in this array is \texttt{NUMDATA+1}. That means that the first value, \texttt{t[1]}, is zero, and the last value, \texttt{t[NUMDATA+1]}, is equal to \texttt{tau}. The constant value of \texttt{NUMDATA} is set in the \texttt{CONST} list to be 200. The time separation between grid points on the time axis is called \texttt{deltaT}.

  All these values are calculated in a procedure called \texttt{SetUpTheTimeVariable}.

- Another global variable, also of the type \texttt{DataVector}, is used to represent the primary function that the program is currently analyzing into its components, and is given the name \texttt{primaryFunc}. A second global \texttt{DataVector} called \texttt{synthFunc} is used to represent the function that the program is currently synthesizing from a previously established spectrum.

  In some of the exercises you will be asked to use a particular primary function which the program does not already calculate. In that case you will have to write your own procedure to calculate the values which make up the variable \texttt{primaryFunc}. There are two empty procedures where you can put this code: \texttt{SetUpFunctionNumberThree} and \texttt{SetUpFunctionNumberFour}.

- Spectrums are stored as an array of two-variable records, representing the amplitude and phase of the individual coefficients. For convenience, such double arrays are given the type name of \texttt{Spectrum}. Note that, contrary to usual Pascal programming custom, these arrays start from index \texttt{i=0} rather than \texttt{i=1}. This is for convenience, because there is a zeroth harmonic in a normal Fourier spectrum.

  Individual items contained in this array are accessed as follows: \texttt{thisSpectrum[n].amp} is the amplitude of the \texttt{n}th harmonic component of the spectrum of that name, and the phase of that harmonic is \texttt{thisSpectrum[n].phase}.

  A global variables of this type, \texttt{givenSpectrum}, is used by the program to represent the spectrum from which the function \texttt{synthFunc} is currently synthesized. Another, \texttt{analysSpectrum}, is used to represent the spectrum which is being constructed by analysing the function \texttt{primaryFunc}.

  In some of the later exercises you may want to use a spectrum which has not been constructed by analyzing some known function. In that case you will have to write your own procedures to calculate the appropriate values for the variable \texttt{givenSpectrum}. There are two empty procedures where you can put this code: \texttt{SetUpSpectrumNumberThree} and \texttt{SetUpSpectrumNumberFour}.

- A global constant called \texttt{highestHarm} represents the maximum number of harmonics the program will handle. In the initial program, this is set to 50. If you want to change it you may do so in the \texttt{CONST} list at the top of the unit.

- Default values for the variables \texttt{primaryFunc} and \texttt{givenSpectrum} are set by the pair of procedures \texttt{SetUpDefaultPrimaryFunction} and \texttt{SetUpDefaultGivenSpectrum}. The default function to be analyzed is set to be a simple sine wave, and the default spectrum to be used in synthesis has all harmonics with equal amplitudes and zero phase. These values are chosen specifically for the first set of exercises. As you work with the program you may want to change them to defaults which are of more use to you.
1.5 Theoretical exercises

These exercises MUST be done outside the microlab, before you start work on the programming or computational exercises 1.6, 7. One of the tutors will ask to see your solutions before you begin.

1.1 Spectrums of simple analytic shapes

Using purely analytical methods, consider the following analytic shapes, which are assumed to repeat themselves after the period $\tau$, and demonstrate that their Fourier components are as specified.

(a) Consider the delta function, $\delta(t)$:

(i) By putting Eq. 1.11 into the same form as Eq. 1.5, show that the delta function $\delta(t)$ can be expressed as a sum of Fourier components which are given by

$$\begin{align*}
  c_n &= \frac{2}{\tau} \\
  \phi_n &= 0
\end{align*}$$  \hfill (1.12)

(ii) Similarly, in the more general case, show that the delta function centred on time other than zero, $\delta(t-t')$, can also be expressed as a sum of Fourier components. Write down an expression for these components similar to Eq. 1.12.

(b) Consider the sawtooth function:

$$f(t) = \begin{cases} 
  t & \text{for } 0 \leq t \leq \tau. 
\end{cases}$$ \hfill (1.13)

(i) By using Eq. 1.3 and Eq. 1.4 directly, show that zeroth Fourier component is:

$$a_0 = \tau$$

while the non-zero components are given by:

$$\begin{align*}
  a_n &= 0 \\
  b_n &= -\frac{\tau}{\pi n}
\end{align*}$$ \hfill (1.14)

(ii) Write down expressions for the corresponding amplitudes $c_n$ and phases $\phi_n$.

