Lecture 17: Function Representation by Fourier Series

Reading:
Kreyszig Sections: 11.1, 11.2, 11.3 (pages478–485, 487–489, 490–495)

Periodic Functions

Periodic functions should be familiar to everyone. The keeping of time, the ebb and flow of tides, the patterns and textures of our buildings, decorations, and vestments invoke repetition and periodicity that seem to be inseparable from the elements of human cognition.[9] Although other species utilize music for purposes that we can only imagine—we seem to derive emotion and enjoyment from making and experience of music.

[9] I hope you enjoy the lyrical quality of the prose. While I wonder again if anyone is reading these notes, my wistfulness is taking a poetic turn:

> They repeat themselves
> What is here, will be there
> It wills, willing, to be again
> spring; neap, ebb and flow, wane; wax
> sow; reap, warp and woof, motif; melody.
> The changed changes. We remain
> Perpetually, Immutably, Endlessly.

Playing with Audible Periodic Phenomena

notebook (non-evaluated) pdf (evaluated) html (evaluated)

Several example of creating sounds using mathematical functions are illustrated for education and amusement.

1: The seven musical notes around middle C indexed here with integers and then their frequencies (in hertz) are defined with a `freq`. The function *Note* takes one of the seven indexed notes and creates a wave-form for that note. The function `Play` takes the waveform and produces audio output.

2: To superimpose notes together to make a chord, it would be convenient to `Map` the function *Note* over a list...

3: The easiest way to exend a function so that it executes over a list is to use `SetAttributes` and declare the function to be `Listable`.

5: Like the function `Plot`, `Play` will frequently need `Evaluate` called on nontrivial arguments.

6: *Chord* make an ascending list of every second note and then uses `Mod` to map those notes onto the primary domain (0,1,...,6).

8: If different notes are wanted at different times, an `If` statement can be used.

9: This is the sequence of notes associated with the displayed musical score.

10: *Beats* is a function that takes a list of notes and arranges them into a list where each member is an `If` statement stating *when* and for what *duration* it should play. In addition to the sequence of notes, the function takes two arguments, *cadence* and *duration* ¡ which specifiy how quickly and how long to sustain the notes.

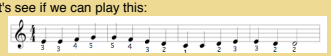
11: This is musical score with notes played every 0.75 seconds and held for 0.5 second. Joy.

12: This is random "music." Oh boy.

13: This is noise generated from a function. Enjoy.

```
1  c = 0; d = 1; e = 2; f = 3; g = 4; a = 5; b = 6;
   freq[c] = 261.6; freq[d] = 293.7; freq[e] = 329.6;
   freq[f] = 349.2; freq[g] = 392.0;
   freq[a] = 440.0; freq[b] = 493.9;
   Note[note_] := Sin[2 Pi freq[note] t];
   Play[Note[c], {t, 0, 2}]
2  Note[{c, e}]
3  SetAttributes[Note, Listable]
4  Play[{Note[c], Note[e]}, {t, 0, 2}]
5  Play[Evaluate[Note[{c, e}]], {t, 0, 2}]
6  Chord[note_] := Table[Note[Mod[note + i, 6]], {i, 0, 4, 2}]
7  Play[Evaluate[Chord[e]], {t, 0, 2}]
8  Play[If[t > 0.25 && t < 1.25, Note[a], Note[c]], {t, 0, 1.5}]
```

Let's see if we can play this:



```
9  twoframes = {e, e, f, g, g, f, e, d, c, c, d, e}
10 Beats[list_, duration_, cadence_] := Table[If[
       t ≥ (i − 1) * cadence && t ≤ (i − 1) * cadence + duration,
       Evaluate[list[[i]]], 0], {i, 1, Length[list]}]
11 Play[Evaluate[Beats[Note[twoframes], 0.5, 0.75]], {t, 0, 12}]
12 randomnotes =
     Map[Note, Table[Random[Integer, {0, 6}], {24}]]
13 Play[Evaluate[Beats[randomnotes, 0.5, 0.5]], {t, 0, 12}]
14 Play[Sin[1000 x Sin[Exp[x/3] + Sin[x]/x]] +
     Exp[x/10] Sin[x] Sin[1500 x], {x, −20, 10}]
```

3.016 Home

Full Screen

Close

Quit

A function that is periodic in a single variable can be expressed as:

$$f(x+\lambda) = f(x) \qquad f(t+\tau) = f(t) \tag{17-1}$$

The first form is a suggestion of a spatially periodic function with wavelength λ and the second form suggests a function that is periodic in time with period τ. Of course, both forms are identical and express that the function has the same value at an infinite number of points ($x = n\lambda$ in space or $t = n\tau$ in time where n is an integer.)

