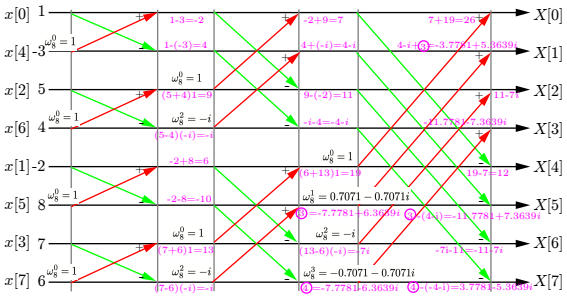
Fourier Methods

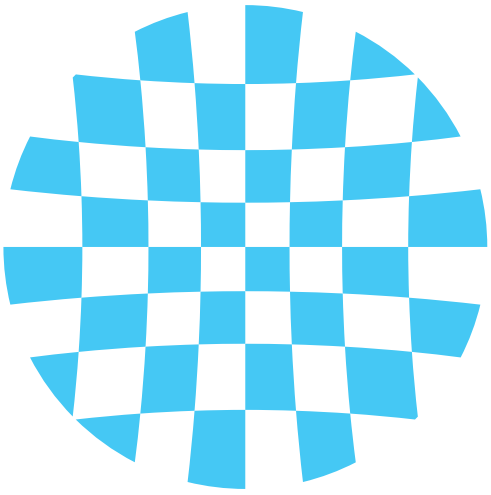
and

Related Topics



David Salomon

To the memory of Harry Nyquist and Vladimir Kotelnikov,
unsung geniuses



There is no such thing as a moral or an immoral book.
Books are well written, or badly written. That is all.
—Oscar Wilde

Preface

- Everything is made of four elements: earth, water, air, and fire. (The ancient Greeks).
- The world is made of atoms. (Physics of the early 20th century).
- Our world is made of three building blocks: the up quark, the down quark, and the electron. (Physics of the 1960's).
- The standard model lists and describes the elementary constituents of the universe. (Physics of the 1990's).
- The universe consists of normal matter, dark matter, and dark energy. The former gives rise to the atoms that make up stars, planets, and all other visible objects in the universe. The latter are still unknown. (Physics of the early 21st century).
- The only reality is quantum fields. (Physics of the early 21st century).
- The world? It consists of pure waves. (Joseph Fourier, around 1820).

Ripley's Believe It or Not! was started in 1918 by Robert Ripley. It was initially a newspaper panel describing unusual sports feats, but it has quickly grown to become a commercial empire that deals in bizarre events and items so strange and unusual that readers might question their existence. The book you are now reading is similar. It deals with strange, hard to believe topics that are nevertheless true—are beautiful, elegant, and pervasive—and affect modern life in profound ways.

The main subject of this book is Fourier methods. Essentially, these methods take objects, either one-dimensional (waves) or two-dimensional (images), and break them up into elementary components that are pure sine waves. In principle, these methods can also be applied to everyday familiar three-dimensional objects and break them up into three-dimensional waves. Thus, in some sense, Fourier waves are the basic, elementary constituents of our universe.

Here are two examples. The left part of Figure 1 shows how a step function, which consists of straight segments and sharp corners, can be expanded into an infinite sum of sine waves. The waves have different frequencies, amplitudes, and phases, but the

surprising fact is that even though they are all smooth and none has corners, their infinite sum is identical to the original step function and features the same straight segments and 90° corners.

A note. The right-hand part of the figure shows how a finite sum of the Fourier series of the step function features overshoots at the corners. A careful analysis of this behavior (see, for example, reference [Gibbs 22]) indicates that the height of the overshoot approaches a finite limit and is never zero, but its width shrinks to zero as more terms are added. This is known as the Gibbs phenomenon and it exists in the Fourier expansion of any function with corners.

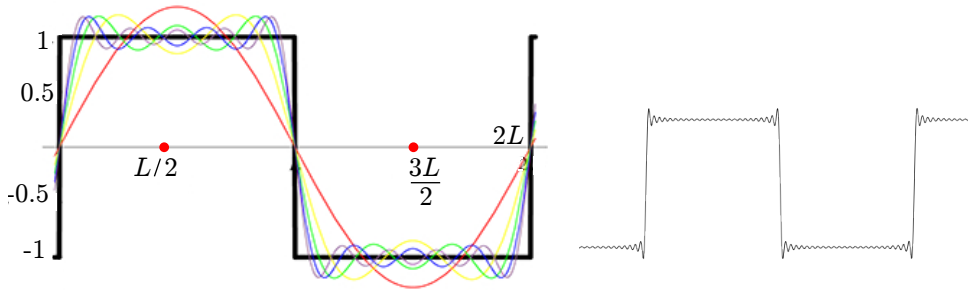


Figure 1: A Step Function and its Fourier Series.

The next example, Figure 2, is even more astonishing. Any grayscale image can be broken up into a (possibly infinite) sum of two-dimensional sine waves. Those waves also have different frequencies and amplitudes, but the surprising fact here is that while each wave is regular and exhibits some symmetry, their sum (finite or infinite) is irregular and non-symmetric. Section 4.7 lists and explains practical Mathematica code that implements the discrete Fourier transform. This important mathematical transform is finite. It can generate an arbitrary digital image by adding together a finite number of two-dimensional sine waves, each of them regular, ending up with a digital (pixel based) image that has no regular features. This result is explained on Page 97 under **Seeing and "seeing"**.

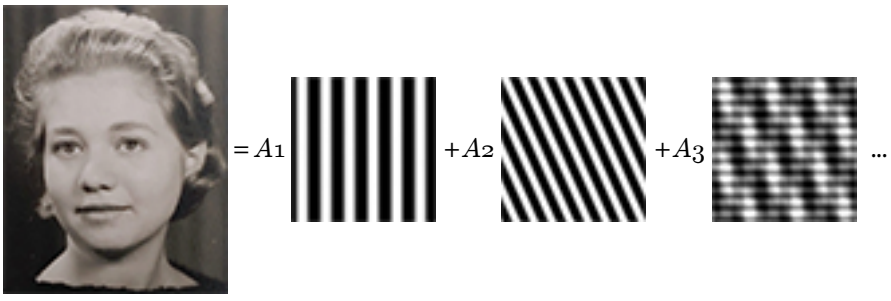


Figure 2: A Grayscale Image and Some 2D Sine Waves.

The Weirdness of the Infinite

How can an infinite sum differ so much from any of its (finite) parts? I would like to argue that this is only one of the many strange, unexpected, and unexplained properties of the infinite. We have to realize that the infinite is not always just an extension of the finite, and that infinity is not a number; it should not be naively manipulated and operated on as a number. This realization came about after more than a century of thinking and research, and here are some examples of what can happen when we naively deal with the infinite as a straight extension of the finite.

- Consider the equation

$$x^{x^{x^{\cdot^{\cdot^{\cdot}}}}} = 2.$$

This scary, infinite equation is easy to solve when we write it as $x^y = 2$ and then realize that y, being an infinite ladder of x's, equals the entire left-hand side of the equation and therefore also equals the right-hand side, which is 2. We can then rewrite the equation as $x^2 = 2$ whose solution is $x = \sqrt{2}$. We have just proved that

$$\sqrt{2}^{\sqrt{2}^{\sqrt{2}^{\cdot^{\cdot^{\cdot}}}}} = 2.$$

Now consider the similar equation

$$x^{x^{x^{\cdot^{\cdot^{\cdot}}}}} = 4.$$

We can similarly rewrite it as $x^y = 4$ or $x^4 = 4$, with a solution $x^2 = 2$ and $x = \sqrt{2}$. We have now proved that

$$\sqrt{2}^{\sqrt{2}^{\sqrt{2}^{\cdot^{\cdot^{\cdot}}}}} = 4.$$

The conclusion is that either $2 = 4$ or that we made a mistake somewhere. The mistake was to treat infinity naively, without the respect it deserves. It turns out that $\sqrt{2}$ is a solution to the first equation, but the second equation has no solutions because the solutions of equations of the form

$$x^{x^{x^{\cdot^{\cdot^{\cdot}}}}} = \text{constant},$$

exist only in certain real intervals that do not include 4.

- There are various ways to prove that the infinite fraction

$$0.999\ldots \stackrel{\text{def}}{=} 0.9\overline{9}$$

equals 1. We now apply one of those proofs to the "opposite" huge number

$$\ldots 9999.0 \stackrel{\text{def}}{=} \overline{9}9.0,$$

Let's denote this number by x (is it infinite). Step 1 is to compute

$$10x = \ldots 99990.0.$$

In step 2 we add 9 to both sides, obtaining

$$10x + 9 = \ldots 99990.0 + 9 = \ldots 99999.0 = x,$$

or $10x+9 = x$, a simple equation whose solution is $x = -1$. This clearly makes no sense and is the result of treating the infinite carelessly when we multiplied both sides by 10 in step 1, simply by shifting x to the left.

The second subject of the book (and the topic of Chapter 5) is the unexpected, magical, Shannon-Nyquist sampling theorem. A concept that started as an intuitive theory in the 1920's and 1930's, and has developed since to become the basis of our digital world. In essence, this theorem states that an analog signal such as a sound wave, an electromagnetic wave, an electrical pulse, or an image (black-and-white, grayscale, or color) can be broken up into individual points (it can be sampled or digitized) and then fully reconstructed from those points, if the sampling rate is high enough. The theorem also specifies the ideal rate of sampling and the reconstruction process. This is surprising, because we feel that once a continuous object is broken up into points, its values in regions between the points have been permanently lost, and cannot be fully measured or computed from the known points.

Figure 3 is an example, showing 16 samples equally-spaced over a wave. The bottom part of the figure shows only the samples, and it is obvious that the task of reconstructing the original wave from these samples is not trivial. If the samples obey the theorem, it would be possible to fully reconstruct this complicated wave from just 16 numbers! It is as if 16 numbers somehow manage to incorporate the infinite information that, in principle, is contained in the original, analog wave.

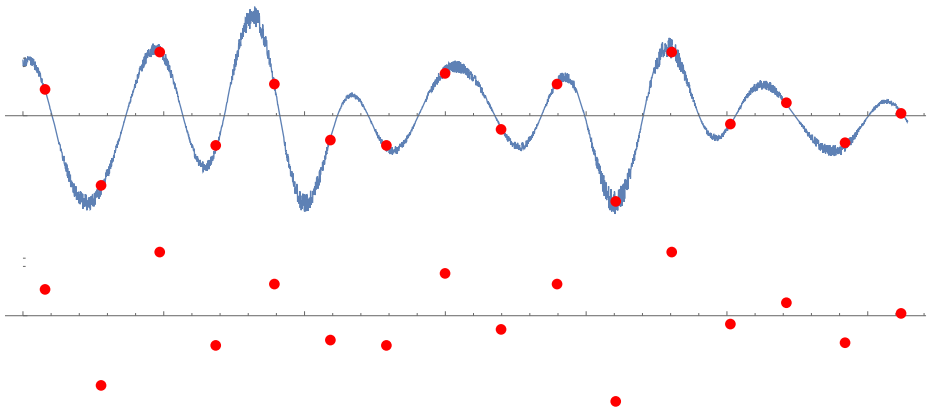


Figure 3: Equally-Spaced Samples over an Irregular Wave.

Chapter 6 is a short survey of convolution. This confusing topic is included in the book because it is related to the Fourier transform (both continuous and discrete). The basic idea of convolution is explained in such a way that its confusing aspect (the reverse direction of the analysing function) comes out naturally. The two-dimensional convolution is then presented with examples, and the chapter closes with a short section that illustrates the important convolution theorem.

> I'm really hungry for surprises because each one makes us ever-so-slightly but substantially smarter.
>
> Tadashi Tokieda, `tinyurl.com/ycyebzsp`

Resources

Because of the importance of the topics discussed in this book, there are myriads of books, articles, videos, and software applications that discuss, illuminate, and explain them. Out of this crowd I would like to single out the following, which have inspired and helped me much:

- Goodman, Joseph W., *Fourier Transforms Using Mathematica*, SPIE, 2020. This is a large Mathematica notebook that covers many Fourier related topics. It can be a time saver and helper to anyone interested in implementing the concepts discussed here. It can be purchased directly from spie.org, and if you buy the paper edition, you get a pointer to the PDF version as well.

- Smith, Alvy Ray, *A Biography of the Pixel*, The MIT Press: Leonardo series, 2021. An unusual book that covers, in addition to Fourier concepts and the sampling theorem, many topics related to pixels and digital media.

- Zach Star, *When the FBI had too many fingerprints in storage,* is a 2020 video at `youtube.com/watch?v=fRjFwTbJfes`

- Session 3, *Demonstration of 2D Fourier expansion*, is 2015 video at `youtube.com/watch?v=a7TUIkn3qjY&t=1394s` It uses Matlab to illustrate an implementation of the two-dimensional discrete Fourier transform, similar to what is discussed in Section 4.7.

The many references at the end of the book are useful. They cover topics and details that are outside the scope of this book but might still interest readers. Inside the text, a reference has the format `[name year]`, where `name` can be an author's name or any text related to the reference.

Target audience

This book is aimed toward those who feel comfortable with mathematical and engineering concepts, who would like to know more about the above topics, and who are familiar with the Mathematica software. The code snippets included in the book are mostly in Mathematica and use only the most basic features of this software. They can serve as basic building blocks on which readers can build more complex programs.

Errors?

Any comments, suggestions, and corrections are welcome and should be emailed to the author at `dsalomon@csun.edu`. However, if you notice something missing, consider the following quote (from Mark Twain) "A successful book is not made of what is in it, but of what is left out of it."

> The purpose of a preface is to make the reader want to read the book, and to convince the reader that he or she will enjoy reading it. It is also meant to provide some background information about the book, and to set the stage for the main content
>
> —Anonymous

Contents

What I have to say about this book can be found inside the book.
— Albert Einstein

Introduction

Fourier's Biography

The name Fourier is familiar to many in science and engineering, mainly because of his important and original ideas related to waves, as well as his work on heat transfer and on vibrations. Less known is his discovery of the greenhouse effect. Figure Intro.1, an engraving, probably from the 1820's, shows him at the height of his career.

Figure Intro.1: Jean-Baptiste Joseph Fourier.

Jean-Baptiste Joseph Fourier (born March 21, 1768, in Auxerre) spent his entire life in France, living in times of great turmoil (the result of the 1789 revolution) in this country. For many years he was either under the power of temporary, unpredictable, revolutionary governments or subject to the whims of Napoleon, a powerful dictator.

Fourier became an early orphan of a large, poor family. After a time in an elementary school at Auxerre (subsidized by the good citizens of that town), he enrolled in the École royale militaire, also in Auxerre, also helped by the same local citizens. It was there that Fourier met his great love, mathematics. So great was his attachment to this lady, that (so legend has it) he used to collect candle stubs so he could study late at night.

However, the army has never benefitted from Fourier's services. Being a mathematical geek as well as unhealthy, in 1787, after graduating from the École royale militaire, he changed direction and entered an abbey in Auxerre, aiming for the priesthood. Teaching mathematics in the abbey to novices, he initially ignored the historic events of July 14, 1789, while trying to publish early math works in the Academy of Science in Paris. Nevertheless, the revolution and its consequences became impossible to ignore, and so, three years later, Fourier left the Abbey before taking his final vows.

In 1793, Fourier has suddenly decided to become involved in politics. His debut speech, in Auxerre in February of that year, described his plan to convince local youth to serve in the new republican army. The town revolutionary politicians were duly impressed and invited him to join the revolution. Considering that 1793 saw the height of the famous reign of terror, even Fourier could see that it was unwise to refuse such an invitation.

Encouraged by his easy success in Auxerre, Fourier traveled to Orléans in an attempt to defend three local citizens wrongly accused. Unfortunately, the three were already on the list of perceived enemies of Roberspierre, which is why defending them cost Fourier his freedom. Being under arrest during the terror almost always led to the guillotine, but luckily for future science, Fourier was saved because just 10 days after his arrest, Robespierre, the leader of the terror, himself became its victim. After causing the execution, mostly by guillotine, of more than 17,000 "enemies of the Revolution" Robespierre was arrested, on July 27, 1794, and executed the very next day, thereby indirectly saving Fourier's life.

In addition to being free, Fourier was now a professor in Paris. First, in 1795, at the École Normale and later, in 1797, at the École Centrale de Travaux, which quickly became the new École Polytechnique. There Fourier succeeded Joseph-Louis Lagrange and became a colleague of Monge and Laplace. It was at the polytechnique, where he gained the reputation of a gifted teacher, that his name came to the attention of Napoleon, thereby starting a whole new chapter in Fourier's life.

In 1798, Napoleon, already an important general with impressive successes in Italy, decided to drive the British out of India and establish France as a power there instead. His first step was an expedition to Egypt, which was then under British influence. In May 1798 he left at the head of a large expedition that included, in addition to 25,000 soldiers, also Joseph Fourier, one of 167 scientists (savants) specializing in a variety of fields.

History relates that this campaign came to a bad end when admiral Nelson discovered the French fleet at Aboukir and had it captured and destroyed. Napoleon himself

managed to escape back to France in August 1799, but his army got trapped in Egypt and the middle east and was greatly reduced in numbers. Two years later, Fourier also managed to get back to France, where he resumed his career as a professor at the École Polytechnique.

In 1801, Napoleon, then a self-appointed first consul of France, appointed him prefect (provincial governor) of the department of Isère in Grenoble. It was during his tenure in Grenoble that Fourier succeeded in constructing an important highway between Grenoble and Turin and in draining the swamps of Bourgoin. For these works, as well as a book on the French Egyptian campaign, he was made a Baron.

Of more interest to us, it was during his tenure in Grenoble that Fourier developed his interest in heat flow. In his 1807 memoir "On the propagation of heat in solid bodies" he included the visionary sentence "Any function of a variable, whether continuous or discontinuous, can be expanded in a series of sines of multiples of the variable." This is recognized today as the Fourier series, but at the time, several French mathematicians objected to Fourier's ideas. They correctly claimed that his ideas did not rest on a firm mathematical foundation and may have been wrong. It was not until 1829, that Dirichlet, in Germany, has provided rigorous proofs of Fourier's ideas and techniques. Dirichlet's work has also provided the basis for the Fourier transform, which is so important to us.

Today, we use the term "signal" instead of Fourier's function. Common signals are sound, electrical voltage, and images (the latter are two-dimensional). Also, Fourier was interested in heat flow, so his variable was distance. To us, a more important variable is time. Thus, the statement above can be rewritten "Any sound can be broken down into a series of sine waves of different frequencies."