(c) Consider the hat function:

$$f(t) = \begin{cases} 
  1 & \text{for } 0 < t < \alpha \tau \\
  0 & \text{for } \alpha \tau \leq t \leq \tau
\end{cases}$$ \hfill (1.15)

(i) Again, by using Eq. 1.3 and Eq. 1.4 directly, show that the Fourier components are

$$\begin{align*}
  a_n &= \frac{\sin 2\pi n\alpha}{\pi n} \\
  b_n &= \frac{1 - \cos 2\pi n\alpha}{\pi n}
\end{align*}$$ \hfill (1.16)

(ii) For the special cases $\alpha = 1/2, 1/3, 1/4$, some of the Fourier components have zero amplitude $c_n$. Which components are these?

1.6 Programming Exercises

This first set of exercises asks you to complete the program by writing the key procedures that are not already included. It is wisest to do them in the following order, checking as you write each that they are working properly.

Please make sure you have read the preceding section “Programming features” before doing these exercises.
1.2 Synthesis procedures

The first job is to write the procedures which add one component at a time to build up the synthesized function as in Eq. 1.5. These procedures have the following headings:

- ConstructOneComponent(n:Integer; givenSpectrum:Spectrum;
var component:DataVector);
- AddComponentToSynthesizedFunction (component:DataVector;
var synthFunc:Datavector);

(a) Write the code for the first procedure, which calculates one of the Fourier components that are summed in Eq. 1.5. Note that, because of the way the variables being passed to the procedure are specified, you will use the variable \( n \) for the number of the harmonic, and the variable \( component \) for the dataVector which represents the Fourier component you are calculating. In order to do the calculation, you will need to extract the appropriate amplitude and phase from the double array \( givenSpectrum \), as described in the third point on page 5.

Don’t forget that the zeroth component must be calculated differently from the others.

(b) Test the code you have written immediately by running the program. Work with the default spectrum that comes up when you choose [Fourier Synthesize]. When you enter a positive integer for the harmonic number the appropriate Fourier component will be drawn. This should be \( \cos(2\pi nt/\tau) \) since all amplitudes are 1 and all phases are zero. The zeroth component will, of course, be different.

Don’t proceed until you are absolutely sure this procedure is working properly.

(c) Now write the second procedure mentioned above. Be clear that you are NOT being asked to sum all the components together. You are simply being asked to add the one component function you just calculated onto another function, called \( synthFunc \), which is being constructed one harmonic at a time. This exercise requires you to write no more than one or two lines of code.

(d) Test the code you have written immediately by running the program as before. This time when you enter successive harmonic numbers you should see the synthesized function being built up.

Observe one feature already built into the program. Once a component has been calculated and added to the synthesized function, it cannot be added in again. The reason for this feature should be obvious.

When you are absolutely sure you have both these procedures working properly tackle the following exercises.

(e) When you add all 50 components together you should have demonstrated the truth of the closure relation Eq. 1.11. How accurately did you do this? What was (effectively) the value of \( t' \) for which you have proved the closure relation? (Refer to your answer to Exercise 1.1a(i)).

(f) Increase the number of harmonics the program will handle, by changing the constant \( highestHarm \). How much does this increase the accuracy of the result?

(g) Change the procedure which sets the default spectrum so that, when you synthesize a function from this spectrum, it demonstrates the truth of Eq. 1.11 for arbitrary values of \( t' \). This involves rewriting the procedure SetUpDefaultGivenSpectrum, so that the phases are no longer all zero, but have the values you calculated in Exercise 1.1a(i).

Hint: You will observe that the procedure SetUpDefaultGivenSpectrum is passed a single number, called \( parameter \), when it is called. This number can be read in from the screen when the button [Choose Spectrum] has been selected. This variable is not used in the procedure as it was originally written. However you should use it to represent the ratio \( t'/\tau \) in the code that you write.

When you have completed all these exercises successfully you can be fairly confident that the part of your program which handles Fourier synthesis is working properly, and you can move on to the next exercises.
1.3 Analysis procedures

The second job is to write the procedures to calculate the Fourier coefficients of the function you would like to analyse. To do this you have to multiply the function by a cos function (or a sin function) and then integrate the product over the whole period. You will need to write the code for two Pascal Functions and one Procedure, whose names should be self-explanatory:

\[
\text{CosCoefficient}(n: \text{Integer}; \ primaryFunc: \text{Datavector}; \ var \ cosProduct: \text{DataVector}): \text{Double};
\]

\[
\text{SinCoefficient}(n: \text{Integer}; \ primaryFunc: \text{Datavector}; \ var \ sinProduct: \text{DataVector}): \text{Double};
\]

\[
\text{ConvertCoefficients}(an, bn: \text{Double}; \ var \ cn, phin: \text{Double});
\]