Specification of a periodic function, $f(x)$, within one period $x \in (x_o, x_o + \lambda)$ defines the function everywhere. The most familiar periodic functions are the trigonometric functions:

$$\sin(x) = \sin(x + 2\pi) \quad \text{and} \quad \cos(x) = \cos(x + 2\pi) \tag{17-2}$$

However, any function can be turned into a periodic function.

3.016 Home

Full Screen

Close

Quit

Using "Mod" to Create Periodic Functions

notebook (non-evaluated) pdf (evaluated) html (evaluated)

Periodic functions are often associated with the "modulus" operation. `Mod`$[x, \lambda]$ is the remainder of the result of *recursively* dividing x by λ until the result lies in the domain $0 \leq \text{Mod}[x, \lambda] < \lambda)$. Another way to think of modulus is to find the "point" where are periodic function should be evaluated if its primary domain is $x \in (0, \lambda)$.

1: *Boomerang* uses `Mod` on the argument of any function `f` of a single argument to map the argument into the domain $(0, \lambda)$. Therefore, calling *Boomerang* on any function will create a infinitely periodic repetition of the function in the domain $(0, \lambda)$.

3: `Plot` called on the periodic extension of wavelength $\lambda = 6$ of a function illustrates the effect of *Boomerang* . a periodic function with a specified period.

Boomerang uses Mod to force a function, f, with a single argument, x, to be periodic with wavelength λ

```
1 Boomerang[f_, x_, λ_] := f[Mod[x, λ]]
2 AFunction[x_] := ((3 - x)^3)/27
```

The following step uses **Boomerang** to produce a periodic repetition of **AFunction** over the range $0 < x < 6$:

```
3 Plot[Boomerang[AFunction, x, 6],
    {x, -12, 12}, PlotRange → All]
```

3.016 Home

Full Screen

Close

Quit

Odd and Even Functions

The trigonometric functions have the additional properties of being an *odd* function about the point $x = 0$: $f_{\text{odd}} : f_{\text{odd}}(x) = -f_{\text{odd}}(-x)$ in the case of the sine, and an *even* function in the case of the cosine: $f_{\text{even}} : f_{\text{even}}(x) = f_{\text{even}}(-x)$.

This can generalized to say that a function is even or odd about a point $\lambda/2$: $f_{\text{odd}\frac{\lambda}{2}} : f_{\text{odd}\frac{\lambda}{2}}(\lambda/2+x) = -f_{\text{odd}\frac{\lambda}{2}}(\lambda/2-x)$ and $f_{\text{even}\frac{\lambda}{2}} : f_{\text{even}\frac{\lambda}{2}}(\lambda/2+x) = f_{\text{even}\frac{\lambda}{2}}(\lambda/2-x)$.

Any function can be decomposed into an odd and even sum:

$$g(x) = g_{\text{even}} + g_{\text{odd}} \tag{17-3}$$

The sine and cosine functions can be considered the odd and even parts of the generalized trigonometric function:

$$e^{ix} = \cos(x) + \imath \sin(x) \tag{17-4}$$

with period 2π.

Representing a particular function with a sum of other functions

A Taylor expansion approximates the behavior of a suitably defined function, $f(x)$ in the neighborhood of a point, x_o, with a bunch of functions, $p_i(x)$, defined by the set of powers:

$$p_i \equiv \vec{p} = (x^0, x^1, \ldots, x^j, \ldots) \tag{17-5}$$

The polynomial that approximates the function is given by:

$$f(x) = \vec{A} \cdot \vec{p} \tag{17-6}$$

where the vector of coefficients is defined by:

$$A_i \equiv \vec{A} = (\frac{1}{0!} f(x_o), \frac{1}{1!} \frac{df}{dx}\Big|_{x_o}, \ldots, \frac{1}{j!} \frac{d^j f}{dx^j}\Big|_{x_o}, \ldots) \tag{17-7}$$

The idea of a vector of infinite length has not been formally introduced, but the idea that as the number of terms in the sum in Eq. 17-6 gets larger and larger, the approximation should converge to

the function. In the limit of an infinite number of terms in the sum (or the vectors of infinite length) the series expansion will converge to $f(x)$ if it satisfies some technical continuity constraints.