In 1822, Fourier became the permanent secretary of the French Academy of Sciences. In 1830, his health problems (attacks of aneurysm of the heart) became serious and he died on 4th May of that year.

The inner product

A vector is a mathematical entity that has two attributes, direction and magnitude. In particular, a vector does not have a location in space. Vectors are defined for Euclidean vector spaces, spaces whose elements are numbers (real or complex).

The dot product of two vectors $\mathbf{v}$ and $\mathbf{u}$ is the sum $\sum_n v_i\, u_i$ of the products of their components. It is a number (a scalar, which is why the dot product is sometimes referred to as a scalar product) and it also equals the product of the absolute values (magnitudes) of the vectors and the cosine of the angle between them. Thus $\mathbf{v} \cdot \mathbf{u} = |\mathbf{v}|\,|\mathbf{u}| \cos\theta$. This property, together with the definition of the dot product, implies the following results:

- Since $\cos 90^\circ$ is zero, the dot product of perpendicular vectors is zero.
- The dot product of a vector with itself is the sum of the squares of the vector's components.
- When dealing with matrices, a vector can be a row or a column. In such a case, one of the vectors may have to be transposed before they can be multiplied

The inner product is a generalization of the basic concept of the dot product and is defined for vector spaces whose elements are not numeric vectors. Examples of such spaces are polynomials, Riemann, and functions. The dot product is therefore a special case of the inner product.

In a vector space whose elements are functions, the inner product of two space elements (two functions) is an integral of the form

$$\langle f(x), g(x)\rangle \stackrel{\text{def}}{=} \int_a^b f(x)\, g(x)\, dx.$$

This is a definite integral, a number, and the integration limits a and b are chosen depending on the specific case.

If the functions are complex, their inner product is defined as

$$\langle f(x), g(x)\rangle \stackrel{\text{def}}{=} \int_a^b f(x)\, \overline{g}(x)\, dx,$$

where $\overline{g}(x)$ is the complex conjugate of $g(x)$.

Inner products are used as a measure of the "distance" between functions. Given two functions $f(x)$ and $g(x)$, we first decide on the interval $[a, b]$ that is of interest to us, and then compute the inner product of the functions over that interval. The bigger the result, the more similar the functions are at the given interval. If $f(x) \neq g(x)$ but $\langle f(x), g(x)\rangle = 0$, we say that the functions are orthogonal.

Figure Intro.2 is an example. The simple sinusoid `c1` (displayed in cyan) is compared to a basic cosine (in orange). Four versions with frequencies 1, 2, 3, and 4 radians/sec are shown. The inner products of the two functions are displayed in gray, and all that remains to be done (see Figure Intro.3) is to determine the sum of the gray areas by integration. The results of the four integrals are similarity measures (or similarity scores) that indicate how close `c1` is to the corresponding cosine.

```
Clear["Global`*"];
c1[t_] := 0.5 Cos[2t+Pi/6] + 0.4 Cos[3t+Pi/3];
g0 = Plot[c1[t], {t, 0, 2 Pi}, PlotStyle -> Cyan];
Do[g1 = Plot[c1[t] Cos[fr*t], {t, 0, 2Pi}, PlotStyle->Gray, Filling->Axis];
 g2 = Plot[Cos[fr*t], {t, 0, 2Pi}, PlotStyle -> Orange];
 Print[Show[g0, g1, g2, AspectRatio -> 1/Pi, PlotRange -> All,
   Ticks -> {{Pi/6, Pi, 2Pi}, Automatic}]], {fr, 1, 4}]
```

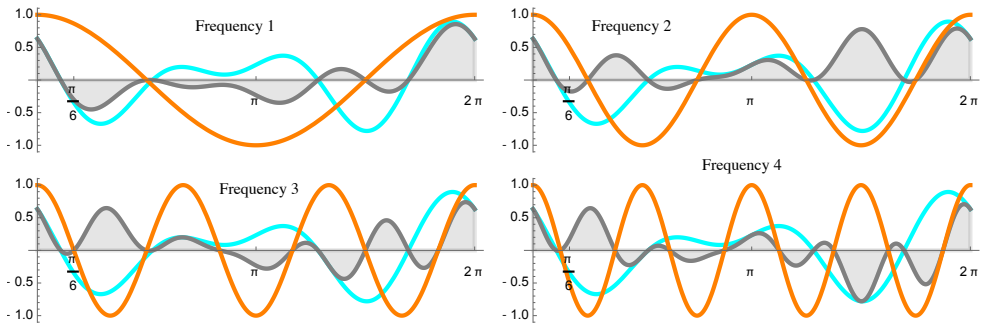


Figure Intro.2: The Inner Product as a Similarity Measure.

Fourier methods are based on inner product of functions as illustrated by the following example, where we try to analyze the simple sum (line 2 of Figure Intro.3)

$$c1(t) = 0.5\cos(2\pi 2t + \pi/6) + 0.4\cos(2\pi 3t + \pi/3).$$

The amplitude of this sum is $\sqrt{0.5^2 + 0.4^2} \approx 0.64$. Since this is an example, we know in advance the two terms that make up $c1(t)$, so we start (line 4) with an inner product `fi(ph)` of $c1(t)$ with a $\cos(2\pi 3t + \texttt{ph})$ and we perform it for phases `ph` ranging from 0 to 2π in steps of $\pi/12$. The resulting 25 values are collected (line 5) in table `tb` and are listed and plotted (lines 6,7 and the table in the figure). Notice that the integration interval is $[0, 2\pi]$, a full cycle. Both the list and the plot indicate that the biggest inner product, 1.24799, is obtained for phase $\texttt{ph} = \pi/3$, suggesting that a cosine function with this phase, a frequency of 3 Hz, and an amplitude of $2 \times 1.24799/(2\pi) = 0.4$ is one of building blocks of our $c1(t)$.

To compute the amplitude, we need to multiply the biggest inner product by 2 (because of the existence of negative frequencies, see Pages 61 and 72) and divide by the length 2π of the integration interval.

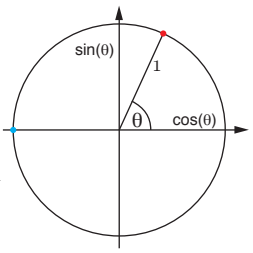
The next test is an inner product of $c1(t)$ with a $\cos(2\pi 2t + \texttt{ph})$, performed for the same 25 values of `ph`. This time, the biggest inner product, 1.56755 (not shown in the figure), is obtained for phase $\texttt{ph} = \pi/6$, suggesting that a cosine function with this phase, a frequency of 2 Hz, and an amplitude of $1.56755/\pi = 0.5$ is another major component of $c1(t)$.

More tests, with $\cos(2\pi f t + \texttt{ph})$ and frequencies f of 0, 1, 4, and 5 Hz, yield biggest inner products of 0.0682589 (at a phase of π), 0.0718558 (at $2\pi/3$), 0.0606261 (at $11\pi/6$), and 0.0387279 (at $19\pi/12$), respectively. These correspond to cosines with small amplitudes, in the range of 0.01–0.02.

This simple example illustrates the intimate relation between Fourier analysis and the inner product. The latter lies at the heart of the process of identifying the component waves of periodic functions as well as transforming functions from the time domain to the frequency domain.

◇ **Exercise Intro.1:** Perform similar tests with simple periodic functions and show that the inner product of two such functions that have different frequencies results in a graph where the red dots are all on the x axis and the curve connecting them is the straight horizontal line $y = 0$.

Euler's Identity and Formula



It has been called "The most beautiful math expression," "The most remarkable formula in Mathematics," and has received much other praise. This short expression, known as Euler's identity, connects five fundamental mathematical constants 0, 1, π, e, and i, as well as addition, exponentiation, and equality. The identity is $e^{i\pi} + 1 = 0$ (the blue dot in the figure), which is a special case of the more general formula $e^{it} = \cos(t) + i\sin(t)$. The general formula provides a connection between complex exponents, the trigonometric functions sin and cos, and circular motion. This is because the absolute value of the complex number e^{it} is $\sqrt{\cos^2 t + \sin^2 t} = 1$ for any t. The figure here

```
ClearAll[c1, fi, tb, g1, g2]
c1[t_] := 0.5 Cos[2Pi 2t + Pi/6] + 0.4 Cos[2. Pi 3 t + Pi/3];
Plot[c1[t], {t, 0, 2 Pi}]
fi[ph_] := Integrate[c1[t] Cos[2Pi 3t + ph], {t, 0, 2 Pi}];
tb = Table[{ph, fi[ph]}, {ph, 0, 2 Pi, Pi/12}]
g1 = ListLinePlot[tb, InterpolationOrder -> 3];
g2 = ListPlot[tb, PlotStyle -> {Red, PointSize[0.015]}];
tickMarks = Table[i, {i, Pi/6, 2 Pi, Pi/3}];
Show[{g1, g2}, Ticks -> {tickMarks, Automatic}]
```

$0, 0.673544$	$\pi/12, 0.922918$	$\pi/6, 1.1094$	$\pi/4, 1.22027$
$\pi/3, 1.24799$	$(5\pi)/12, 1.19065$	$\pi/2, 1.05218$	$(7\pi)/12, 0.842$
$(2\pi)/3, 0.574441$	$(3\pi)/4, 0.267735$	$(5\pi)/6, -0.0572174$	$(11\pi)/12, -0.37827$
$\pi, -0.673544$	$(13\pi)/12, -0.922918$	$(7\pi)/6, -1.1094$	$(5\pi)/4, -1.22027$
$(4\pi)/3, -1.24799$	$(17\pi)/12, -1.19065$	$(3\pi)/2, -1.05218$	$(19\pi)/12, -0.842$
$(5\pi)/3, -0.574441$	$(7\pi)/4, -0.267735$	$(11\pi)/6, 0.0572174$	$(23\pi)/12, 0.37827$
$2\pi, 0.673544$			

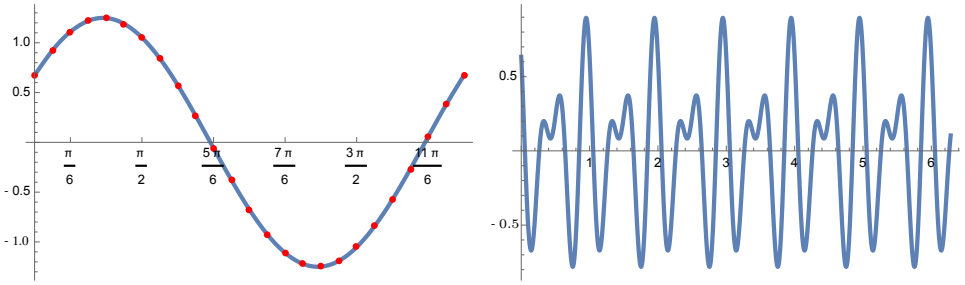


Figure Intro.3: Relation between Fourier Methods and Inner Products: An Example.

is an Argand diagram where the real and imaginary axes are labeled $\cos\theta$ and $\sin\theta$, respectively. Any point on the radius-1 circle has coordinates $a + ib$, and the red point in the figure has coordinates $(\cos\theta, \sin\theta)$. Thus, Euler's formula tells us that this sum equals $e^{i\theta}$.

The absolute value of e^{ik} is 1, which is why this term corresponds to a point on the radius-1 circle. Similarly the term Ae^{ik} corresponds to a point at distance A from the center. This is important because the term Ae^{ik} appears often in Fourier theory, where A and k correspond to the amplitude and phase of a sinusoid, respectively. Here are some identities that derive from Euler's formula

$$e^{ix} = \cos(x) + i\sin(x), \quad e^{-ix} = \cos(x) - i\sin(x),$$
$$\cos(x) = \frac{e^{ix} + e^{-ix}}{2}, \quad \sin(x) = \frac{e^{ix} - e^{-ix}}{2i}.$$

Here is a short and elegant derivation of Euler's formula. We start with the infinite sum representation of e, derived by the Taylor series:

$$e \overset{\text{taylor}}{=} \sum_{k=0}^{\infty} \frac{1}{k!},$$

from which we immediately get

$$e^x = \sum_{k=0}^{\infty} \frac{x^k}{k!}.$$

Next, we include i in the exponential to end up with

$$\begin{aligned}
e^{ix} &= \sum_{k=0}^{\infty} \frac{(ix)^k}{k!} \\
&= 1 + \frac{ix}{1!} + \frac{(ix)^2}{2!} + \frac{(ix)^3}{3!} + \frac{(ix)^4}{4!} + \frac{(ix)^5}{5!} + \frac{(ix)^6}{6!} + \dots \\
&= 1 + \frac{ix}{1!} - \frac{x^2}{2!} - \frac{ix^3}{3!} + \frac{x^4}{4!} + \frac{ix^5}{5!} - \frac{x^6}{6!} - \frac{ix^7}{7!} + \frac{x^8}{8!} \dots \\
&= \left[1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \frac{x^8}{8!} - \dots\right] + \left[\frac{ix}{1!} - \frac{ix^3}{3!} + \frac{ix^5}{5!} - \frac{ix^7}{7!} + \dots\right] \\
&= \cos x + i \sin x.
\end{aligned}$$

> Like a Shakespearean sonnet that captures the very essence of love, or a painting that brings out the beauty of the human form that is far more than just skin deep, Euler's equation reaches down into the very depths of existence.
>
> —Keith Devlin

The nth Roots of Unity

I assume that the reader is familiar with the basics of complex numbers, but the roots of unity are discussed here because they play an important role in the FFT algorithm (see Page 117).

The English word "unity" is typically defined in a dictionary as "the quality or state of being made one (unification)." In mathematics, the nth root of unity is any number z such that $z^n = 1$ for a positive integer n. However, it is easier to understand this concept by considering the equivalent definition which says: the nth roots of unity are the roots of the polynomials of the form $x^n - 1$. For $n = 2$, we factor the equation $x^2 - 1 = 0$ into $(x + 1)(x - 1) = 0$ to immediately obtain the two roots $+1$ and -1. However, for larger n, things quickly get more complicated. For $n = 4$, the roots of unity are the solutions to $x^4 - 1 = 0$, so we factor $(x + 1)(x - 1)(x^2 + 1)$ to immediately obtain the three roots 1, -1, and i. A little thinking may convince the reader that $-i$ is also a solution because $(-i)^2 = i^2 = -1$.

It turns out that most roots of unity are complex numbers, and are governed by the fundamental theorem of algebra. This important statement proves that every nth-degree polynomial has exactly n complex roots (although some of them may have a zero

imaginary part and may therefore be real). As a result, we conclude that the equation $x^n = 1$ has n complex solutions, which are nth roots of unity and also constitute ALL the nth roots of unity. Figure Intro.4 shows examples of the roots for several values of n.

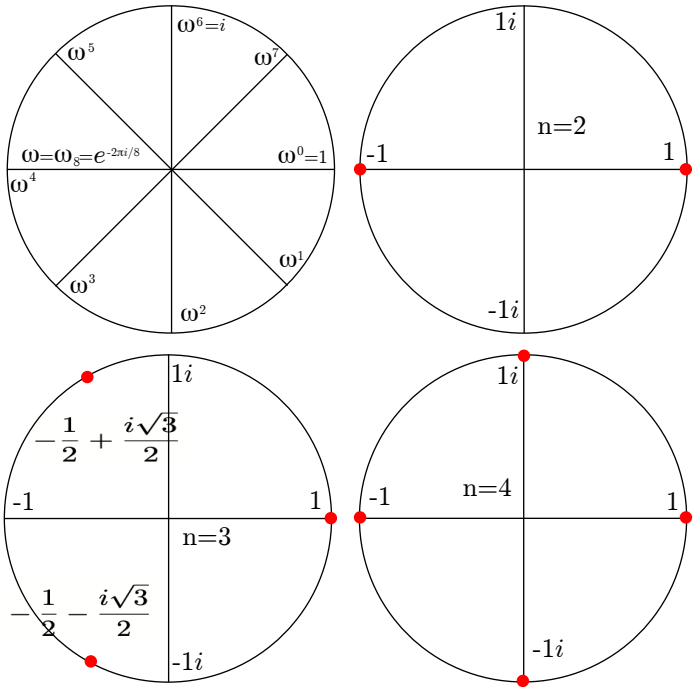


Figure Intro.4: Several Roots of Unity.

The figure suggests that the roots are uniformly distributed over the unit circle (even if they don't form opposite pairs) and that the complex number $1 + 0i$ is always a root. The angle between consecutive roots on the circle is $2\pi/n$, so we define $\omega = e^{2\pi i/n}$ and spread the roots over the circle at points that correspond to powers of ω. Another interesting algebraic property of the roots is that their sum is zero for any n.

◇ **Exercise Intro.2:** What are the roots of unity for $n = 8$?

A good introduction is half the battle.
—Proverb

1
Fourier Methods

The terms "Fourier methods" and "Fourier analysis" refer to two different mathematical techniques, namely Fourier series and the Fourier transform. The former is used to break up a periodic function into a sum of sinusoids, while the latter is used to compute the frequency components of an arbitrary, aperiodic function or of a sequence of numbers. Knowing the frequency components of a function is very similar to breaking it up into its component sinusoids, which is why the Fourier transform, Chapter 4, can be viewed as the limit of the Fourier series of a function when its period approaches infinity.

1.1 Basic Terms

Among many other topics, mathematics deals with real functions. The domain and range (input and output, respectively) of a function may be continuous or not, and they often consist of infinitely many points. In engineering applications, which are practical and are based on experiments and measurements, it is common to have as the domain of a function a finite set of input values, often sampled from an infinite set of values. Examples of such discrete sets of inputs are (1) a sound wave that is sampled (digitized) and is converted to many audio samples, and (2) a painting that it digitized and is converted to pixels. In the one-dimensional case, such a finite set of input values is referred to as a *time signal*, while in the two-dimensional case it is a spatial signal. This is why Fourier methods have two varieties, one for continuous functions and one for discrete data. The following paragraphs offer more detail,

- The technique of Fourier series is used to decompose an arbitrary periodic function into a (usually infinite) linear combination of sinusoids with different frequencies and amplitudes. This technique is based on the orthogonality of the trigonometric functions sine and cosine, and is referred to as harmonic analysis. Intuitively, it is not obvious

that any periodic function, especially one with straight segments and sharp corners, can be decomposed in this way, which is why Fourier's work comes as a surprise.