(a) Write the code for the first of these, to calculate the \(n\)th cosine coefficient \(a_n\) according to Eq. 1.3. You are required to multiply the primary function by the appropriate cosine function and store the product, which is the integrand of the integration you have to do, as \text{cosproduct}. Next carry out the integration — i.e. find the area under the function \text{cosproduct}. You can use any method of numerical integration you choose, although it is probably accurate enough simply to divide the area up into vertical rectangles and add their areas. Don’t forget to multiply by the factor outside the integral sign before you finish.

As is the syntax with a Pascal Function, the last statement must be

\[
\text{CosCoefficient} := \text{(the result of your calculation)};
\]

(b) Run the program immediately to test that your code is correct. Choose \text{Fourier Analyse} and work with the default primary spectrum which is equal to \(\sin(2\pi t/\tau)\). No matter what harmonic number you input, you should get zero as the result of the integration — which is what you expect from Eq. 1.8. If you do not get exactly zero, can you account for the discrepancy in terms of your integration algorithm? You could, for example, increase the number of integration steps (\text{NUMDATA}) and see what difference that makes.

(c) Now write the code for the second Pascal Function, to calculate the \(n\)th sine coefficient \(b_n\) according to Eq. 1.4. Run the program immediately to test that your code is correct. This time you should find that the results of all integrations are zero, except one, in accordance with Eqs. 1.7 and 1.8.

When you are absolutely sure your code is giving you the right answers proceed to the next exercise.

(d) Write the third of the above procedures, to calculate the amplitude \(c_n\) and phase \(\phi_n\), using previously calculated values of \(an\) and \(bn\), using Eq. 1.6. To fit in with the way the procedure links to the rest of the program, use the variables \(an\) and \(bn\) to represent \(a_n\) and \(b_n\), and \(cn\) and \(phin\) to represent \(c_n\) and \(\phi_n\).

Calculation of \(cn\) is straightforward, but calculation of \(phin\) is where you have to be careful. The standard Pascal Function \text{ArcTan} returns a value for the angle which lies between \(-\pi/2\) and \(+\pi/2\). Your code will have to make sure that the phase you calculate is in the right quadrant (between 0 and \(2\pi\)) depending on the signs of \(an\) and \(bn\). You also need to make explicit allowance for the fact that \(an\) and/or \(bn\) might be zero.

(e) Run the program and convince yourself that your code calculates amplitudes and phases correctly for the default primary function that is there.

(f) Change the procedure which sets the default function, \text{SetUpDefaultPrimaryFunction}, so that it is equal to a few of \(\cos(2\pi nt/\tau)\) and \(\sin(2\pi nt/\tau)\). Check carefully that your code still calculates amplitudes and phases correctly each time. Again you might like to use the passed variable, \text{parameter}, to set different sine (or cosine) functions as the default.

\text{Note}: when the amplitudes of the Fourier components are represented graphically by the program, they are plotted on a logarithmic scale. What is actually plotted is \(\ln(c/c_{\text{min}})\), where \(c_{\text{min}}\) is the smallest amplitude we will bother about. In the program it is called \text{lowestAmp}, and it is set, in the \text{CONST} list, to be 0.0001. You may, of course, change it to anything you want.
1.7 Computational exercises

1.4 Spectrums of simple computed shapes

(a) Sawtooth function

(i) Write a procedure to set up as the function to be Fourier analyzed the sawtooth function described in Exercise 1.1(b). Note that there is an empty procedure called SetUpFunctionNumberThree in the unit FThm_Phys. This is where you should put the code you write. You won’t be able to change the name of this procedure without finding where it is called in other parts of the program. That is probably not worth the trouble.

(ii) Compute its Fourier spectrum, and compare with the predictions of Exercise 1.1b(i). Note: You should be able to use the “graph inspector” that comes with Delphi/SciTools to show that the amplitudes are exactly as predicted by Eq. 1.14.

(b) Square wave

(i) Write a procedure to set up as the function to be Fourier analyzed the square wave function described in Exercise 1.1c.

(ii) It is important to write this in such a way that the variable $\alpha$ (which must be less than 1) can be changed during the running of the program. As before, you can use the passed variable parameter and know that you will be able to enter its value at runtime.

(iii) When you have written this code, compute the Fourier spectrum of the function you have calculated. Compare the results you get with the calculations you did in Exercise 1.1c(i).