However, for periodic functions, the domain over which the approximation is required is only one period of the periodic function—the rest of the function is taken care of by the definition of periodicity in the function.

Because the function is periodic, it makes sense to use functions that have the same period to approximate it. The simplest periodic functions are the trigonometric functions. If the period is λ, any other periodic function with periods $\lambda/2$, $\lambda/3$, λ/N, will also have period λ. Using these "sub-periodic" trigonometric functions is the idea behind Fourier Series.

Fourier Series

The functions $\cos(2\pi x/\lambda)$ and $\sin(2\pi x/\lambda)$ each have period λ. That is, they each take on the same value at x and $x+\lambda$.

There are an infinite number of other simple trigonometric functions that are periodic in λ; they are $\cos[2\pi x/(\lambda/2))]$ and $\sin[2\pi x/(\lambda/2))]$ and which cycle two times within each λ, $\cos[2\pi x/(\lambda/3))]$ and $\sin[2\pi x/(\lambda/3))]$ and which cycle three times within each λ, and, in general, $\cos[2\pi x/(\lambda/n))]$ and $\sin[2\pi x/(\lambda/n))]$ and which cycle n times within each λ.

The constant function, $a_0(x) = \text{const}$, also satisfies the periodicity requirement.

The superposition of multiples of any number of periodic function must also be a periodic function, therefore any function $f(x)$ that satisfies:

$$\begin{aligned} f(x) &= \mathcal{E}_0 + \sum_{n=1}^{\infty} \mathcal{E}_n \cos\left(\frac{2\pi n}{\lambda}x\right) + \sum_{n=1}^{\infty} \mathcal{O}_n \sin\left(\frac{2\pi n}{\lambda}x\right) \\ &= \mathcal{E}_{k_0} + \sum_{n=1}^{\infty} \mathcal{E}_{k_n} \cos(k_n x) + \sum_{n=1}^{\infty} \mathcal{O}_{k_n} \sin(k_n x) \end{aligned} \tag{17-8}$$

where the k_i are the *wave-numbers* or *reciprocal wavelengths* defined by $k_j \equiv 2\pi j/\lambda$. The k's represent inverse wavelengths—large values of k represent short-period or high-frequency terms.

If any periodic function $f(x)$ could be represented by the series in in Eq. 17-8 by a suitable choice of coefficients, then an alternative representation of the periodic function could be obtained in terms of the simple trigonometric functions and their amplitudes.

3.016 Home
Full Screen
Close
Quit

The "inverse question" remains: "How are the amplitudes $\mathcal{E}_{k_n}$ (the even trigonometric terms) and $\mathcal{O}_{k_n}$ (the odd trigonometric terms) determined for a given $f(x)$?"

The method follows from what appears to be a "trick." The following three integrals have simple forms for integers M and N:

$$\begin{aligned}
&\int_{x_0}^{x_0+\lambda} \sin\left(\frac{2\pi M}{\lambda}x\right)\sin\left(\frac{2\pi N}{\lambda}x\right)dx = \begin{cases} \frac{\lambda}{2} \text{ if } M = N \\ 0 \text{ if } M \neq N \end{cases} \\
&\int_{x_0}^{x_0+\lambda} \cos\left(\frac{2\pi M}{\lambda}x\right)\cos\left(\frac{2\pi N}{\lambda}x\right)dx = \begin{cases} \frac{\lambda}{2} \text{ if } M = N \\ 0 \text{ if } M \neq N \end{cases} \\
&\int_{x_0}^{x_0+\lambda} \cos\left(\frac{2\pi M}{\lambda}x\right)\sin\left(\frac{2\pi N}{\lambda}x\right)dx = 0 \text{ for any integers } M, N
\end{aligned} \tag{17-9}$$

The following shows a demonstration of this *orthogonality relation for the trignometric functions.*

3.016 Home

Full Screen

Close

Quit

Orthogonality of Trignometric Functions

notebook (non-evaluated) pdf (evaluated) html (evaluated)

Demonstrating that the relations in Eq. 17-9 are true.