The Fourier transform is a mathematical operation that breaks a time-signal $x(t)$ into its constituent frequencies and ends up with a function $X(\omega)$ that indicates the strength of each frequency ω in the original signal. This operation transforms the original signal, which can be periodic or aperiodic, from the time domain, where it looks familiar, to the frequency domain, where we can examine the frequencies that constitute it and can also edit them in order to improve the signal in various ways. The inverse operation also exists. Given a function $X(\omega)$, it is possible to compute its inverse Fourier transform and end up with the corresponding time-signal $x(t)$. These concepts and techniques apply also to two-dimensional, spatial signals.

The Fourier series has to do with waves, so we start with a simple definition of this important term.

A wave is a disturbance that travels through space and carries with it energy but not matter. If the wave propagates through matter, it undulates or disturbs that matter; the wave moves it back and forth, periodically and temporarily. If the disturbance created by the wave undulates in the direction of the wave, as in sound waves through air or water, then the wave is longitudinal. If the disturbance is perpendicular to the direction of the wave, as in ocean waves or electromagnetic waves, the wave is transverse.

The case where an ocean wave crashes on the beach is different and it does involve moving matter, but as long as a wave moves in the sea, the water is only disturbed vertically and in a periodic fashion.

Electromagnetic waves are transverse. What undulates in those waves is the electric and magnetic fields it carries and they vary in directions that are perpendicular to the direction of the wave. When we are interested in the energy carried by an electromagnetic wave, it is best to consider the wave a stream of photons, massless particles that move at the speed of light and carry both energy and momentum.

We also need the concept of a mathematical function. This is simply a rule of calculation. Given a quantity x, the function $y = f(x)$ tells how to obtain quantity y from x by means of calculations. A simple example is $y = x^2 + 1$. Given a number x, square it and add 1 to obtain the result y. This can be done for any number x, and can therefore be illustrated graphically as a two-dimensional curve where each point has the two coordinates $(x, x^2 + 1)$. Clearly, there can be infinitely many functions and curves, with all kinds of shapes, which is why the Fourier series comes as a surprise. It says that any function can be expressed as the sum of the trigonometric functions sine and cosine; a very unintuitive claim.

A mathematical function may be simple or complicated, continuous or discontinuous. It may be defined over a narrow or a wide interval of x values (its domain), but in practice, we can obtain, measure, or compute the values of a function in only a finite number of x values. Thus, functions used in engineering are often discrete; they are known only at a finite number of points and require the discrete version of the Fourier transform, Section 4.4, to compute their frequency components.

> L'étude profonde de la nature est la source la plus féconde de découvertes mathématiques.
>
> The deep study of nature is the most fertile source of mathematical discoveries.
>
> —Joseph Fourier.

The Trigonometric Sine and Cosine Functions. The term sine comes from the Latin sinus, meaning bay, but historians tell us that it came to Latin from Arabic, which in turn had inherited it from Sanskrit.

The well-known sine and cosine functions are the building blocks of the Fourier series, but they originate in trigonometry (the study of triangles and especially right-angled, orthogonal triangles). Ancient mathematicians and astronomers have long ago noted that the ratio of the lengths of two sides of a right-angled triangle depends only on one acute angle of the triangle. Figure 1.1 shows how the two important sin and cos functions are defined geometrically as ratios of triangle sides divided by the length of the hypotenuse. In the special case where the hypotenuse equals 1, those functions equal the lengths of the two right-angled triangle sides. As the angle θ varies from 0 to 360° (or 2π radians), the graph of the sine function becomes the familiar wave, and that of the cosine function is identical in shape but lags 90° behind.

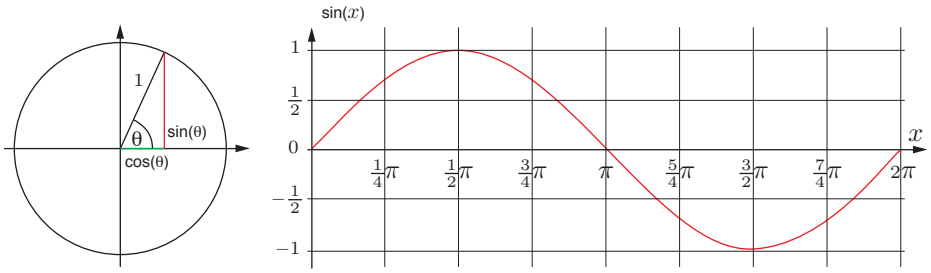


Figure 1.1: Definitions of the sin and cos Functions.

Thus, $\sin(0) = 0$, $\sin(\pi) = 1$, and $\sin(2\pi) = 0$, which implies that the function $\sin(2\pi t)$ goes through 360° when t varies from 0 to 1. Moreover, the similar function $\sin(2\pi kt)$ goes through this range k times while t varies from 0 to 1. The parameter k can therefore be considered the frequency of the sin (or cos) function. In engineering applications, the parameter t often stands for time, in which case the units of the frequency k are Hertz (Hz, the number of periods of the function per second) and the units of t are seconds.

The well-known Euler's formula $e^{it} = \cos(t) + i\sin(t)$ provides a connection between complex exponents, the trigonometric functions sin and cos, and circular motion. This is because the absolute value of the complex number e^{it} is $\sqrt{\cos^2(t) + \sin^2(t)} = 1$ for any t.

◇ **Exercise 1.1:** Why do we use the trigonometric functions sine and cosine to represent waves and not the more familiar (and perfectly symmetric) circle?

◇ **Exercise 1.2:** What is the respiratory rate of an adult expressed in Hz?

It is true that M. Fourier had the opinion that the principal end of mathematics was the public utility and the explanation of natural phenomena.

—Carl Gustav Jacob Jacobi.

2
Sine Waves

Fourier methods are concerned with the relation between functions (periodic and aperiodic) and sine waves, which is why this chapter discusses those waves, in dimensions from one to three.

2.1 One-Dimensional Sine Waves

We know how to break up a vector into its components, so it is natural to ask how to break up a complex wave into components that are simple waves. Exercise 1.1 explains why we use sinusoids and not circles in Fourier series and Fourier transform, so we start with the one-dimensional sine wave. In engineering applications, many signals of interest vary with time, which is why the basic form of a sine wave $y(t)$ that varies with the time t is often written as

$$y(t) = A\sin(2\pi ft + \phi) = A\sin(\omega t + \phi),$$

where:

- The amplitude A is the maximum distance of $y(t)$ from zero. More accurately, the amplitude is half the distance between the peak and trough of the wave, because a sinusoid does not have to be centered vertically about the x-axis and can be shifted up or down. Some texts define the amplitude $A(t)$ of a signal $y(t)$ as its instantaneous height $y(t)$ at time t. In such a case, the maximum amplitude is referred to as peak amplitude.

- The frequency f is the number of cycles of $y(t)$ per time unit. It is often measured in Hertz (Hz), but can also be expressed in radians, where 2π radians equal a complete cycle.

- The angular frequency $\omega = 2\pi f$ is the argument of f, measured in radians per time unit.

- The phase ϕ describes how much the entire wave is shifted horizontally by ϕ/ω time units. We can think of the phase as where in its cycle the oscillation is at $t = 0$, or as the number that describes the stage of the oscillation at the origin $t = 0$. (Much like how the moon's phases tell us where the moon is in its cycle.) Thus, a phase of 180° (or π radians) is equivalent to a reflection of the sine wave about a horizontal mirror.

Two sine waves with slightly different phases look like shifted versions of each other, which is why we tend to use terms such as positive phase or negative phase. However, a small positive phase is indistinguishable from a large negative phase (their absolute values always add up to $\pi/2$), which is why the correct units of a phase are circular. A phase of 90° (a quarter of a circle in a clockwise direction) is the same as that of 270° (three quarters of a circle in the opposite direction). In fact, the cosine is a sine wave with a $90° = \pi/2$ phase.

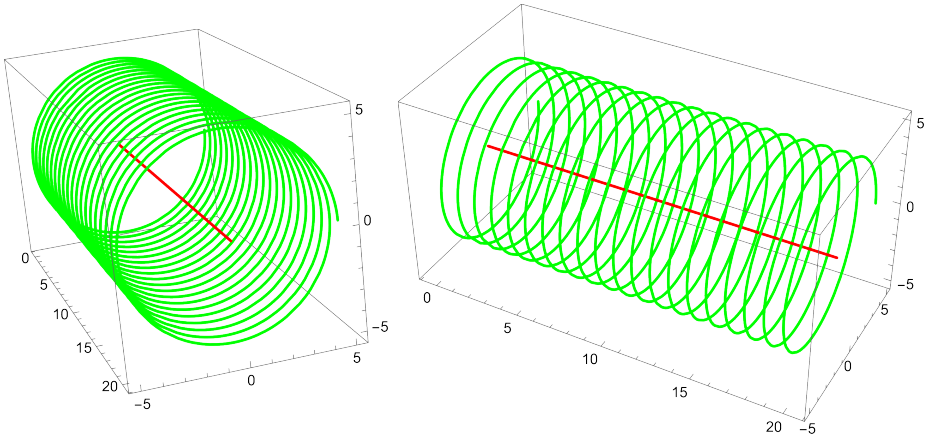
Sine and cosine are orthogonal as demonstrated by Equation (3.2). Two functions are orthogonal over an interval $[a, b]$ if their inner product (f_i, f_j), defined as the integral $\int_a^b f_i(x)f_j(x)dx$, is zero for $i \neq j$.

Sine waves are often real-valued functions, but sinusoids in general can also be complex. Much as we take a real number x and make it imaginary by multiplying it by $i = \sqrt{-1}$, we can take a real-valued $\sin(k)$ and multiply it to become the complex sine function $i\sin(k)$. Also, in the same way that a complex number consists of two real numbers, a complex sine wave consists of two real-valued sines with the same amplitude and frequency and with a certain phase difference. Experience suggests that the best phase difference is 90° or $\pi/2$, because this produces a complex sine wave whose components are the familiar real-valued sine and cosine. Thus, we end up with the sum $A\cos(k) + iA\sin(k)$, which according to Euler's formula, equals Ae^{ik}. For the purposes of Fourier series and transforms, we normally replace k with $2\pi ft$. See also the discussion of complex waves on Page 49. Because of its two real components, the best way to plot a complex sine curve (Figure 2.1) is as a three-dimensional curve, where the three dimensions are (1, 2) the real and imaginary parts of the curve, and (3) the time.

We start by looking at the sum of several sinusoids, and then trying to break this sum into its basic components. Part 1 of Figure 2.2 is the sum of three sine waves with frequencies 1, 2, and 4 Hz, where the last one has a phase of $\pi/4$. The Mathematica code for this figure is also listed. Notice that the specification `72*5` indicates 5-inch-wide plots.

The sum of the three sine waves is not a sine wave but it is periodic, as indicated by the red dot in the figure. This suggests the following approach to analyzing and transforming a complex wave structure. Given a wave form (a periodic function) w, it can perhaps be decomposed into basic components C_f that are sinusoids with increasing frequencies f and with appropriate amplitudes and phases, cleverly chosen so as to make the sum identical, or very close, to the original w. If w is very different from a smooth wave, if it has straight segments and sharp corners, the sum of the C_f's may have to be infinite, but it may converge to w in the limit.

Part 2 of Figure 2.2, together with its Mathematica code, Figure 2.3, illustrate such a process. Part d shows our target curve in blue with a yellow sine wave of frequency 1




```
amp = 5; f = 1; phs = 0;
csin[t_] := amp Exp[I (-2 Pi  f t) + phs];
ParametricPlot3D[{{t, Re[csin[t]], Im[csin[t]] }, {t, 0, 0 }},
 {t, 0, 20}, PlotStyle -> {Green, Red, Thickness[0.02]}]
```

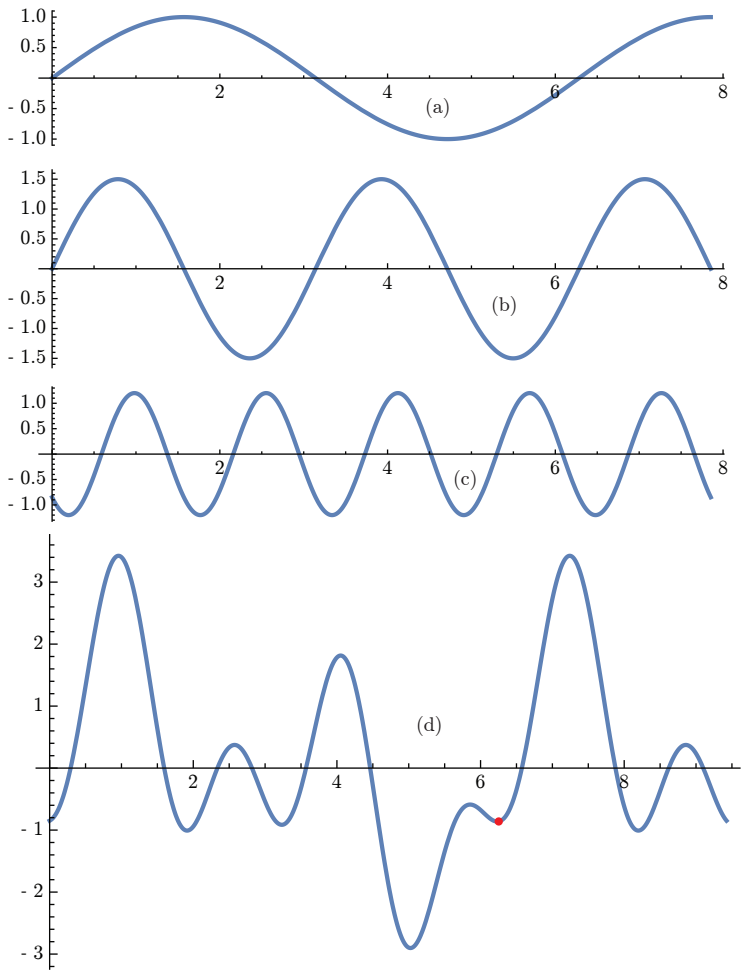
Figure 2.1: A Complex Sine Curve and Mathematica Code

as the start of the reconstruction. A blue sine wave of frequency 2, similar to that of part b, is added to the frequency-1 curve to obtain the approximation shown in yellow in part f. This is already close to our goal, part d, so we try to add a sine wave of frequency 3 (part g), to obtain part h. The wave of part h seems a better approximation, so we try to add a sine wave of frequency 4 (part i, similar to part c and with the same phase) to obtain an even closer approximation, but the resulting part j is a disappointment. It gets closer to part d in some places but moves away from it in other places. We know that our target wave, part d, was obtained by adding sine waves of frequencies 1, 2, and 4, so our conclusion is obvious; the sine wave of frequency 3 (part g), is unnecessary. The final lesson from this simple example is that given a periodic wave w, there is a chance that a sum of sine waves C_f of increasing frequencies f may converge to w, but we have to choose the amplitudes and phases of those waves carefully, and some of the C_f waves should be skipped by setting their amplitudes to zero.

Encouraged by the above example, we try to generalize it to any periodic functions. Given a periodic function $f(t)$ with period 2π, we first try to write it as the infinite sum of sine waves, each with its own amplitude and phase, and with increasing frequencies

$$f(t) = \sum_{n=0}^{\infty} b_n \sin(nt + p_n).$$

Deriving the expressions of the amplitudes b_n, however, is easier than deriving those of the phases p_n, so we try instead an infinite sum of pairs of sine and cosine, each with




```
Clear;(*Add 1D sine waves*)
SetOptions[Plot, BaseStyle -> {FontFamily -> "cmr10", FontSize -> 10}];
pp = Plot[Sin[t], {t, 0, 2.5 Pi}, AspectRatio -> 32/162, ImageSize -> 72*5]
ImageDimensions[pp]
ImageAspectRatio[pp]
Plot[1.5 Sin[2 t], {t, 0, 2.5 Pi}, AspectRatio -> 47/162, ImageSize -> 72*5]
Plot[-1.2 Sin[4t+Pi/4], {t, 0, 2.5 Pi}, AspectRatio->32/162, ImageSize->72*5]
Plot[Sin[t]+1.5 Sin[2 t]-1.2 Sin[4 t+Pi/4], {t, 0, 2.5 Pi}, ImageSize->72*5]
(*Export["/Users/davidsalomon/Desktop/five.pdf",%]*)
```

Figure 2.2: Part 1, Adding Sine Waves.