(iv) You should observe that, for $\alpha = 1/2, 1/3, 2/3, 1/4, \ldots$, certain of the Fourier coefficients vanish completely. Explain in words why this occurs.

(c) Triangular wave

(i) Write a procedure to set up as the function to be Fourier analyzed the triangular wave function described by

\[
\begin{align*}
  f(t) &= \frac{t}{\alpha} \quad \text{for} \quad 0 \leq t \leq \alpha \tau \\
  &= \frac{\tau - t}{1 - \alpha} \quad \text{for} \quad \alpha \tau \leq t < \tau.
\end{align*}
\]

Again make sure you can vary the quantity $\alpha$ while the program is running.

(ii) Compute its Fourier spectrum, and note the similarities which exist with the spectrum calculated in Exercise 1.4b above. Explain why this similarity occurs.

(iii) You should observe that, for $\alpha = 1/2, 1/3, 2/3, 1/4, \ldots$, exactly the same Fourier coefficients vanish as vanished for the square wave. Explain in words why this would be expected.

1.8 Bibliography

1.9 Program layout

unit FThm_Phys;

interface

uses

....

type

....

var

....

CONST

NUMDATA = 200;  // Number of points in arrays
highestHarm = 50;  // Max number of Fourier components calculated

highestAmp : Double = 1.0000;  // Limits for plotting the spectrum
lowestAmp : Double = 0.0001;

tau : Double = 1;  // period of the primary function

TYPE

DataVector = ARRAY[1..NUMDATA+1] OF Double;
VectorArray = ARRAY[1..2] OF DataVector;
TwoVector = ARRAY[1..2] OF Double;
AmpPhase = RECORD
  amp: Double;
  phase: Double;
END;
Spectrum = ARRAY[0..highestHarm] OF AmpPhase;

VAR

t : DataVector;  // the time variable
deltaT : Double;

primaryFunc : DataVector;
synthFunc : DataVector;
component : DataVector;
cosProduct : DataVector;
sinProduct : DataVector;
analysSpectrum : Spectrum;
givenSpectrum : Spectrum;

procedure ....

{list of all procedures and functions used externally}

// ****************************************************** //

implementation

{$R *.DFM}
// ******** SETTING UP PROCEDURES ******** //

procedure SetUpTheTimeVariable;
var
 i : Integer;
begin
 highestT := tau;
 lowestT := 0;
 deltaT := (highestT - lowestT) / NUMDATA;
 for i:=1 to NUMDATA + 1 do
  t[i] := (i - 1) * deltaT;
end;

procedure InitializeJob;
begin
 SetUpTheTimeVariable;
 SetUpDefaultPrimaryFunction(1);
 SetUpDefaultGivenSpectrum(1);
end;

// ******** PROCEDURES FOR SYNTHESIS ******** //

procedure ConstructOneComponent(n:Integer;
       givenSpectrum:Spectrum;
       var component:DataVector);
begin
end;

procedure AddComponentToSynthesizedFunction
       (component:DataVector;
       var synthFunc:Datavector);
begin
end;

// ******** PROCEDURES FOR ANALYSIS ******** //

function CosCoefficient(n:Integer;
       primaryFunc:Datavector;
       var cosProduct:DataVector):Double;
begin
end;

function SinCoefficient(n:Integer;
       primaryFunc:Datavector;
       var sinProduct:DataVector):Double;
begin
end;

procedure ConvertCoefficients(an,bn:Double;
       var cn,phin:Double);
begin
end;
// ******** PROCEDURES FOR SPECIFYING SPECTRUMS ******** //

procedure SetUpDefaultGivenSpectrum(parameter: Double);
var
  n: Integer;
begin
  for n:=0 to highestHarm do
    begin
      givenSpectrum[n].amp := 2/tau;
      givenSpectrum[n].phase := 0;
    end;
end;

procedure SetUpSpectrumNumberThree(parameter: Double);
begin
end;

procedure SetUpSpectrumNumberFour(parameter: Double);
begin
end;

// ******** PROCEDURES FOR SPECIFYING FUNCTIONS ******** //

procedure SetUpDefaultPrimaryFunction(parameter: Double);
var
  i: Integer;
  temp: Double;
begin
  temp := 2 * pi * deltaT;
  for i:=1 to NUMDATA+1 do
    primaryFunc[i] := sin((i-1) * temp);
end;

procedure SetUpFunctionNumberThree(parameter: Double);
begin
end;

procedure SetUpFunctionNumberFour(parameter: Double);
begin
end;

end.