1: Using `Integrate` for $\cos(2\pi Mx/\lambda)\cos(2\pi Nx/\lambda)$ over a definite interval of a single wavelength, does not produce a result that obviously vanishes for $M \neq N$.

2: However, replacing any of the symbolic integers with actual integers results in a zero. So, one the orthogonality relation is plausible.

3: Using `Assuming` and `Limit`, one can show that the relation ship vanishes for $N = M$. Although, it is a bit odd to be thinking about continuous limits with integers.

5: Similarly for $\int \cos(2\pi Mx/\lambda)\sin(2\pi Nx/\lambda)dx$.

9: and for $\int \sin(2\pi Mx/\lambda)\sin(2\pi Nx/\lambda)dx$.

```
1  coscos = Integrate[Cos[(2 π Minteger x)/λ] Cos[(2 π Ninteger x)/λ],
     {x, xo, xo + λ}, Assumptions → {Minteger ∈ Integers,
       Ninteger ∈ Integers, xo ∈ Reals, λ > 0}]
2  Simplify[coscos /. {Minteger → 4 , Ninteger → 34}]
3  Assuming[Minteger ∈ Integers &&
     Ninteger ∈ Integers && xo ∈ Reals && λ ∈ Reals,
    Limit[coscos, Minteger → Ninteger]]
4  cossin = Integrate[Cos[(2 π Minteger x)/λ] Sin[(2 π Ninteger x)/λ],
     {x, xo, xo + λ}, Assumptions → {Minteger ∈ Integers,
       Ninteger ∈ Integers, xo ∈ Reals, λ > 0}]
5  Simplify[cossin /. {Minteger → -7 , Ninteger → 35}]
6  Assuming[Minteger ∈ Integers &&
     Ninteger ∈ Integers && xo ∈ Reals && λ ∈ Reals,
    Limit[cossin, Minteger → Ninteger]]
7  sinsin = Integrate[Sin[(2 π Minteger x)/λ] Sin[(2 π Ninteger x)/λ],
     {x, xo, xo + λ}, Assumptions → {Minteger ∈ Integers,
       Ninteger ∈ Integers, xo ∈ Reals, λ > 0}]
8  Simplify[sinsin /. {Minteger → 10 , Ninteger → 9}]
9  Assuming[Minteger ∈ Integers &&
     Ninteger ∈ Integers && xo ∈ Reals && λ ∈ Reals,
    Limit[sinsin, Minteger → Ninteger]]
```

Using this orthogonality trick, any amplitude can be determined by multiplying both sides of Eq. 17-8 by its conjugate trigonometric function and integrating over the domain. (Here we pick the domain to start at zero, $x \in (0, \lambda)$, but any other starting point would work fine.)

$$\cos(k_M x) f(x) = \cos(k_M x) \left(\mathcal{E}_{k_0} + \sum_{n=1}^{\infty} \mathcal{E}_{k_n} \cos(k_n x) + \sum_{n=1}^{\infty} \mathcal{O}_{k_n} \sin(k_n x) \right)$$

$$\int_0^\lambda \cos(k_M x) f(x) dx = \int_0^\lambda \cos(k_M x) \left(\mathcal{E}_{k_0} + \sum_{n=1}^{\infty} \mathcal{E}_{k_n} \cos(k_n x) + \sum_{n=1}^{\infty} \mathcal{O}_{k_n} \sin(k_n x) \right) dx \tag{17-10}$$

$$\int_0^\lambda \cos(k_M x) f(x) dx = \frac{\lambda}{2} \mathcal{E}_{k_M}$$

This provides a formula to calculate the even coefficients (amplitudes) and multiplying by a sin function provides a way to calculate the odd coefficients (amplitudes) for $f(x)$ periodic in the fundamental domain $x \in (0, \lambda)$.

$$\begin{aligned} \mathcal{E}_{k_0} &= \frac{1}{\lambda} \int_0^\lambda f(x) dx \\ \mathcal{E}_{k_N} &= \frac{2}{\lambda} \int_0^\lambda f(x) \cos(k_N x) dx \qquad k_N \equiv \frac{2\pi N}{\lambda} \\ \mathcal{O}_{k_N} &= \frac{2}{\lambda} \int_0^\lambda f(x) \sin(k_N x) dx \qquad k_N \equiv \frac{2\pi N}{\lambda} \end{aligned} \tag{17-11}$$

The constant term has an extra factor of two because $\int_0^\lambda \mathcal{E}_{k_0} dx = \lambda \mathcal{E}_{k_0}$ instead of the $\lambda/2$ found in Eq. 17-9.