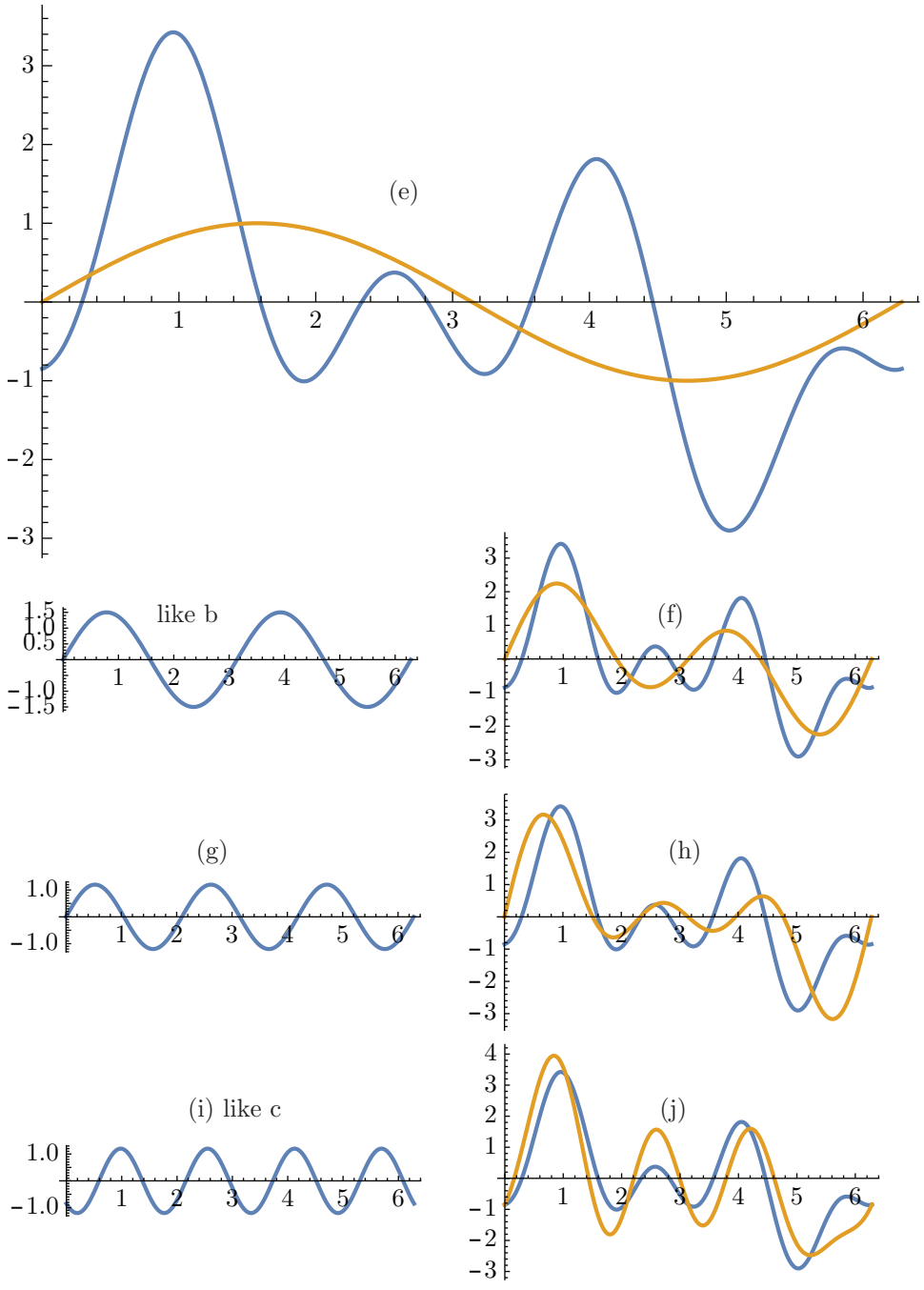


Figure 2.2: Part 2, Reconstructing a Wave From Sinusoids.

```
Clear; (* Reconstruct 1D sine waves *)
SetOptions[Plot,
 BaseStyle -> {FontFamily -> "Times New Roman", FontSize -> 10}];
pp = Plot[{Sin[t] + 1.5 Sin[2 t] - 1.2 Sin[4 t + Pi/4], Sin[t]}, {t, 0, 2 Pi}]
ImageDimensions[pp]
ImageAspectRatio[pp]
Plot[1.5 Sin[2 t], {t, 0, 2 Pi}, AspectRatio -> 47/162, ImageSize -> 72*2.25]
Plot[{Sin[t] + 1.5 Sin[2 t] - 1.2 Sin[4 t + Pi/4],
  Sin[t] + 1.5 Sin[2 t]}, {t, 0, 2 Pi}, ImageSize -> 72*2.25]
Plot[1.2 Sin[3 t], {t, 0, 2 Pi}, AspectRatio -> 32/162, ImageSize -> 72*2.25]
Plot[{Sin[t] + 1.5 Sin[2 t] - 1.2 Sin[4 t + Pi/4],
  Sin[t] + 1.5 Sin[2 t] + 1.2 Sin[3 t]}, {t, 0, 2 Pi}, ImageSize -> 72*2.25]
Plot[-1.2 Sin[4t+Pi/4], {t, 0, 2 Pi}, AspectRatio -> 32/162, ImageSize->72*2.25]
Plot[{Sin[t] + 1.5 Sin[2 t] - 1.2 Sin[4 t + Pi/4],
  Sin[t] + 1.5 Sin[2 t] + 1.2 Sin[3 t] - 1.2 Sin[4 t + Pi/4]},
 {t, 0, 2 Pi}, ImageSize -> 72*2.25]
Export["/Users/davidsalomon/Desktop/five.pdf", %]
```

Figure 2.3: Mathematica Code for Figure 2.2, part 2.

its own amplitude and with zero phase

$$\begin{aligned}
&\sum_{n=0}^{\infty} [a_n \cos(nt) + b_n \sin(nt)] \\
&= \sum_{n=0}^{\infty} a_n \cos(nt) + \sum_{n=0}^{\infty} b_n \sin(nt) \\
&= a_0 + \sum_{n=1}^{\infty} a_n \cos(nt) + \sum_{n=1}^{\infty} b_n \sin(nt). \qquad (2.1)
\end{aligned}$$

(Notice that the first term of Equation (2.1), for $n = 0$, is simply a_0, which is why it makes sense to write this equation as a_0 plus the two sums that now start from $n = 1$.)

The term a_0 is sometimes referred to as the DC component of the Fourier series.

This technique of eliminating the phases is based on the fact that a sum of the form $a\cos(t) + b\sin(t)$ is a sinusoid with frequency t and both an amplitude and a phase that are determined by a and b. It is easy to see from Figure 1.1 that the amplitude of the sum is $\sqrt{a^2 + b^2}$ and its phase is $\arctan(b/a)$. Notice that there are always two phases, often located on opposite sides of $t = 0$, and their absolute values add up to $\pi/2$.

2.2 Two-Dimensional Sine Waves

Section 2.1 introduces one-dimensional sine waves and their sums. This is followed by a discussion of the one-dimensional Fourier series. Similarly, before we discuss the two-dimensional discrete Fourier transform, this section illustrates and explains two-dimensional sine waves and their sums. Figure 2.4 displays several such waves and lists the Mathematica code used to compute them. The figure makes clear that periodicity exists also in two dimensions, where it is a vector quantity because it has both magnitude and direction.

Graphically, a two-dimensional sine wave resembles an unrolled cylinder or a corrugated plastic/metal sheet whose four edges (either two or all four) are one-dimensional sine waves each. Mathematically, such a wave is a function of two parameters k and l that determine its frequency and are known as the Miller indices. They are borrowed from a notation system in crystallography for lattice planes in crystal (Bravais) lattices.

$$y_{k,l}(x,y) = A\sin\bigl(2\pi(kx+ly)+\phi\bigr), \tag{2.2}$$

These waves are referred to in this document as two-dimensional because (1) each depends on two frequency parameters, and (2) they are used in the two-dimensional Fourier transform. However, mathematically they should be termed three-dimensional because they are surfaces and each point in them has the three coordinates $(x, y, \sin(kx+ly))$.

The three waves at the top row of Figure 2.4 correspond to parameter pairs $(0,1)$, $(0,2)$, and $(-1,5)$. The first is therefore flat in the x direction and features one cycle in the interval $[0,2\pi]$ in the y direction. The second wave is also flat in x, but has two cycles in the same interval in the y direction. The third wave features one cycle in the x direction, but going in the "opposite" direction, and five cycles in the y direction. The waves of the second row are similar combinations of the parameters, but the waves on the bottom row are each a sum of two two-dimensional sine waves. These are produced by function `d2sinep` which has four frequency parameters and their shapes are much more intricate.

2.3 Three-Dimensional Sine Waves

The one-dimensional Fourier transform (FT) is used to analyze one-dimensional time series. Such a series can be graphed as a curve, which is why the one-dimensional FT employs one-dimensional sine waves, i.e., curves, as analyzing functions. Similarly, the two-dimensional FT deals with two-dimensional objects, images, which is why it employs two-dimensional sine waves, which are images, as analyzing functions. Finally, three-dimensional Fourier transforms are used to analyze three-dimensional objects, solids, which why the three-dimensional sine waves are also solids.

Three-dimensional Fourier transforms are used in fields such as 3D microscopy, X-ray crystallography, and 3D NMR.

The equation of a three-dimensional sine wave is an extension of the two-dimensional case, Equation (2.2)

$$y_{h,k,l}(x,y,z) = A\sin\bigl(2\pi(hx+ky+lz)+\phi\bigr), \tag{2.3}$$

```
Clear["Global`*"]
phs = 0;
d2sine[l_, k_] := DensityPlot[Sin[l x + k y + phs], {x, 0, 2 Pi}, {y, 0, 2 Pi},
   ColorFunction -> "GrayTones"];
d2sinep[l_, k_, m_, n_] := DensityPlot[Sin[l x + k y] + Sin[m x + n y],
 {x, 0, 2 Pi}, {y, 0, 2 Pi}, ColorFunction -> "GrayTones"];
GraphicsGrid[
 {{d2sine[0, 1], d2sine[0, 2], d2sine[-1, 5]},
 {d2sine[1, 1], d2sine[2, 1], d2sine[3, -2]},
 {d2sinep[0, 1, 1, 1], d2sinep[0, 3, 2, 1], d2sinep[-1, 5, 3, -2]}}
           ]
```

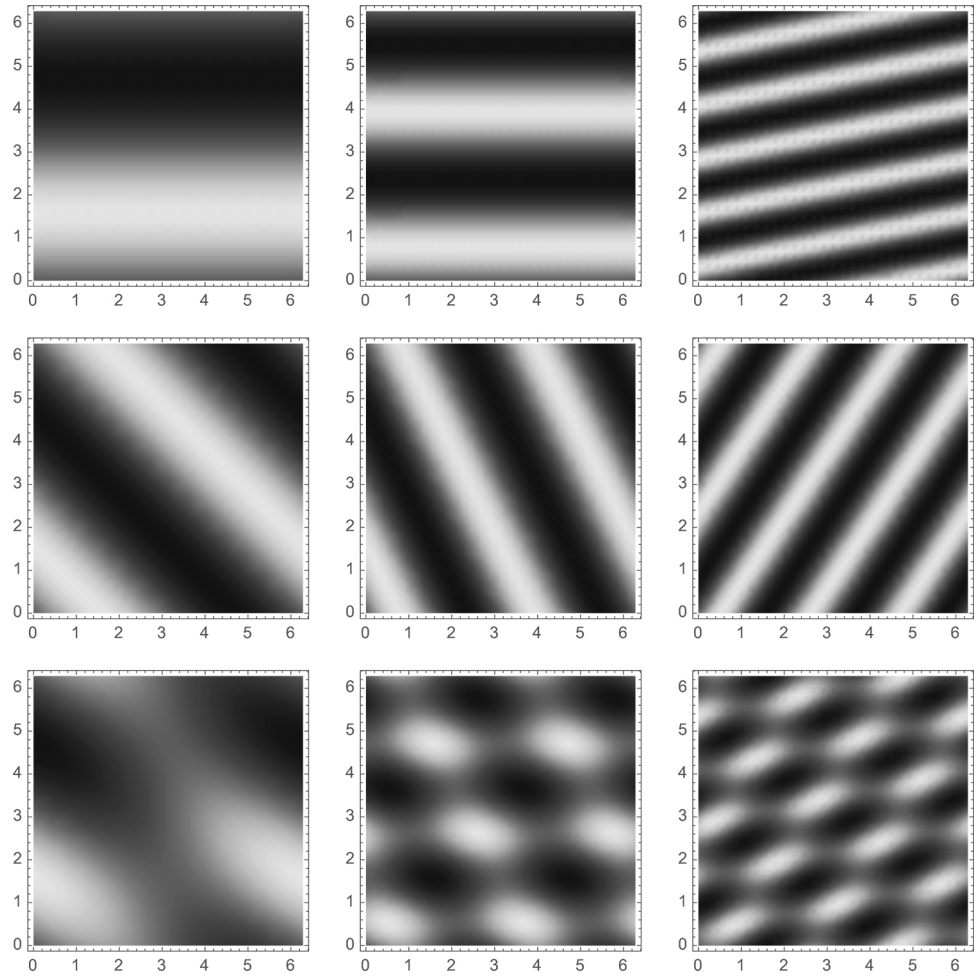


Figure 2.4: Two-Dimensional Sine Waves.

Figure 2.5 illustrates two examples. Part (a) of the figure shows a wave propagating in the y direction. Four cuts through this wave are shown, with values (from bottom to top) of 0, 1, 0, and -1. These are four of infinitely many cuts whose values vary continuously according to the sine wave. As we move in the y direction from bottom to top, the cuts perpendicular to our direction (i.e., in the xz plane) have values that vary with the wave. The entire wave is solid and is not not limited to the square box shown in the figure.

Part (b) similarly shows a 3D sine wave moving in the xy plane. Each cut shown has a value that depends on the mixture of sine waves in the x and y directions.

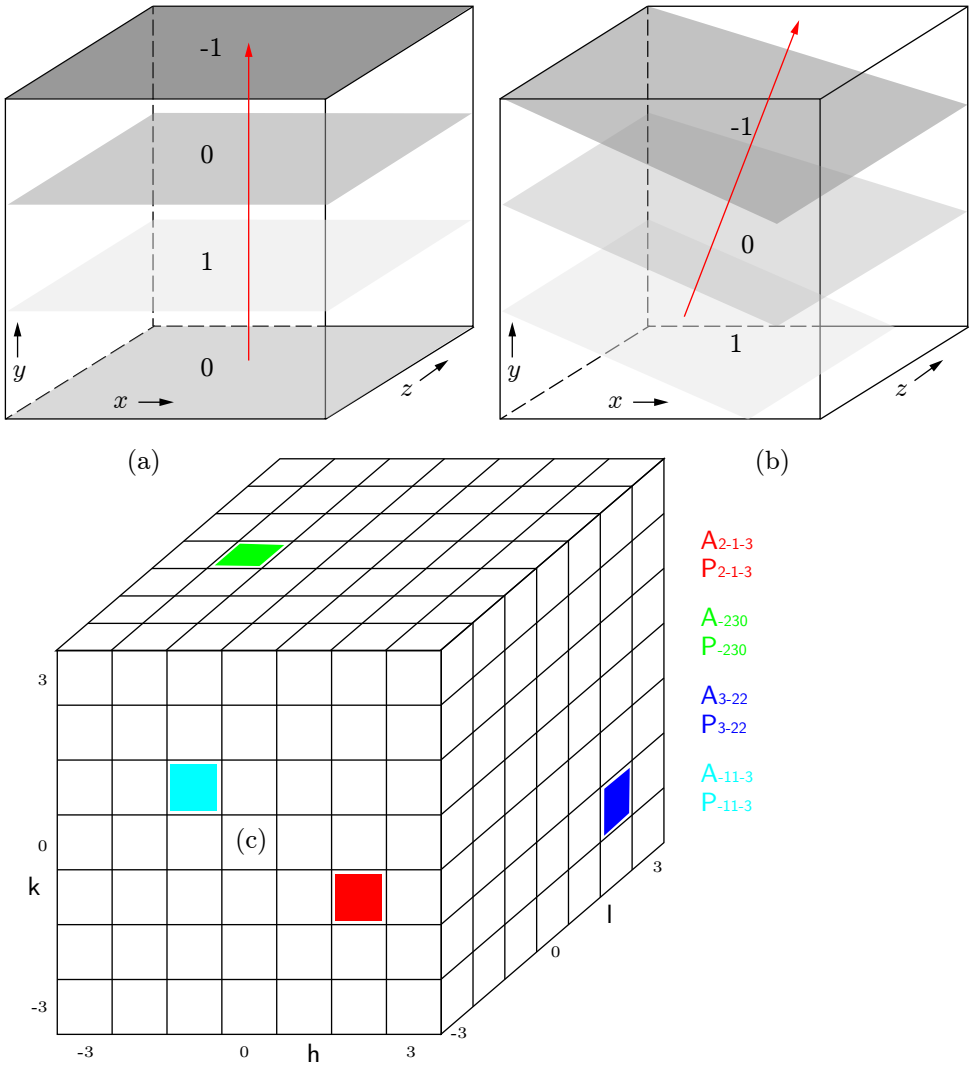


Figure 2.5: 3D Sine Waves.

Part (c) of the figure is an example of a discrete 3D cube where each of the $7 \times 7 \times 7 = 343$ small cubes is identified by three Miller indices `h`, `k`, and `l`, which vary from -3 to 3. When a Fourier transform is performed on a discrete 3D set of data values, each small cube receives an amplitude A_{hkl} and a phase P_{hkl} according to its position in the cube. This is similar to the 2D Fourier transform, whose results become two 2D arrays of amplitude and phase, similar to the ones listed in Figure 4.32. The Miller indices correspond to the frequencies of the transform components, and each component can therefore be fully constructed by its indices (the frequency), amplitude, and phase. The sum of all the 3D transform components (each a 3D sine wave) fully reconstructs the original 3D data.

Notice that the wave of Equation (2.3) is different from the complex sine wave of Figure 2.1 because the latter is a two–dimensional wave plotted over time, while the former describes a solid object whose color or grayscale varies periodically as a sine in a certain direction.

If you saw a sine wave, would you wave back?
—Steven Wright (paraphrased)

3
Fourier Series

The vast field of Fourier methods or Fourier analysis is divided into two main areas, Fourier series and the Fourier transform. The former is concerned with breaking up a periodic function into basic components which are pure sine and cosine waves, as shown in Equation (2.1), duplicated here. The latter transforms an aperiodic function, often a function of time, into its frequency components which indicate the strength of each frequency in the original function.

$$\begin{aligned}\sum_{n=0}^{\infty}[a_n\cos(nt)+b_n\sin(nt)] &= \sum_{n=0}^{\infty}a_n\cos(nt)+\sum_{n=0}^{\infty}b_n\sin(nt)\\ &= a_0+\sum_{n=1}^{\infty}a_n\cos(nt)+\sum_{n=1}^{\infty}b_n\sin(nt).\end{aligned}\tag{2.1}$$

Equation (2.1) suggests an approach to Fourier series, and the rest of this chapter discusses ways to derive the coefficients of this equation. Two methods are proposed, that end up with the same expressions for the coefficients. The chapter also inlcudes examples of Fourier series.

3.1 The Role of Complex Numbers

The continuous Fourier series, Equation (3.1), is a sum of sines and cosines. As explained in Section 2.1 and Page 18, such a sum eliminates the need to compute phases and requires only the amplitudes of the individual series components. In practice, however, engineers prefer a different representation for both the continuous and discrete versions of the Fourier series, one that uses complex numbers. The following are the main reasons for this approach:

- A single complex number can express both an amplitude and a phase.
- If we compute the Fourier series of a function with only real-valued sines or real-valued cosines, the result would depend on the phase offset between the real-valued waves and the function. If the function is shifted by even a small amount, the series would produce a different set of sinusoids.

For the one-dimensional continuous Fourier series, the complex-number representation is the sum

$$x(t) = \sum_{k=-\infty}^{\infty} C_k e^{ik\omega_0 t},$$

where the coefficients C_k are given by

$$C_k = \begin{cases} \frac{1}{T_0}\int_{T_0} x(t)dt, & k = 0 \\ \frac{1}{T_0}\int_{T_0} x(t)e^{-ik\omega_0 t}dt, & k \neq 0. \end{cases}$$

The subscript T_0 of the integral means an integration interval of one period. It can go from the start to the end of a period or from any point in a period to the corresponding point in the next period.