Other forms of the Fourier coefficients

Sometimes the primary domain is defined with a different starting point and different symbols, for instance Kreyszig uses a centered domain by using $-L$ as the starting point and $2L$ as the period, and in these cases the forms for the Fourier coefficients look a bit different. One needs to look at the domain in order to determine which form of the formulas to use.

Potentially interesting but currently unnecessary

The "trick" of multiplying both sides of Eq. 17-8 by a function and integrating comes from the fact that the trigonometric functions form an orthogonal basis *for functions with inner product defined by*

$$f(x) \cdot g(x) = \int_0^\lambda f(x) g(x) dx$$

Considering the trigonometric functions as components of a vector:

$$\begin{aligned}
\vec{e_0}(x) &= (1, 0, 0, \dots,) \\
\vec{e_1}(x) &= (0, \cos(k_1 x), 0, \dots,) \\
\vec{e_2}(x) &= (0, 0, \sin(k_1 x), \dots,) \\
\dots &= \quad \vdots \\
\vec{e_n}(x) &= (\dots\dots, \sin(k_n x), \dots,)
\end{aligned}$$

then these "basis vectors" satisfy $\vec{e_i} \cdot \vec{e_j} = (\lambda/2)\delta_{ij}$*, where* $\delta_{ij} = 0$ *unless* $i = j$*. The trick is just that, for an arbitrary function represented by the basis vectors,* $\vec{P}(x) \cdot \vec{e_j}(x) = (\lambda/2) P_j$*.*

Calculating Fourier Series Amplitudes

notebook (non-evaluated) pdf (evaluated) html (evaluated)

Functions are developed which compute the even (cosine) amplitudes and odd (sine) amplitudes for an input function of one variable. These functions are extended to produce the first N terms of a Fouriers series.

1: *EvenTerms* computes symbolic representations of the even (cosine) coefficients using the formulas in Eq. 17-11. The $N = 0$ term is computed with a supplemental defintion because of its extra factor of 2. The domain is chosen so that it begins at $x = 0$ and ends at $x = \lambda$.

2: *OddTerms* performs a similar computation for the sine-coefficients; the $N = 0$ amplitude is set to zero explicitely. It will become convenient to include the zeroth-order coefficient for the odd (sine) series which vanishes by definition. The functions work by doing an integral for each term—this is not very efficient. It would be more efficient to calculate the integral symbolically once and then evaluate it once for each term.

4: efOddAmplitudeVector and *EvenAmplitudeVectors* create *amplitude vectors* for the cosine and sine terms with specified lengths and domains.

5: This function, $f(x) = x(1-x)^2(2-x)$, will be used for particular examples of Fourier series, note that it is an even function over $0 < x < 2\ldots$

7: The functions, *OddBasisVector* and *EvenBasisVector* , create vectors of *basis functions* of specified lengths and perioidic domains.

8: The Fourier series up to a certain order can be defined as the sum of two inner (dot) products: the inner product of the odd coefficient vector and the sine basis vector, and the inner product of the even coefficient vector and the cosine basis vector.

12: This will illustrate the approximation for a truncated ($N = 6$) Fourier series

```
1  EvenTerms[0, function_, λ_] :=
     1/λ Integrate[function[dummy], {dummy, 0, λ}]
   EvenTerms[SP_Integer, function_, λ_] :=
     2/λ Integrate[function[z] * Cos[(2 * SP * Pi * z)/λ], {z, 0, λ}]

2  OddTerms[0, function_, wavelength_] := 0
   OddTerms[SP_Integer, function_, λ_] :=
     2/λ Integrate[function[z] * Sin[(2 * SP * Pi * z)/λ], {z, 0, λ}]

3  OddAmplitudeVector[
       NTerms_Integer, function_, wavelength_] :=
     Table[OddTerms[i, function, wavelength], {i, 0, NTerms}]

4  EvenAmplitudeVector[
       NTerms_Integer, function_, wavelength_] :=
     Table[EvenTerms[i, function, wavelength], {i, 0, NTerms}]

5  myfunction[x_] := (x * (2 - x) * (1 - x)^2)

6  OriginalPlot = Plot[myfunction[x], {x, 0, 2},
     PlotStyle → {Hue[1], Thickness[0.015]}]

7  OddBasisVector[NTerms_Integer, var_, wavelength_] :=
     Table[Sin[2 π i var / wavelength], {i, 0, NTerms}]

8  EvenBasisVector[NTerms_Integer, var_, wavelength_] :=
     Table[Cos[2 π i var / wavelength], {i, 0, NTerms}]

9  FourierTruncSeries[n_, function_, var_, wavelength_] :=
     EvenAmplitudeVector[n, function, wavelength].
       EvenBasisVector[n, var, wavelength] +
     OddAmplitudeVector[n, function, wavelength].
       OddBasisVector[n, var, wavelength]

10 FourierTruncSeries[6, myfunction, x, 2]

11 FourierPlot =
     Plot[FourierTruncSeries[6, myfunction, x, 2], {x, −2, 4}]

12 Show[OriginalPlot, FourierPlot]
```

3.016 Home

◀◀ ◀ ▶ ▶▶

Full Screen

Close

Quit

Using the `Calculus‘FourierTransform‘` package

notebook (non-evaluated) pdf (evaluated) html (evaluated)

Fourier series expansions are a common and useful mathematical tool, and it is not surprising that MATHEMATICA® would have a package to do this and replace the inefficient functions defined in the previous example.

1: The functions in `Calculus‘FourierTransform‘` are designed to operate on the unit period located at $x \in (-1/2, 1/2)$. Therefore, the domains of functions of interest must be mapped onto this domain by a change of variables.

4: *ReduceHalfHalf* is an example of a function design to do the required mapping. First the length of original domain is mapped to unity by dividing through by λ and then the origin is shifted by mapping the x (that the MATHEMATICA® functions will see) to $(0, 1)$ with the transformation $x \to x + \frac{1}{2}$.

8: Particular amplitudes of the properly remapped function can be obtained with the functions `FourierCosCoefficient` and `FourierSinCoefficient`. In this example, a symbolic n is entered and a symbolic representation of the n^{th} amplitude is returned.

9: A truncated Fourier series can be obtained symbolically to any order with `FourierTrigSeries`.

```
1 << Calculus`FourierTransform`
2 AFunction[x_] := (x - 3)^3 / 27
3 Plot[AFunction[x], {x, 0, 6}]
```

Mathematica's Fourier Series functions are defined for function that are periodic in the domain x ∈ (-1/2,1/2). So we need to map the periodic functions to this domain

```
4 ReduceHalfHalf[f_, x_, λ_] := f[(x + 1/2) * λ]
5 ReducedFunction =
    ReduceHalfHalf[AFunction, x, 6] // Simplify
6 Plot[ReducedFunction, {x, -1/2, 1/2}, PlotRange → All]
7 FourierCosCoefficient[ReducedFunction, x, n]
8 FourierSinCoefficient[ReducedFunction, x, n]
9 FourierTrigSeries[ReducedFunction, x, 5]
```

Visualizing Convergence of the Fourier Series: Gibbs Phenomenon

notebook (non-evaluated) pdf (evaluated) html (evaluated)

Functions that produce animations (each frame representing a different order of truncation of the Fourier series) are developed. This example illustrates *Gibbs phenomenon* where the approximating function oscillates wildly near discontinuities in the original function.

1: *AnimateTruncatedFourierSeries* is a simple example of an animation function for the truncated Fourier series. It uses the `Table` function with three arguments in the iterator for the initial truncation `truncationstart`, final truncation, and the number to skip in between....

2: However, because the entire series is recomputed for each frame, the function above is not very efficient. In this second version, only two arguments are supplied to the iterator. At each frame, the two N^{th} Fourier terms are added to the sum of terms computed previously.

3: Because *ReducedFunction* has a discontinuity (its end-value and intitial value differ), this animation will show Gibbs phenomena near the edges of the domain.

5: `FourierCosCoefficient` will show a frequently observed feature in the amplitudes of Fourier coefficients. The amplitude's magnitude becomes smaller and smaller with larger n, and the sign of the terms tend to oscillate between positive and negative values.