This sum (see also Equation (4.2) for the discrete case) converts a periodic function $x(t)$ of time to a set of sinusoids of higher and higher frequencies. The terms time domain and frequency domain are also commonly used. The symbol ω (omega) represents frequency. If we write, for example, $\sin(3)$, the frequency is three radians per second. However, it is common to use cycles per second, also denoted by Hz (Hertz) as the unit of frequency, which is why we use terms such as $\sin(2\pi\omega)$ or $\sin(2\pi\omega t)$. They indicate sine waves of frequencies ω Hz or ωt Hz, because there are 2π radians in a full circle.

Since we are talking about complex numbers, the symbol i stands for $\sqrt{-1}$, although electrical engineers often use j instead, because for them i also stands for electrical current.

Equation (4.1), duplicated here, makes use of a complex exponent because the derivative of this function is simple. The derivative of $e^{-i\omega t}$ with respect to ω is $-i\,t\,e^{-i\omega t}$. In order to understand the relation of (4.1) to Equation (3.1), we observe that the well-known Euler's formula satisfies $e^{-i\omega t} = \cos(-\omega t) + i\sin(-\omega t) = \cos(\omega t) - i\sin(\omega t)$.

$$F(\omega) = \int_{-\infty}^{\infty} f(t)e^{-i\omega t}dt. \tag{4.1}$$

Computer Generated Text

A Fourier series is a mathematical way of expressing a periodic function as a sum of sines and cosines. It can be used to represent any periodic function as a sum of harmonically-related sinusoids.

The Fourier series of a function $f(x)$ defined on the interval $[-L, L]$ is given by:

$$f(x) = a_0/2 + \sum_{n=1}^{\infty}[a_n \cos(n\pi/L) + b_n \sin(n\pi/L)].$$

where a_0, a_n, and b_n are the Fourier coefficients of the function. The a_0 term is called the "DC component" of the Fourier series, and the terms with $n > 0$ are called the "AC components."

The Fourier series can be used to represent a wide variety of functions, including functions that are not smooth or that have discontinuities. It is often used in signal processing, image processing, and other fields where periodic functions need to be analyzed or synthesized.

Generated by ChatGPT, 22 Dec 2022.

3.2 Deriving the Fourier Coefficients, 1 of 2

Based on the discussion above, we start by trying to represent a given periodic function $f(t)$ as a three-part sum, commonly referred to as the continuous Fourier series, of the form

$$f(t) = \frac{a_0}{2} + \sum_{n=1}^{\infty} a_n \cos(nt) + \sum_{n=1}^{\infty} b_n \sin(nt), \tag{3.1}$$

and we try to derive expressions for the coefficients a_0, a_n, and b_n such that the sum would converge to $f(t)$. We start by considering the concept of orthogonal functions and evaluating several integrals that are relevant to this task.

A set of functions $f_i(x)$ is said to be orthogonal over an interval $[a, b]$ if any two distinct functions in the set are orthogonal to each other over that interval. Those familiar with vectors know that two vectors are orthogonal if their inner product is zero. Similarly, two functions are orthogonal if their inner product, defined as the integral

$$(f_i, f_j) = \int_a^b f_i(x) f_j(x) dx$$

is zero for any $i \neq j$. (Angle brackets $\langle f_i, f_j \rangle$ are often also used.)

As the first step in deriving the Fourier coefficients, we notice that the set of functions $\{1, \cos(nx), \sin(nx)\}$ for positive integers n is orthogonal on $[0, 2\pi]$. We examine the following cases, which are computed with the help of various trigonometric identities.

$$\begin{aligned}
&\int_0^{2\pi} \sin(mt)\sin(nt)dt = \begin{cases} 0 & m \neq n \\ \pi & m = n \end{cases}, \\
&\int_0^{2\pi} \cos(mt)\cos(nt)dt = \begin{cases} 0 & m \neq n \\ \pi & m = n, \end{cases}, \\
&\int_0^{2\pi} \cos(mt)\sin(nt)dt = 0.
\end{aligned} \tag{3.2}$$

Figure 3.1 illustrates the six basic orthogonality relations of the sine and cosine. It is easy to see intuitively that in four of the six cases, the areas over and under the curves cancel each other out and add up to zero.

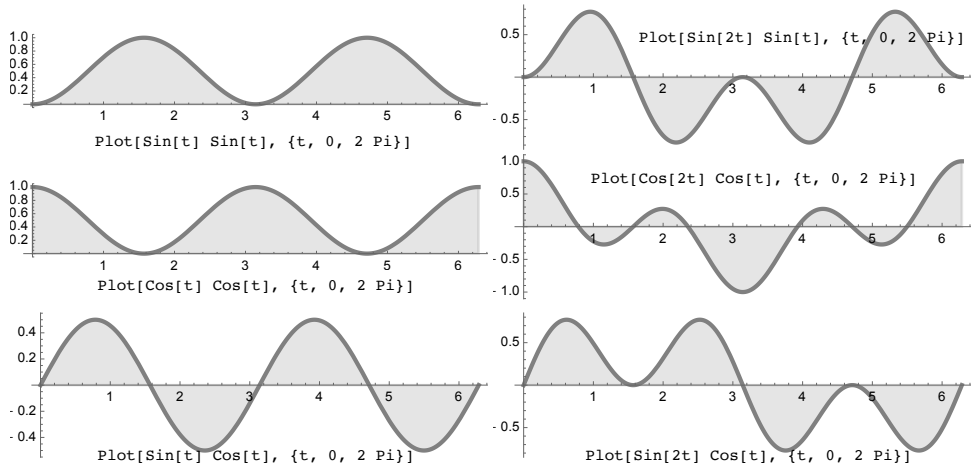


Figure 3.1: Orthogonality of the sin and cos Functions.

In the next step, we use the results above to compute the three integrals

$$\begin{aligned}
&\int_0^{2\pi} f(t)\sin(mt)dt \\
=&\int_0^{2\pi}\left[\frac{a_0}{2}+\sum_{n=1}^{\infty}a_n\cos(nt)+\sum_{n=1}^{\infty}b_n\sin(nt)\right]\sin(mt)dt \\
=&\int_0^{2\pi}\left[0+0+\sum_{n=1}^{\infty}b_n\sin(nt)\right]\sin(mt)dt \\
=&b_m\pi.
\end{aligned}$$

Which yields the expressions for the b_n coefficients

$$b_n=\frac{1}{\pi}\int_0^{2\pi} f(t)\sin(nt)dt.$$

The a_n coefficients are similarly obtained by

$$\begin{aligned}
&\int_0^{2\pi} f(t)\cos(mt)dt \\
=&\int_0^{2\pi}\left[\frac{a_0}{2}+\sum_{n=1}^{\infty}a_n\cos(nt)+\sum_{n=1}^{\infty}b_n\sin(nt)\right]\cos(mt)dt \\
=&\int_0^{2\pi}\left[0+\sum_{n=1}^{\infty}a_n\cos(nt)+0\right]\cos(mt)dt \\
=&a_m\pi.
\end{aligned}$$

Which yields the expressions for the a_n coefficients

$$a_n = \frac{1}{\pi}\int_0^{2\pi} f(t)\cos(nt)dt.$$

Finally, the a_0 coefficient is similarly derived

$$\begin{aligned}
&\int_0^{2\pi} f(t)1dt \\
=&\int_0^{2\pi}\left[\frac{a_0}{2}+\sum_{n=1}^{\infty}a_n\cos(nt)+\sum_{n=1}^{\infty}b_n\sin(nt)\right]1dt \\
=&\int_0^{2\pi}\left[\frac{a_0}{2}+0+0\right]1dt \\
=&a_0\pi.
\end{aligned}$$

Which yields the expressions for a_0

$$a_0 = \frac{1}{\pi}\int_0^{2\pi} f(t)dt.$$

Note. We obtained this particular result because we started with $a_0/2$ in our original sum instead of just a_0. Had we started with a_0 in the original sum, the expression above would have become

$$a_0 = \frac{1}{2\pi}\int_0^{2\pi} f(t)dt,$$

which is the average of $f(t)$, its integral divided by the length of its period. This value is also used in the second example below.

We can now write Equation (3.1) explicitly

$$\begin{aligned}
f(t) =&\frac{a_0}{2}+\sum_{n=1}^{\infty}a_n\cos(nt)+\sum_{n=1}^{\infty}b_n\sin(nt) \\
=&\frac{1}{2\pi}\int_0^{2\pi} f(t)dt+\frac{1}{\pi}\sum_{n=1}^{\infty}\int_0^{2\pi} f(t)\cos(nt)dt\ \cos(nt) \\
+&\frac{1}{\pi}\sum_{n=1}^{\infty}\int_0^{2\pi} f(t)\sin(nt)dt\ \sin(nt). \qquad (3.3)
\end{aligned}$$

Notice that the a_n and b_n coefficients have similar forms and each is a Fourier transform, as shown in Chapter 4. Each coefficient is an integral of the product of an analyzed signal $f(t)$ and an analyzing function which is a sinusoid. Such an integral is a continuous form of an inner product.

3.3 Deriving the Fourier Coefficients, 2 of 2

Those who were put off by the complex-looking integrals of the previous section may like the approach discussed here. The Fourier coefficients are derived here by borrowing several concepts from the field of linear algebra and applying them to our problem. The coefficients are derived in a process similar to decomposing a vector in an orthonormal basis. In essence, we try to manipulate the sine and cosine functions similar to what we do with vectors. We notice that these functions are 2π-periodic, are smooth, and are defined over the field $\mathcal{R}$ of real numbers. We also notice that the 2π-periodic functions are closed under addition and scalar multiplication. Adding such functions produces a 2π-periodic function, as also does a linear combination such as $x\sin(t) + y\cos(t)$.

In fact, the value 2π isn't special. All periodic functions with the same period T are similarly closed, and this observation suggests a connection between functions and vectors. Vectors over the real field $\mathcal{R}^n$ (i.e., vectors with n components that are real numbers) are also closed under addition and scalar multiplication and form a vector space. An important feature of a vector space is the existence of bases. A basis of such a space is a set of vectors that are linearly independent and also span the entire space. Here is what these terms mean:

- In a set of linearly independent (orthogonal) vectors, any inner product $(\mathbf{v}_i, \mathbf{v}_j)$ is zero for $i \neq j$.

- Any vector $\mathbf{v}$ in the space can be represented uniquely (in one way only) as a linear combination $\sum a_i \mathbf{b}_i$, where each of of the basis vectors $\mathbf{b}_i$ is multiplied by a weight a_i which is a real number.

The common base of $\mathcal{R}^3$ (the space of real three-dimensional vectors) is the set $\mathbf{e}_1 = (1, 0, 0)$, $\mathbf{e}_2 = (0, 1, 0)$, and $\mathbf{e}_3 = (0, 0, 1)$. These vectors are orthonormal, i.e., in addition to being orthogonal they also have unit lengths. Thus, they form an orthonormal basis where any vector (a, b, c) can be represented uniquely as the linear combination $a\,\mathbf{e}_1 + b\,\mathbf{e}_2 + c\,\mathbf{e}_3$.

Once the concepts of a vector space and its bases are grasped, it is easy to translate them to the space of smooth 2π-periodic functions $f(t)$ on $\mathcal{R}^n$. The idea is that the Fourier coefficients, the three parts of Equation (3.1), form an orthonormal basis of this space. The quantities 1, $\cos(nt)$ and $\sin(nt)$—which are multiplied by the coefficients $a_0/2$, a_n, and b_n—are linearly independent and also span the entire space of the smooth 2π-periodic functions. Equation (3.2) shows why they are linearly independent. The same equation also implies that those quantities have a "size" of one unit. All we need to do is define the inner product of two functions f and g as

$$\langle f_i, f_j \rangle = \frac{1}{\pi} \int_{-\pi}^{\pi} f(t) g(t)\, dt.$$

There is another feature of vectors that can easily be "borrowed" by functions. Given a vector space with basis vectors $\mathbf{e}_i$, any vector $\mathbf{x}$ in this space can be uniquely represented as a linear combination $\sum a_i \mathbf{e}_i$ of the basis vectors, where each coefficient a_i is the projection of vector $\mathbf{x}$ on basis vector $\mathbf{e}_i$ (the component of $\mathbf{x}$ in the $\mathbf{e}_i$ direction), However, such a projection is the inner product $(\mathbf{x}, \mathbf{e}_i)$, which is why we can write the

linear combination in the form

$$\mathbf{x} = \sum_i (\mathbf{x}, \mathbf{e}_i)\mathbf{e}_i, \tag{3.4}$$

which is very similar to the original Fourier sum of Equation (3.1). In that equation, the coefficients a_0, a_n, and b_n are inner products of (1) the function $f(t)$ that is being decomposed and (2) the set $\{1, \cos(nx), \sin(nx)\}$ of functions. We can therefore write the Fourier coefficients a_0, a_n, and b_n as the inner products

$$a_0 = \langle f(t), 1\rangle, \quad a_n = \langle f(t), \cos(nt)\rangle, \quad \text{and } b_n = \langle f(t), \sin(nt)\rangle.$$

It is as if our function $f(t)$ was a vector projected onto the orthonormal basis of vectors $\{1, \cos(nx), \sin(nx)\}$ by means of inner products. Once we agree with this statement, we can look at the Fourier series simply as a change of basis in the vector space of the smooth 2π-periodic functions. This way of looking at the series reveals an elegant connection (or parallel) between the two seemingly disparate disciplines of linear algebra and the Fourier series.

3.4 The Fourier Series and Least-Squares

The coefficients of the continuous Fourier series are derived in this section from the seemingly unrelated technique of least squares. The presentation is appropriate for mathematically-savvy readers who would like to see how very different approaches can produce the same result. General references for this section are [leastSQ (2017a)] through [leastSQ (2017d)].

Figure 3.2 illustrates the least-squares approach. Imagine that a researcher has measured the height and weight of seven people. Each pair of (height, weight) measurements is plotted as a point in the figure. Five of the points (in black) seem to suggest that taller persons tend to weigh more, while the other two points (in red) contradict this assumption, are considered outliers by the researcher, and are ignored. The green line is the result of applying the least-squares technique to the five points. This is the line that is closest to these points. In contrast, the red line may be closer to some points, but farther away from the others. The green line can now be used to decide on the ideal weight for a person whose height is given by the gray point.

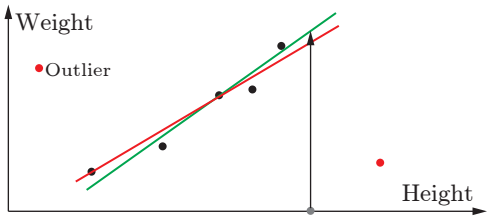


Figure 3.2: Experimental Points, Outliers, and Interpolating Lines.

Mathematically, the least-squares technique is the case where a system of equations has more equations than unknowns. In general, such a system cannot be solved, but with least-squares we can at least obtain the best case, the solution which is the closest to all the equations, even though it may not be an exact solution of any. A linear algebraic equation has the form $a_1x_1 + a_2x_2 + \cdots + a_nx_n = b$, where the a_i's are the (known) coefficients, the x_i's are the unknowns, and b is the (known) right-hand side of the equation, often referred to as the target. Using the compact matrix notation, a system of such equations is written as $Ax = b$, where A is the coefficient matrix, x is a column of the unknowns, and b is a column of all the right-hand-side terms. Each row of A contains the coefficients of an equation. Assuming that there are n unknowns and m equations, there can be three cases as follows:

- $m < n$. There are too few equations. In this case, there are many solutions. The simplest example is a system of one equation with two unknowns, such as $x + y = 0$. For every solution x, there is a solution y which is simply $-x$.
- $m = n$. If the equations are independent, there is a unique solution. We say that such a system of equations has zero degrees of freedom.
- $m > n$. Any set of n independent equations can be used to compute a solution, so there are many sets of solutions. The best that can be done is to employ least-squares and obtain the best possible set of solutions. The (m, n) coefficient matrix A has, in this case, more rows (m) than columns (n).

Here is the principle of the least-squares method. As there cannot be a unique solution to $Ax = b$, we must look for a set x of unknowns that minimizes the difference between Ax and b. We start by writing $r = b - Ax$, where r is the discrepancy vector, and try to compute a set x of unknowns that will minimize r. Since r is a vector, the square of its length is r^Tr, where the superscript T means a transpose. This is the dot product of r with itself, or the sum of the squares of the elements of r. Written explicitly, this quantity is now expanded to yield

$$\begin{aligned} r^Tr &= (b - Ax)^T(b - Ax) \\ &= (b^T - x^TA^T)(b - Ax) \\ &= x^TA^TAx - x^TA^Tb - b^TAx + b^Tb. \end{aligned}$$

Notice that the term b^Tb does not depend on x. It can therefore be ignored, since our task is to find the best value of x. Also, the terms x^TA^Tb and b^TAx are transposes of each other, and since each is a single number, they must be equal (a number is its own transpose). We can therefore write them as $-2x^TA^Tb$. This implies that

$$r^Tr = 2\left[\frac{1}{2}x^TA^TAx - x^TA^Tb\right] + b^Tb,$$

It is now clear that the only part of r^Tr that depends on x is the quadratic form (a polynomial with terms all of degree two)

$$\frac{1}{2}x^TA^TAx - x^TA^Tb.$$

We temporarily denote matrix A^TA by D and vector A^Tb by h, thereby simplifying the previous expression to

$$\frac{1}{2}x^TDx - x^Th,$$

and it is well known (see, e.g., reference [leastSQ 17e]) that such a quadratic form has a minimum when $Dx = h$, implying $x = D^{-1}h$ or, equivalently $(A^TA)x = A^Tb$. We know that A is invertible, because its rows must be linearly independent, thereby giving it the maximum rank. This is why we can finally obtain the least-squares value of x as

$$x = (A^TA)^{-1}A^Tb. \tag{3.5}$$

We now try to apply Equation (3.5) to determine the Fourier coefficients a_0, a_n, and b_n, and we do this by reducing the infinite sums of Equation (3.1) to finite sums. The Fourier problem is to represent a given periodic function $f(t)$ in terms of the simple periodic functions sine and cosine, and we are now trying to approach this problem from the direction of linear algebra and vector spaces instead of from calculus and infinite series.