8: Plots of the magnitudes of the amplitudes are a "signature" of the original function in a new space. Each term indicates what "amount" of each periodicity is present in the original function. The plot could be interpreted as a "frequency" or "wavelength" representation of the original function.

```
1  AnimateTruncatedFourierSeries[function_,
       {truncationstart_, truncationend_, truncjump_}] :=
     Table[Plot[Evaluate[FourierTrigSeries[function, x, itrunc]],
       {x, -1, 1}, PlotRange → {-2, 2}],
       {itrunc, truncationstart, truncationend, truncjump}];
```

The function is demonstrative, but is inefficient because coefficients are recalculated needlessly. A more efficient version appears below.

```
2  AnimateTruncatedFourierSeries[function_,
       {truncationstart_, truncationend_}] :=
     Module[{coscof, sincof, currentappx, n, TwoPi = 2π},
       currentappx = FourierCosCoefficient[function, x, 0];
       coscof[n_] =
         Simplify[FourierCosCoefficient[function, x, n]];
       sincof[n_] = Simplify[
           FourierSinCoefficient[function, x, n]];

       Table[Plot[Evaluate[currentappx +=
             coscof[itrunc] * Cos[TwoPi itrunc x] +
               sincof[itrunc] * Sin[TwoPi itrunc x]],
         {x, -1, 1}, PlotRange → {-2, 2}],
       {itrunc, truncationstart, truncationend}];]
```

The following will demonstrate how convergence is difficult where the function changes rapidly---this is known as Gibbs' Phenomenon

```
3  AnimateTruncatedFourierSeries[ReducedFunction, {1, 60}]
```

```
4  ReducedMyFunction =
     ReduceHalfHalf[myfunction, x, 2] // Simplify
```

General form of the even amplitudes

```
5  myfunccos = FourierCosCoefficient[4 x^2 - 16 x^4, x, n]
```

```
6  FourierSinCoefficient[4 x^2 - 16 x^4, x, n]
```

```
7  ListPlot[Table[myfunccos, {n, 1, 50}]]
```

```
8  ListPlot[Table[myfunccos, {n, 1, 10}],
     PlotJoined → True, PlotRange → All]
```

3.016 Home

◀◀ ◀ ▶ ▶▶

Full Screen

Close

Quit

Complex Form of the Fourier Series

The behavior of the Fourier coefficients for both the odd (sine) and for the even (cosine) terms was illustrated above. Functions that are even about the center of the fundamental domain (reflection symmetry) will have only even terms—all the sine terms will vanish. Functions that are odd about the center of the fundamental domain (reflections across the center of the domain and then across the x-axis.) will have only odd terms—all the cosine terms will vanish.

Functions with no odd or even symmetry will have both types of terms (odd and even) in its expansion. This is a restatement of the fact that any function can be decomposed into odd and even parts (see Eq. 17-3).

This suggests a short-hand in Eq. 17-4 can be used that combines both odd and even series into one single form. However, because the odd terms will all be multiplied by the imaginary number $\imath$, the coefficients will generally be complex. Also because $\cos(nx) = (\exp(inx) + \exp(-inx))/2$, writing the sum in terms of exponential functions only will require that the sum must be over both positive and negative integers.

For a periodic domain $x \in (0, \lambda)$, $f(x) = f(x + \lambda)$, the *complex form of the fourier series is given by:*

$$\begin{aligned} f(x) &= \sum_{n=-\infty}^{\infty} \mathcal{C}_{k_n} e^{\imath k_n x} \qquad \text{where } k_n \equiv \frac{2\pi n}{\lambda} \\ \mathcal{C}_{k_n} &= \frac{1}{\lambda} \int_0^{\lambda} f(x) e^{-\imath k_n x} dx \end{aligned} \tag{17-12}$$

Because of the orthogonality of the basis functions $\exp(\imath k_n x)$, the domain can be moved to any wavelength, the following is also true:

$$\begin{aligned} f(x) &= \sum_{n=-\infty}^{\infty} \mathcal{C}_{k_n} e^{\imath k_n x} \qquad \text{where } k_n \equiv \frac{2\pi n}{\lambda} \\ \mathcal{C}_{k_n} &= \frac{1}{\lambda} \int_{-\lambda/2}^{\lambda/2} f(x) e^{-\imath k_n x} dx \end{aligned} \tag{17-13}$$

although the coefficients may have a different form.

Index