The first step is to trim the infinite sums of Equation (3.1) into finite sums

$$f(t) = \frac{a_0}{2} + \sum_{n=1}^{N} a_n \cos(nt) + \sum_{n=1}^{N} b_n \sin(nt). \tag{3.6}$$

It is clear that only certain "lucky" functions $f(t)$ can be represented exactly in terms of the $2N + 1$ coefficients of Equation (3.6). In general, such a representation would be approximate, and we therefore apply the technique of least-squares to obtain the best representation. We apply Equation (3.5) but in a form which uses inner products, because our problem deals with vectors, not with matrices. We consider the quantities 1, $\sin(nt)$, and $\cos(nt)$ a small, incomplete set of basis vectors for the entire space of functions $f(t)$.

We therefore start with a large vector space of vector quantities denoted by $\mathbf{v}$. We assume that only three basis vectors $\mathbf{e}_1$, $\mathbf{e}_2$, and $\mathbf{e}_3$ are known for this space. We construct the matrix whose three columns are those basis vectors and denote it by B. Thus

$$B \stackrel{\text{def}}{=} [\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3].$$

Each vector $\mathbf{v}$ in this space can now be represented approximately as a linear combination $\alpha_1\mathbf{e}_1 + \alpha_2\mathbf{e}_2 + \alpha_3\mathbf{e}_3$ of the three basis vectors for some coefficients α_i that have to be computed by least-squares. We write the three inner products

$$\begin{aligned}
(\mathbf{v}, \mathbf{e}_1) &= \alpha_1(\mathbf{e}_1, \mathbf{e}_1) + \alpha_2(\mathbf{e}_2, \mathbf{e}_1) + \alpha_3(\mathbf{e}_3, \mathbf{e}_1),\\
(\mathbf{v}, \mathbf{e}_2) &= \alpha_1(\mathbf{e}_1, \mathbf{e}_2) + \alpha_2(\mathbf{e}_2, \mathbf{e}_2) + \alpha_3(\mathbf{e}_3, \mathbf{e}_2),\\
(\mathbf{v}, \mathbf{e}_3) &= \alpha_1(\mathbf{e}_1, \mathbf{e}_3) + \alpha_2(\mathbf{e}_2, \mathbf{e}_3) + \alpha_3(\mathbf{e}_3, \mathbf{e}_3),
\end{aligned}$$

and observe that the nine inner products $(\mathbf{e}_i, \mathbf{e}_j)$ above can be written as a matrix M that equals B^TB, implying that the inverse M^{-1} equals the inverse $(B^TB)^{-1}$, which happens

to be the first term of Equation (3.5). However, inner products are commutative, which implies that M is symmetric and thus its inverse equals its transpose.

A quick look at Equation (3.5) also shows that the quantity that is equivalent to its next term A^T is our new matrix B (transposed), and the equivalent of its last term, vector b, is a vector whose ith element is the inner product of a $\mathbf{v}$ (a general element of our vector space) and the ith basis vector $\mathbf{e}_i$. Thus, the equivalent of Equation (3.5) is

$$\begin{bmatrix} (\mathbf{e}_1,\mathbf{e}_1) & (\mathbf{e}_1,\mathbf{e}_2) & (\mathbf{e}_1,\mathbf{e}_3) \\ (\mathbf{e}_2,\mathbf{e}_1) & (\mathbf{e}_2,\mathbf{e}_2) & (\mathbf{e}_2,\mathbf{e}_3) \\ (\mathbf{e}_3,\mathbf{e}_1) & (\mathbf{e}_3,\mathbf{e}_2) & (\mathbf{e}_3,\mathbf{e}_3) \end{bmatrix} \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \end{bmatrix} = \begin{bmatrix} (\mathbf{v},\mathbf{e}_1) \\ (\mathbf{v},\mathbf{e}_2) \\ (\mathbf{v},\mathbf{e}_3) \end{bmatrix}. \tag{3.7}$$

(The confused reader may want to take a coffee break and then ponder this paragraph again.)

Equation (3.7) can now be applied to the derivation of the Fourier coefficients. In this particular case, the equivalent of the general vector $\mathbf{v}$ is an arbitrary periodic function $f(t)$. The equivalent of the three basis vectors $\mathbf{e}_i$ are the Fourier functions 1, $\sin(nt)$, and $\cos(nt)$. We already know, from Equation (3.2), that they are orthogonal. Finally, the equivalent of the unknowns α_i is the Fourier coefficients a_0, a_n, and b_n. As the inner product of our functions $f(t)$ we choose, as usual, the integral

$$(f,g) = \frac{1}{2\pi}\int_{-\pi}^{\pi} f(t)g(t)\,dt.$$

With these properties in mind, Equation (3.7) becomes

$$\begin{bmatrix} (1,1) & 0 & 0 \\ 0 & (\cos nt,\cos nt) & 0 \\ 0 & 0 & (\sin nt,\sin nt) \end{bmatrix} \begin{bmatrix} a_0 \\ a_n \\ b_n \end{bmatrix} = \begin{bmatrix} (f,1) \\ (f,\cos nt) \\ (f,\sin nt) \end{bmatrix} \tag{3.8}.$$

(The matrix of Equation (3.8) is diagonal because its elements are inner products of the basis functions, which themselves are orthogonal.)

Because its matrix is diagonal, Equation (3.8) is easy to solve, and its solutions are

$$\begin{aligned} a_0 &= \frac{(f,1)}{(1,1)} = \frac{1}{2\pi}\int_{-\pi}^{\pi} f(t)\,dt, \\ a_n &= \frac{(f,\cos nt)}{(\cos nt,\cos nt)} = \frac{1}{\pi}\int_{-\pi}^{\pi} f(t)\cos nt\,dt, \\ b_n &= \frac{(f,\sin nt)}{(\sin nt,\sin nt)} = \frac{1}{\pi}\int_{-\pi}^{\pi} f(t)\sin nt\,dt, \end{aligned}$$

identical to those of Equation (3.3).

3.5 Odd-Even Time-Saving Considerations

Computing the Fourier coefficients for a given function may be time consuming, which is why it is useful to know in advance when some of the coefficients are zero, and this has to do with the parity of the function $f(t)$ being decomposed. Certain functions, but not all, can be designated as odd or even, and these properties are defined here.

- A function $f(x)$ is odd if $f(-x) = -f(x)$ for all values of the parameter x. The sine function is odd and it is easy to see, just by examining its graph, that any area under a sine curve from 0 to any a, equals the corresponding area *under* the curve from $-a$ to 0. This is expressed mathematically by

$$\int_{-a}^{a} f(x)dx = 0.$$

- A function $f(x)$ is even if $f(-x) = f(x)$ for all values of the parameter x. The cosine function, for example, is even, and it is again easy to see, from its graph (check Figure 3.1), that for an even function, the following is always true

$$\int_{-a}^{a} f(x)dx = 2\int_{0}^{a} f(x)dx.$$

For our purposes, it is important to know the parity of a product of two functions $f(x)$ and $g(x)$, so we examine the following basic cases:

- Both f and g are odd. Their product satisfies $f(-x)g(-x) = [-f(x)][-g(x)] = f(x)g(x)$, and is therefore even.

- Both f and g are even. Their product satisfies $f(-x)g(-x) = f(x)g(x)$, and is also even.

- f and g have different parities, so $f(-x)g(-x) = -f(x)g(x)$, and their product is odd.

What this implies for the computations of the Fourier coefficients is that (1) if the function $f(t)$ that is being decomposed is odd, then the a_0 and all a_n coefficients are zero, and if $f(t)$ is even, then its b_n coefficients are zero. Both these facts are trivial to verify, and they may save computational time.

3.6 Examples

The first example of the Fourier series is the well-known square wave (or step function). It is defined by

$$f(t) = \begin{cases} 1, & 0 < t < L \\ -1, & L < t < 2L \end{cases}.$$

Notice that this function is undefined at $t = L$, because it is vertical at this point (see Figure 1 in the Preface) and a mathematical function of the form $y = f(x)$ cannot produce a vertical segment (vertical lines can be computed and displayed as parametric functions, though).

This makes the step function piecewise continuous on the interval $[0, 2L]$. In general, the Fourier series of a function f converges to f in regions where f is continuous. At a point where f is undefined, its series converges to a midpoint value, which in our case is 0.5.

Because $f(t)$ is -1 from L to $2L$, the integrals in the expressions for the coefficients need only go from zero to L, an interval where $f(t)$ equals 1. This simplifies these expressions which become

$$b_n = \frac{1}{L}\int_0^{2L} f(x)\sin\left[\frac{n\pi x}{L}\right] dx,$$

which is further reduced to

$$\begin{aligned} b_n &= \frac{2}{L}\int_0^{L} f(x)\sin\left[\frac{n\pi x}{L}\right] dx \\ &= \frac{4}{n\pi}\sin^2(\frac{1}{2}n\pi) \\ &= \frac{2}{n\pi}[1-(-1)^n] \\ &= \frac{4}{n\pi}\begin{cases} 0, & n \text{ even} \\ 1 & n \text{ odd.} \end{cases} \end{aligned}$$

The final Fourier series is then

$$f(x) = \frac{4}{\pi}\sum_{n=1,3,5,\ldots}^{\infty} \frac{1}{n}\sin\left[\frac{n\pi x}{L}\right].$$

Figure 3.3 shows how this sum approaches the square wave when more and more terms are included. The Mathematica code for this plot (for $L = \pi$) is

```
Plot[(4/Pi)Sum[(1/n )Sin[n Pi t/L],{n,{1,3,5,7,9,11,13,15}}],{t,-2L,2L}]
```

The next example is a 2π-periodic wave that is an inverted pyramid (Figure 3.4). Its central part is defined by

$$f(t) = \begin{cases} -t & -\pi \le t < 0 \\ t & 0 \le t < \pi \end{cases},$$

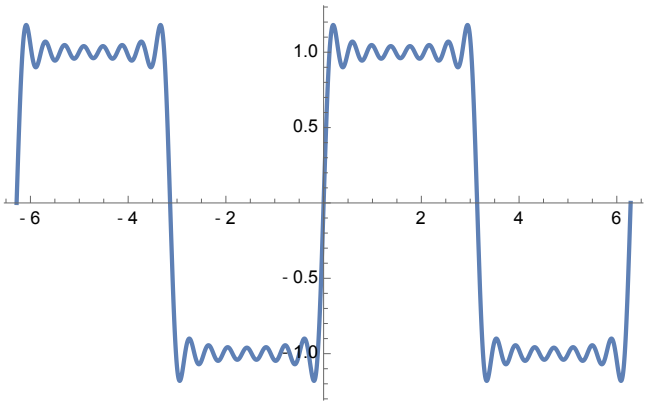


Figure 3.3: An Approximation of a Square Wave.

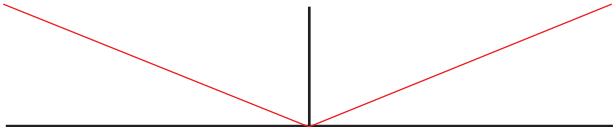


Figure 3.4: An Inverted Pyramid Periodic Wave.

which shows that $f(t)$ is symmetric about the y-axis and is therefore an even function, implying that its b_n Fourier coefficients are all zero. The other coefficients are easy to derive.

$$\begin{aligned}
a_0 =& \frac{1}{2\pi}\int_{-\pi}^{\pi} f(t)dt = \frac{2}{2\pi}\int_{0}^{\pi} f(t)dt = \frac{1}{\pi}\int_{0}^{\pi} t\,dt = \frac{1}{\pi}\,\frac{t^2}{2}\,\Big|_0^{\pi} = \pi/2,\\
a_n =& \frac{1}{\pi}\int_{-\pi}^{\pi} f(t)\cos(nt)dt = \frac{2}{\pi}\int_{0}^{\pi} f(t)\cos(nt)dt = \frac{2}{\pi}\int_{0}^{\pi} ft\cos(nt)dt =\\
& \frac{2}{\pi}\left[\frac{t}{n}\sin(nt)\,\Big|_0^{\pi} - \frac{1}{n}\int_{0}^{\pi}\sin(nt)dt\right] = \frac{2}{\pi}\cdot\frac{1}{n}\left(\frac{1}{n}\cos(nt)\right)\Big|_0^{\pi} =\\
& \frac{2}{n^2\pi}\big(\cos(n\pi) - \cos(0)\big) = \frac{2}{n^2\pi}((-1)^n - 1).
\end{aligned}$$

As before, the term $((-1)^n - 1)$ is zero for even values of n and equals -2 for odd values. We can now write the Fourier sum for this function as:

$$f(t) = \pi/2 + \sum_{n=1}^{\infty} \frac{-4}{(2n-1)^2\pi}\cos\big((2n-1))t\big),$$

which includes only the odd values of n. The single Mathematica line

```
Plot[(Pi/2) + Sum[(-4/((2n-1)^2 Pi)) Cos[(2n-1) t], {n, 1, 5}], {t, -Pi, Pi}]
```

generates a convincing plot, very similar to that of Figure 3.4.

◇ **Exercise 3.1:** Find the Fourier series of the 2π-periodic wave whose central part is

$$f(t) = \begin{cases} -t & -\pi \le t < 0 \\ 0 & 0 \le t < \pi \end{cases} .$$

Next, Figure 3.5 lists similar code for a function that performs two quarter-period of a cosine.

```
In[190]:= T = Pi; (* Example, Two 1st quarters of a cosine *)
          dsin[x_] := Piecewise[{{Cos[x], 0 < x ≤ Pi / 2}, {Cos[x - Pi / 2.], Pi / 2 < x < Pi}}]
          g0 = Plot[dsin[x], {x, 0, T}, AxesOrigin → {0, 0}, PlotStyle → Thickness[0.01]];
          a0 = (2 / T) Integrate[dsin[x], {x, 0, T}]
          ak = (2 / T) Integrate[dsin[x] Cos[2 k Pi x / T], {x, 0, T}]
          bk = (2 / T) Integrate[dsin[x] Sin[2 k Pi x / T], {x, 0, T}]
          s[n_, x_] := a0 / 2 + Sum[ak Cos[2 k Pi x / T] + bk Sin[2 k Pi x / T], {k, 1, n}]
          g1 = Plot[{s[1, x], s[2, x], s[3, x], s[4, x], s[8, x], s[15, x]}, {x, 0, T}, AxesOrigin → {0, 0}];
          Show[g0, g1, AxesOrigin → {0, 0}, PlotRange → {{0, T}, All}]
Out[192]= 1.27324

Out[193]= 2 × (-1. Cos[3.14159 k] - 1. Cos[6.28319 k] - 2. k Sin[3.14159 k] - 5.79087 × 10^-16 k Sin[6.28319 k]) / ((-1. + 4. k^2) π)

Out[194]= 2 × (2. k + 2. k Cos[3.14159 k] + 5.79087 × 10^-16 k Cos[6.28319 k] - 1. Sin[3.14159 k] - 2. Cos[3.14159 k] Sin[3.14159 k]) / ((-1. + 4. k^2) π)
```

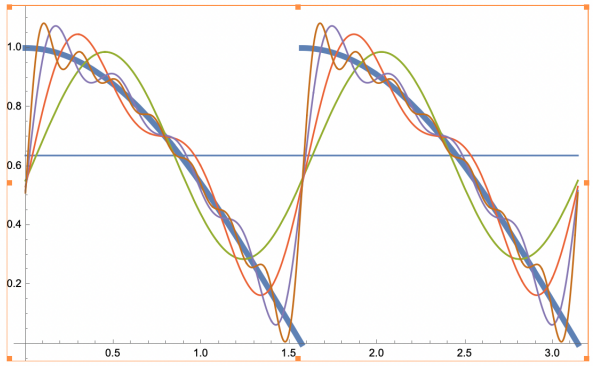


Figure 3.5: Fourier Series for a Cosine Step Function.

Finally, Figure 3.6 lists Mathematica code for the Fourier series that approximates a half circle. This is based on the code shown in reference [ScuolaTech 12].

```
Clear; R = 5; T = 10; (* Example, half a circle *)
hc[x_] := Sqrt[R^2 - x^2];
g0 = Plot[hc[x], {x, -R, R}, PlotStyle → Thickness[0.01]];
a0 = (2 / T) Integrate[hc[x], {x, -T / 2, T / 2}]
ak = (2 / T) Integrate[hc[x] Cos[2. k Pi x / T], {x, -T / 2, T / 2}]
bk = (2 / T) Integrate[hc[x] Sin[2 k Pi x / T], {x, -T / 2, T / 2}]
s[n_, x_] := a0 / 2 + Sum[ak Cos[2 k Pi x / T] + bk Sin[2 k Pi x / T], {k, 1, n}]
g1 = Plot[{s[1, x], s[2, x], s[3, x], s[4, x], s[8, x], s[15, x]}, {x, -R, R}];
Show[g0, g1, PlotRange → {{-R, R}, All}]
```

Out[104]= $\frac{5\pi}{2}$

Out[105]= $\frac{5.\ \text{BesselJ}[1,\ 3.14159\ \text{Abs}[k]]}{\text{Abs}[k]}$ if $k \in \mathbb{R}$

Out[106]= 0

Out[109]=

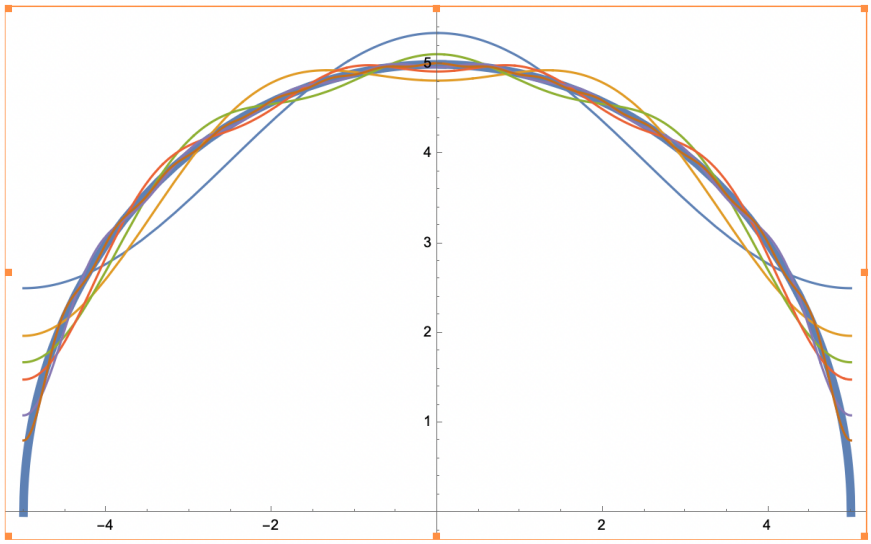


Figure 3.6: Fourier Series for a Half Circle.

3.7 Two-Dimensional Fourier Series

The one-dimensional Fourier Series can be extended to two dimensions, where a periodic two-parameter function $f(x, y)$ is broken up into its sinusoid components. The main expressions are given here with examples but with no derivation. Two short references are [2D series 22a] and [2D series 22b]. Given a two-parameter function $f(x, y)$ with a period of 2π in both parameter, its two-dimensional Fourier series is

$$\frac{1}{2}a_0(y) + \sum_{n=1}^{\infty} \left[a_n(y)\cos(nx) + b_n(y)\sin(nx)\right],$$

where the coefficients are

$$a_n(y) = \frac{1}{\pi}\int_{-\pi}^{\pi} f(x, y)\cos(nx)\,dx, \quad \text{and} \quad b_n(y) = \frac{1}{\pi}\int_{-\pi}^{\pi} f(x, y)\sin(nx)\,dx.$$

The value of n determines the order of the term in the series. Thus, $n = 1, 2, 3$ generates the third term.

Figure 3.7 is an example. It displays the periodic function $\cos(x^2)+\sin(-y)$ and the first five terms of its series. Notice that the computations are done by the Mathematica command `FourierSeries`.

◇ **Exercise 3.2:** Use Mathematical software to compute and display several terms of the series of $\cos(x^2) \times \sin(y)$. Employ the expressions given here, rather than any built-in commands of the software. Include a full listing of the mathematical expressions for each term of the series.

3.8 The Complex Exponential Representation

The expressions for the Fourier series that we have so far derived are referred to as trigonometric. There is, however, a different approach to the Fourier series, one that expresses it in terms of complex exponential numbers.

Given a periodic function $x(t)$ with a fundamental period of T_0 (there may be other periods), and a fundamental frequency $\omega_0 = 2\pi/T_0$, its continuous complex Fourier series $X(t)$ is given by

$$X(t) = \sum_{k=-\infty}^{\infty} C_k e^{ik\omega_0 t},$$

where the (infinitely many) coefficients C_k are given by

$$C_k = \begin{cases} \frac{1}{T_0}\int_{T_0} x(t)dt, & k = 0 \\ \frac{1}{T_0}\int_{T_0} x(t)e^{-ik\omega_0 t}dt, & k \neq 0. \end{cases}$$

Note that the subscript T_0 of the integral means an integration interval of one period. It can go from the start to the end of a period or from any point in a period to the corresponding point in the next period.

```
Clear["Global`*"]
f[x_, y_] := Cos[x^2] + Sin[-y];
Plot3D[f[x, y], {x, -Pi, Pi}, {y, -Pi, Pi}]
Do[p = FourierSeries[f[x, y], {x, y}, {d, d}];
 Print[Plot3D[p, {x, -Pi, Pi}, {y, -Pi, Pi}]], {d, 1, 5}]
```

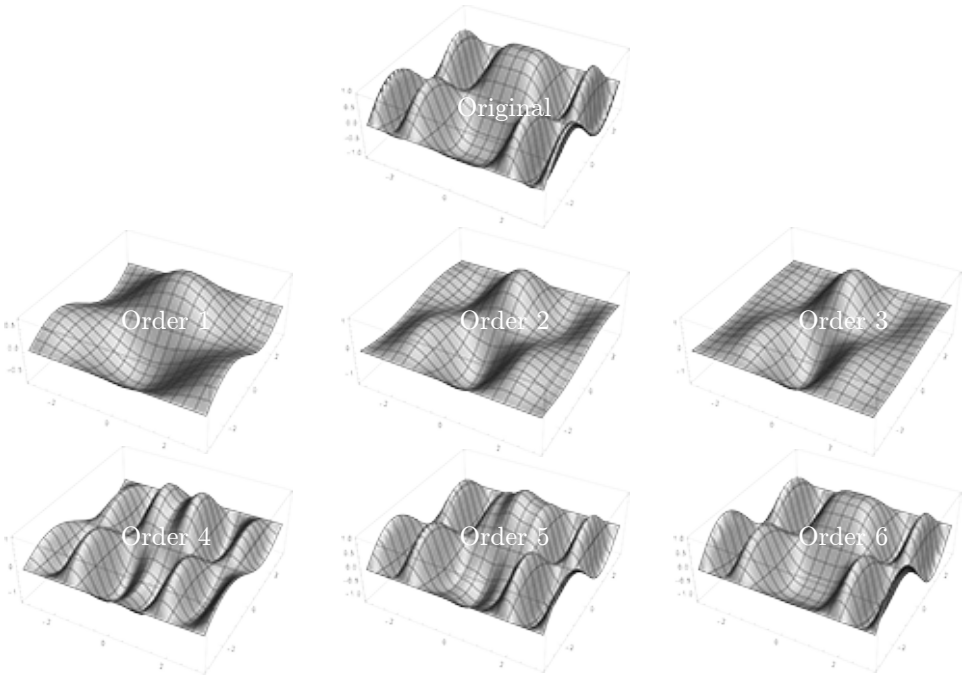


Figure 3.7: A 2D Fourier Series of $\cos(x^2) + \sin(y)$.

In general, the coefficients C_k are complex numbers that have the form $C_k = \alpha_k + i\beta_k$. If $x(t)$ is real, C_{-k} equals the conjugate of C_k. Also, if function $x(t)$ is even, then the C_k's are real numbers, and if $x(t)$ is odd, all the C_k are imaginary. However, most functions used in practice are neither odd nor even.

The relations between the trigonometric and the complex exponential representations of the Fourier series are expressed in terms of their coefficients as follows:

$$a_k = 2\,\text{Real}[C_k] \quad \text{and} \quad b_k = -2\,\text{Imaginary}[C_k].$$

3.9 The Discrete Fourier Series

The discrete case of the complex representation is very similar. It is called the discrete-time-Fourier series (DTFS), and is closely related to the discrete Fourier transform (DFT) of Section 4.4. Given a periodic sequence of numbers $x[n]$ with period N, its DTFS $X[n]$ is given by

$$X[n] = \sum_{k=\langle N\rangle} C_k e^{2\pi i k\frac{n}{N}}, \tag{3.9}$$

where $2\pi/N$ is the fundamental frequency ω_0 and the N coefficients C_k are given by

$$C_k = \frac{1}{N}\sum_{k=\langle N\rangle} x[n]e^{-2\pi i k\frac{n}{N}}. \tag{3.10}$$

Example. Compute the discrete Fourier series of the periodic sequence

Value	...	1	0	0	0	1	2	1	0	0	0	1	2	1	...
Index	...	−5	−4	−3	−2	−1	0	1	2	3	4	5	6	7	...

We first notice that its period is 6, so its fundamental frequency is $\omega_0 = 2\pi/6 = \pi/3$. The first step is to compute the six coefficients C_k from Equation (3.10). The sum has to be over a full period of the sequence, any period, so we select the period from -2 to 3.

$$\begin{aligned}
C_k &= \frac{1}{N}\sum_{n=\langle N\rangle} x[n]e^{-\frac{2\pi}{N}ikn} = \frac{1}{6}\sum_{n=-2}^{3} x[n]e^{-\omega_0 ikn}\\
&= \frac{1}{6}\left[0 + 1\cdot e^{-\omega_0 ik(-1)} + 2\cdot e^{-\omega_0 ik(0)} + 1\cdot e^{-\omega_0 ik(1)} + 0 + 0\right]\\
&= \frac{1}{6}\left[e^{\omega_0 ik} + 2 + e^{-\omega_0 ik}\right]\\
&= \frac{1}{6}\left[2 + 2\cos(k\pi/3)\right]. \text{ (from Euler}'\text{s formula)}
\end{aligned}$$

The next step is to compute the six C_k coefficients

$$\begin{aligned}
C_{-2} &= \frac{1}{6}\left[2 + 2\cos((-2)\pi/3)\right] = \frac{1}{6}\left[2 + 2\cos(2\pi/3)\right] = \frac{1}{6}\left[2 + 2\cdot(-1/2)\right] = 1/6\\
C_{-1} &= \frac{1}{6}\left[2 + 2\cos((-1)\pi/3)\right] = \frac{1}{6}\left[2 + 2\cos(\pi/3)\right] = 3/6 = 1/2\\
C_0 &= \frac{1}{6}\left[2 + 2\cos(0\cdot\pi/3)\right] = 4/6 = 2/3\\
C_1 &= C_{-1} = 1/2 \text{ (symmetric because cosine is even)}\\
C_2 &= C_{-2} = 1/6\\
C_3 &= \frac{1}{6}\left[2 + 2\cos(3\pi/3)\right] = \frac{1}{6}\left[2 + 2\cdot(-1)\right] = 0
\end{aligned}$$

.

The final result is

Value	...	1/2	1/6	0	1/6	1/2	2/3	1/2	1/6	0	1/6	1/2	2/3	1/2	...
Index	...	−5	−4	−3	−2	−1	0	1	2	3	4	5	6	7	...

It tells how much of ω_0 is needed for each value of k in order to reconstruct the original sequence with Equation (3.9).

Normally, we would start by computing the N coefficients C_k from Equation (3.10) and then use them to compute the series $X[n]$, also for the M values of n, from Equation (3.9). This, however, is time consuming and requires many calculations, so it's nice to know that sometimes there is a shortcut. There are cases where it is possible to compute the C_k coefficients directly from Equation (3.9).

Imagine that we can write an equation with $X[n]$ on the left-hand side, and on the right have an expression similar to the right-hand side of Equation (3.9), i.e., the sum of constants multiplied by complex exponentials. In such a case, these constants must be the coefficients C_k. Here is a simple example where $X[n] = \cos\bigl(\frac{2\pi}{8}n\bigr)$. The period of this expression is $M = 8$, and we first rewrite it using Euler's formula

$$X[n] = \cos\left(\frac{2\pi}{8}n\right) = \frac{1}{2}\left[e^{2\pi i\frac{n}{8}} + e^{-2\pi i\frac{n}{8}}\right]$$
$$= \frac{1}{2}e^{2\pi i(1)\frac{n}{8}} + \frac{1}{2}e^{2\pi i(-1)\frac{n}{8}} = C_1 e^{2\pi i(1)\frac{n}{8}} + C_{-1}e^{2\pi i(-1)\frac{n}{8}}.$$

When we compare this expression to Equation (3.9) it becomes clear that both C_1 and C_{-1} equal 1/2 and the other coefficients are zero. As for C_{-1}, it equals C_7 both because the period of the sequence is 8 and because it can easily be shown by direct computation that $e^{2\pi i(-1)\frac{n}{8}} = e^{2\pi i(7)\frac{n}{8}}$.

◇ **Exercise 3.3:** Go through a similar shortcut starting from the series

$$X[n] = \sin\left(\frac{2\pi}{10}n\right).$$

The Two-Dimensional Discrete Fourier Series

This version of the Fourier series applies to the case where the data consists of a two-dimensional array $x[n,m]$ of numbers that are periodic in both dimensions. The numbers in each row have a period N and the numbers in each column have a period M. Here is an example of such an array with 6×3 periodicity.

```
... 13   9 -3 0 16 -81 13   9 -3 0 16 -81 13   9 -3 0 16 -81 ...
...  4 -23 77 8 41   0  4 -23 77 8 41   0  4 -23 77 8 41   0 ...
... 14  93  0 8 -1  10 14  93  0 8 -1  10 14  93  0 8 -1  10 ...
... 13   9 -3 0 16 -81 13   9 -3 0 16 -81 13   9 -3 0 16 -81 ...
...  4 -23 77 8 41   0  4 -23 77 8 41   0  4 -23 77 8 41   0 ...
... 14  93  0 8 -1  10 14  93  0 8 -1  10 14  93  0 8 -1  10 ...
... 13   9 -3 0 16 -81 13   9 -3 0 16 -81 13   9 -3 0 16 -81 ...
...  4 -23 77 8 41   0  4 -23 77 8 41   0  4 -23 77 8 41   0 ...
... 14  93  0 8 -1  10 14  93  0 8 -1  10 14  93  0 8 -1  10 ...
...
```

The DFTS of such data is given by a direct extension of Equations (3.9) and (3.10) as follows:

$$X[n,m] = \sum_{k=\langle N\rangle}\sum_{l=\langle M\rangle} C_{k,l}e^{2\pi i(k\frac{n}{N}+l\frac{m}{M})},$$
$$C_{k,l} = \frac{1}{N\cdot M}\sum_{k=\langle N\rangle}\sum_{l=\langle M\rangle} x[n,m]e^{-2\pi i(k\frac{n}{N}+l\frac{m}{M})}.$$

Fourier's theorem is not only one of the most beautiful results of modern mathematics, but it may be said to furnish an indispensable instrument in the treatment of nearly every recondite question in modern physics.

—James Clerk Maxwell.

4
The Fourier Transform

The Fourier transform is a mathematical operation that breaks a time-signal $x(t)$ into its constituent frequencies and ends up with a function $X(\omega)$ that indicates the strength of each frequency ω in the original signal. This operation transforms the original signal, which can be periodic or aperiodic, from the time domain, where it looks familiar, to the frequency domain, where we can examine the frequencies that constitute it and can also edit them, by modifying or deleting certain frequencies, before transforming the signal back, in order to improve it in various ways. The inverse transform exists and it is perfect in the sense that no information is lost during the transform or its inverse. Given a function $X(\omega)$, it is possible to compute its inverse Fourier transform and end up with the original time-signal $x(t)$.

Another way to look at the Fourier transform of a function $f(t)$ is to start with the Fourier series of $f(t)$, and then to collapse each of the component sinusoids of this series into a vertical bar whose height and position on the x-axis correspond respectively to the amplitude of the sinusoid and to its frequency.

Judging by the popularity of the Fourier transform we can claim that this mathematical operation produces results that are interesting, useful, and may also provide important insights into the characteristics of a time signal.

4.1 Preliminaries

We start with a few important preliminaries, for the benefit of many readers.

Transforms. A mathematical transform is a function or a rule that shows how to transform one type of data to another. Here is the story of the Fourier transform. In the early 1800's, Fourier, then a governor in Grenoble, became interested in the process of heat transfer. He thought about it, analyzed this process in his mind, and came up with an equation. The problem then changed from physics (heat transfer) to mathematics

(solving the equation). After much thought and trials, Fourier tried to transform it into a different representation. To his surprise, his equation then became much simplified and easy to solve. In fact, it naturally separated into two equations. After solving it, he still had to transform it back to its original representation.

Today, scientists and engineers use many transforms, but the underlying process is always the same. Transform your data to another representation, solve your problem in that representation, and then transform the result back to its original form. Here are a few examples:

- Roman numerals. The ancient Romans presumably knew how to operate on Roman numerals, but when *we* have to, say, multiply two Roman numerals, we may find it more convenient to transform them into the modern (Arabic) notation, multiply, and then transform the result back into a Roman numeral. Here is a simple example:

$$\text{XCVI} \times \text{XII} \rightarrow 96 \times 12 = 1152 \rightarrow \text{MCLII}.$$

- The Discrete Cosine Transform (see Wikipedia). This important transform expresses a set of data points as the sum of cosine functions with different amplitudes and frequencies. The DCT is invaluable in the data compression field, where it is used extensively in the mp3 and JPEG algorithms.

- The Hough Transform, (see Wikipedia), is used to detect patterns such as edges, in images.

Time series. (In English, the word "series" is both singular and plural.) Time series (or a time signal) is an important term in engineering. It is used in statistics, signal processing, weather forecasting, and many other scientific and engineering applications where data is collected temporally (over time). The term "time series" is defined as a series of data points which are indexed, listed, or graphed in time order. In practice, however, a time series is often a sequence of data items taken at successive equally-spaced points in time. We can also say that a time series is a sequence of discrete-time data. Examples of time series are (see also Figure 4.1) the intensity of sound waves emitted by a source of sound, the voltage generated by a microphone picking up that sound, temperature values measured daily over a year, and the weight of a person, measured periodically.

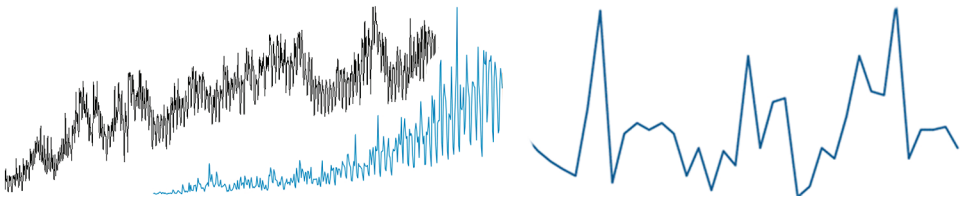


Figure 4.1: Various Time Series.

The frequency domain illustrated. A garage owner wants to increase car traffic in his business. He starts by counting the number of cars that are brought in every hour

for service or repair. The result is a time series that gets longer every day. After a few days the owner plots this sequence graphically by connecting the data points with a smooth curve. To his surprise, it is immediately obvious that more cars are brought in early in the day and also in the middle of the day, around noon (car owners use their lunch break to leave a car in the garage). Thus, the plot is a simple wave with maxima at 8 AM and 12 noon (top-left of Figure 4.2). Another surprise awaits the owner after a few weeks. It turns out that more cars are brought to the garage on Thursday and Friday (in preparation for weekend trips), on Monday and Tuesday (with problems caused by weekend trips), and on Saturday (when owners have more time). The bottom of Figure 4.2 shows a typical plot for a few weeks.

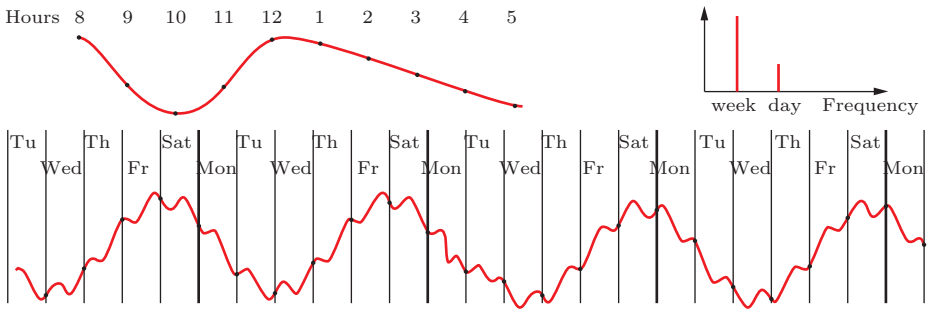


Figure 4.2: Activity in a Garage.

This information is already useful to the owner, but there is more. He now hires a consultant to compute the discrete Fourier transform of his data, and is again surprised to see the simple result (the top-right corner of the figure). Just two red vertical lines, a long one on the left and a shorter one on the right. It turns out that these lines are part of the frequency domain of the wavy function on the bottom of the figure. The original function is now said to be in the time domain, which is familiar to anyone, but the frequency domain may be both simpler and more useful. The garage owner is told that the weekly frequency component of his data is greater than the daily frequency component. The variations between days of the week are more pronounced than the variations between the hours of the day. The owner now concludes that if he wants to hire extra help for the busy times of his business, he should hire mechanics to work on Friday, Saturday, and Monday, instead of asking them to work every day, but only in the morning and around noon.

Today, both one- and two-dimensional sine waves are used to decompose objects and break them up into sums of sine waves. One-dimensional "objects" such as sound and electrical signals can be decomposed by the one-dimensional Fourier transform into sums of one-dimensional sine waves, but the big surprise is that objects of higher, two- or even three-dimensions, can also be decomposed by higher-dimensional versions of the same transform. Two-dimensional objects are visual patterns, images, and regardless of how complex and irregular they are, they can be decomposed into sums of two-dimensional sine waves, which can also be thought of as surfaces (Figure 2.4). Thus, the work of

Fourier has shown us that the world around us is, at its lowest level, a set of sums of sine waves.

How is it possible that the simple periodic functions sin and cos can be used to model complex waves and complicated periodic phenomena? This is even more surprising when we realize that the mathematical functions continue forever, their domains may be infinite; In contrast, real phenomena eventually die out. A possible explanation goes like this: Imagine a finite time signal that starts from zero and eventually dies out. Such a signal is not periodic, but we can make it periodic simply by repeating it in time both in the future and in the past. This process is referred to as a periodization of a signal. We may be interested in only a part of the signal, but we extend it in the hope that this would allow us to represent the signal by means of simple waves.

Computer Generated Text

The Fourier transform is a mathematical operation that decomposes a function into its constituent frequencies. The Fourier transform is a generalization of the Fourier series, which is used to represent periodic functions as a sum of sinusoidal functions.

The Fourier transform of a function $f(x)$ is denoted as $F(k)$, and it is defined as:

$$F(k) = \int_{-\infty}^{\infty} f(x) e^{-2\pi i k x} \, dx.$$

where i is the imaginary unit and k is a real number representing the frequency. The inverse Fourier transform is given by:

$$f(x) = (1/2\pi) \int_{-\infty}^{\infty} F(k) e^{2\pi i k x} \, dk.$$

The Fourier transform has many important properties and applications in mathematics and physics. It is used in a wide range of fields, including signal processing, image processing, and quantum mechanics. The Fourier transform can be efficiently computed using the fast Fourier transform (FFT) algorithm.

Generated by ChatGPT, 22 Dec 2022.

4.2 The Continuous Fourier Transform

The mathematics of the Fourier transform is based on the important concepts of convolution (Chapter 6) and the inner product. The latter is introduced on Page 3, where the text, examples, and figures show how the inner product provides a similarity measure between mathematical entities such as vectors and functions. Thus, given a time function $f(t)$, we can hope to identify its frequency components by computing the inner product of $f(t)$ with many sine or cosine waves of higher and higher frequencies. The codes and plots of Figures (Intro.2) and (Intro.3) explain in detail how to use integrals of inner products to identify and compute one frequency component at a time. Here is a step by step summary of this process:

1. Given a time function $c1(t)$, select a frequency f and choose the wave $\cos(ft)$ as our first candidate for a frequency component of $c1$.

2. Select a range of values for the phase ph of this cosine and compute a definite integral of the inner product of $c1(t)$ and $\cos(ft+ph)$. The integration interval should be a full period of $c1$ (but can start at any point in the period, as long as it ends at the same point). Each integral results in one number, the similarity measure (or score) between $c1(t)$ and $\cos(ft+ph)$ for the particular phase ph. These are time consuming computations and they should be done for as many phase values as possible.

3. Plot the graph of the similarity measures as a function of the phase values. Figures (Intro.3) and (Ans.1) are examples of such graphs.

4. Now comes the crucial point. Locate the maximum of this graph. The height of the graph at its maximum is proportional to the amplitude of the frequency component we are looking for. (If the height is zero, then $c1(t)$ has no component at the particular frequency f selected in step 1.) The horizontal axis of the graph at its maximum gives the phase of the frequency component for frequency f.

5. We have selected a particular frequency f and we now know the amplitude and phase of the frequency component of $c1(t)$ that corresponds to f. These three numbers completely identify the component. To find more frequency components we should repeat this tedious process for other frequencies.

Conclusion. It is now clear that identifying and computing the frequency components of a given function $c1(t)$ is easy to understand, but requires many computations and is time consuming. We now describe a more practical approach, one that uses the concept of convolution and eliminates step 4 above. We start with the following statement: Given a cosine and a sine waves with identical frequencies, zero phases, and amplitudes a and b, respectively, their sum $a\cos\theta + b\sin\theta$ is a cosine wave with amplitude $\sqrt{a^2+b^2}$ and phase $\tan^{-1}(-b/a)$.

◇ **Exercise 4.1:** Prove this statement.

We derive the expressions for the continuous Fourier transform from the results of Exercise and from the definition of convolution in Chapter 6. The idea is to change the inner products of step 2 above to convolutions, and to eliminate step 4 by performing instead two convolutions, one for a cosine and one for a sine. The definition of continuous

convolution is

$$(f \star g)(t) = \int_0^t f(\tau)g(t-\tau)d\tau = \int_0^t g(\tau)f(t-\tau)d\tau.$$

We therefore replace the integral on line 4 of Figure (Intro.3)
`fi[ph_] := Integrate[c1[t] Cos[2Pi 3t + ph], {t, 0, 2 Pi}];`
with the two convolutions of Figure 4.3.

```
(*The basic continuous FT with a convolution.
Cos[2Pi-8t]=Cos[8t] but Sin[2Pi-8t]=-Sin[8t]*)
Clear["Global`*"]
signal[t_] := 0.3 Sin[8t + 60 Degree];
a = Integrate[signal[t] Cos[8t], {t, 0, 2 Pi}];
b = Integrate[signal[t] (-Sin[8t]), {t, 0, 2 Pi}];
scor = Sqrt[a^2 + b^2];
phas = 90 - ArcTan[-b/a]/Degree;
(*Notice that $\arctan(\tan(x))$ does not always equal
 $x$, because ArcTan is a multivalued function*)
amp = 2 scor/(2Pi);
Print[a, " ", b, " ", scor, " ", phas, " ", amp]

0.81621 -0.471239 0.942478 60. 0.3
```

Figure 4.3: Two Convolutions with the Resulting Score, Amplitude, and Phase.

Notice that the definition of convolution calls for the expressions $\cos(2\pi - 8t)$ and $\sin(2\pi - 8t)$, but because of the parities of those functions, the former equals $\cos(8t)$ and the latter equals $-\sin(8t)$. Also, the arctan is a multivalued function. The Mathematica command is `ArcTan[a,b]`. It considers parameters (a, b) the coordinates of a point, and its output depends on the point's location in the 2D coordinate system. In our example, the output was always the complement of 90°. Assuming that a standard `arctan` function is available, whose range is in the interval $(-\pi/2, \pi/2)$, the `ArcTan` command is defined as follows:

$$\text{ArcTan}(x,y) = \begin{cases} \arctan(y/x), & x > 0 \\ \arctan(y/x) + \pi, & x < 0,\ y \geq 0 \\ \arctan(y/x) - \pi, & x < 0,\ y < 0 \\ \pi/2, & x = 0,\ y > 0 \\ -\pi/2, & x = 0,\ y < 0 \\ \text{undefined}, & x = 0,\ y = 0. \end{cases}$$

Once a frequency f is chosen, the outputs of the two convolutions are easily converted to amplitude and phase, and these, together with the frequency, completely define the frequency component for frequency f. The main gain over the five steps above is the elimination of step 4, which is time consuming.

Even though the continuous Fourier transform is simple and elegant, practical application are almost always based on the discrete version of this transform, which is the topic is Section 4.4.

> Fourier's great idea—the world is music, it's all waves—is the Rosetta Stone of science.
> —Alvy Ray Smith, *A Biography of the Pixel*, 2021.

4.3 The Role of Complex Numbers

Earlier, we have looked at the continuous Fourier series, Equation (3.1), as the sum of sines and cosines. As explained in Section 2.1 and Page 18, such a sum eliminates the need for phases and only requires the amplitudes of the individual series components. The continuous Fourier transform (see especially Figure 4.3) also employs separate sine and cosine waves to compute a score and a phase. In practice, however, engineers prefer a different representation of both the continuous and discrete versions of the Fourier transform, one that uses complex numbers, because of the following:

- A single complex number can express both an amplitude and a phase.
- If we compute the Fourier transform of a signal with only real-valued sines or real-valued cosines, the result would depend on the phase offset between the real-valued waves and the signal. If the signal is shifted by even a small amount, the transform would produce a different frequency domain.

For the one-dimensional continuous Fourier transform, the complex-number representation is the sum

$$F(\omega) = \int_{-\infty}^{\infty} f(t)e^{-i\omega t}dt. \tag{4.1}$$

This sum (see also Equation (4.2) for the discrete case) converts a periodic function $f(t)$ of time to a function $F(\omega)$ of frequency. The terms time domain and frequency domain are also commonly used. The symbol ω (omega) represents frequency. If we write, for example, $\sin(3)$, the frequency is 3 radians per second. However, it is common to use cycles per second, also denoted by Hz (Hertz) as the unit of frequency, which is why we use terms such as $\sin(2\pi\omega)$ or $\sin(2\pi\omega t)$. They indicate sine waves of frequencies ω Hz or ωt Hz, because there are 2π radians in a full circle.

Since we are talking about complex numbers, the symbol i stands for $\sqrt{-1}$, although electrical engineers often use j instead, because for them i also stands for electrical current.

Equation (4.1) makes use of a complex exponent because the derivative of this function is simple. The derivative of $e^{-i\omega t}$ with respect to ω is $-i\,t\cdot e^{-i\omega t}$. In order to understand the relation of (4.1) to Equation (3.1), we observe that the well-known Euler's formula satisfies

$$e^{-i\omega t} = \cos(-\omega t) + i\sin(-\omega t) = \cos(\omega t) - i\sin(\omega t),$$

which is why Equation (4.1) can be written

$$\begin{aligned} F(\omega) &= \int_{-p/2}^{p/2} f(t)\big[\cos(\omega t) - i\sin(\omega t)\big]dt, \\ &= \int_{-p/2}^{p/2} f(t)\cos(\omega t)dt - i\int_{-p/2}^{p/2} f(t)\sin(\omega t)dt, \end{aligned}$$

where p is the period of $f(t)$. Many texts claim that the integrals above should go from $-\infty$ to ∞, but if $f(t)$ is periodic, integrating over $[-\infty, \infty]$ would be the same as over $[-p/2, p/2]$ or $[0, p]$. If $f(t)$ is aperiodic, its period can be considered infinite. In such a case, integration intervals $[-\infty, \infty]$ and $[-p/2, p/2]$ would be the same, but $[0, p]$ would be different.

Now choose any frequency ω, prepare the products $f(t)\cos(\omega t)$ and $f(t)\sin(\omega t)$, which are functions of t and can therefore be visualized as curves, and then integrate each with respect to t from $-p/2$ to $p/2$. The value of each integral is the area under one of the curves, and we consider these areas the real and imaginary parts of a complex number $a + bi$. The magnitude $\sqrt{a^2 + b^2}$ of that number is the amplitude of the ωth component of the continuous Fourier transform of $f(t)$, and the angle $\arctan(b/a)$ of the number is the phase of that component, as illustrated in Figure 4.4.

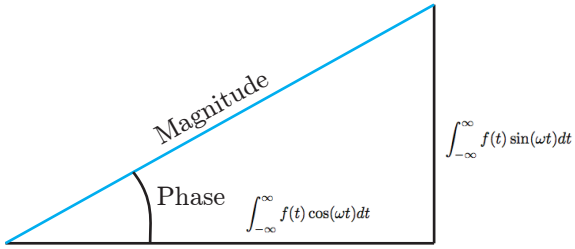


Figure 4.4: A Complex Number as Amplitude and Phase.

A word about ambiguous terminology. The term amplitude can be ambiguous, as the following paragraphs show.

- The amplitude of a sine wave is its vertical magnitude or half the distance between its top and bottom. This kind of amplitude is non-negative. It can be zero, a special case where the wave is a horizontal line which has zero frequency and an infinite period.
- The amplitude of an arbitrary wave can vary significantly all the time (we are talking about a wave in the time domain) and can even be negative, as illustrated by Figure 4.5.

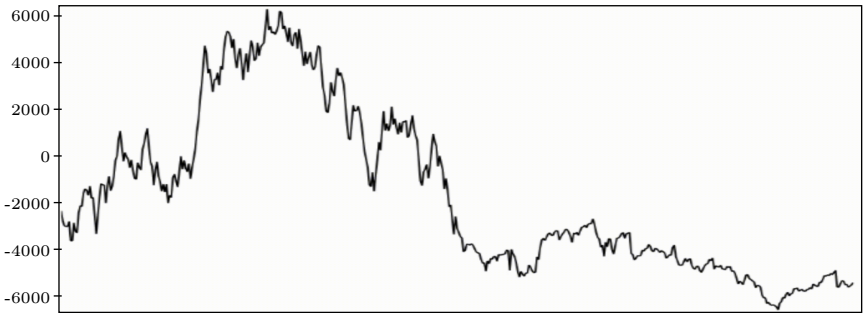


Figure 4.5: Positive and Negative Amplitudes.

The amplitude of a complex number $a+bi$ is the same as its magnitude $\sqrt{a^2+b^2}$, as shown in Figure 4.4. This quantity is positive, because it is a length. It cannot be negative and it is zero only when $a=b=0$, implying that the complex number is zero.

Section 4.4 mentions the term "frequency coefficient" many times. Such a coefficient is in general a complex number, which is why the text often uses the term amplitude to refer to the magnitude of the coefficient. This is especially noticeable in the discussion of the power spectrum on Page 72, where the term "amplitude spectrum" is used extensively. It refers to the magnitude of the frequency coefficients and not to the vertical size of any wave.

A word about the phases. The phase ϕ of a sinusoid describes how much the entire wave is shifted horizontally by ϕ/ω time units. We can think of the phase as where in its cycle the oscillation is at $t=0$, or as the number that describes the stage of the oscillation at the origin $t=0$. The relation between the amplitude of a sinusoid and its phase is clearly illustrated by an Argand diagram, such as Figure 4.6. The top of the diagram shows large and small amplitudes with different phases. Two things are clear. (1) In practice it may often be difficult to measure the phase of a sinusoid with a small amplitude. In the extreme case where the amplitude is zero, the phase has no meaning. If a DFT computation results in a complex number $0+h\,i$, then the phase h is meaningless. (2) In practice, there often is an uncertainty about the precise amplitude of a wave, as illustrated by part 3 of the figure. As long as the amplitude is large, the uncertainty in the amplitude does not affect the uncertainty in the phase much, but, as is shown in part 4, for a wave with a very small amplitude, any uncertainty about the amplitude translates to a large uncertainty of the phase.

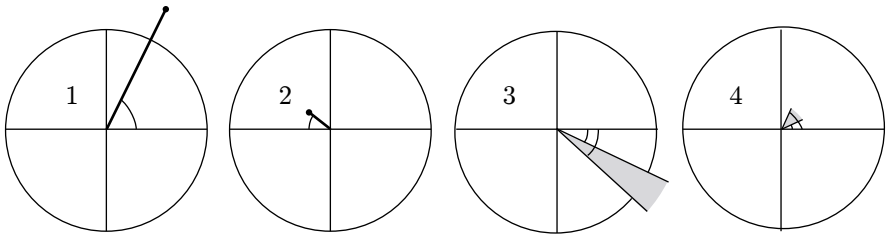


Figure 4.6: Argand Diagrams.

The entire Fourier transform of $f(t)$ is obtained when we vary ω over the interval that is the period of $f(t)$. In the continuous case, where the exact function f is available, we can vary ω continuously and for each value of ω obtain an amplitude. Plotting the amplitudes produces a curve of the amplitudes of the transform as a function of ω. In the discrete case, only n values of $f(t)$ are known. When the above process is applied to those values, the result is n complex numbers that can then be processed by the inverse one-dimensional discrete Fourier transform to reconstruct the original n values of f. The inverse transform is

$$f(t)=\int_{-\infty}^{\infty}F(\omega)e^{i\omega t}d\omega.$$

Example. Consider the function $f(t) = e^{-t}\cos(4t)$ defined on $[0, \infty]$. We construct the functions $e^{-t}\cos(4t)\cos(\omega t)$ and $e^{-t}\cos(4t)\sin(\omega t)$, compute their integrals over the entire interval of t for several values of ω and calculate the Fourier amplitude and phase for each value. Figure 4.7 shows plots of the functions for $\omega = 0$, 1, 5, 10, 15, and 20. The included code also lists the areas (array `ap`) and calculates the corresponding amplitude and phase for each of the six. Given enough values of the amplitude and phase, both can be plotted as functions of ω.

Example. The duality of the important rect and sinc functions. Figure 4.8 shows the rect function, which is simply a rectangle of width $2T$ and height 1, together with the sinc function, to be introduced in Subsection 5.2. The latter is important in signal and image processing, where it is used to reconstruct an analog, continuous time signal from a set of its digital samples. Its definition is $\text{sinc(x)} = \sin(x)/x$. This example shows that rect and sinc are Fourier duals; each is the Fourier transform of the other.

The Fourier transform of the rect function (which is often used as a filter) is

$$\begin{aligned} X(\omega) &= \int_{-\infty}^{\infty} x(t)e^{-i\omega t}dt = \int_{-T}^{T} e^{-i\omega t}\,dt \\ &= -\frac{1}{i\omega}\left(e^{i\omega T} - e^{-i\omega T}\right) = \frac{2}{\omega}\sin(\omega T) = 2T\,\text{sinc}(\omega T), \end{aligned}$$

because the rect is defined only from $-T$ to T, where its value is 1.

Going in the opposite direction, we are given a rect in the frequency domain, i.e., a frequency interval (bandwidth in engineering jargon) of $[-a, a]$ with a rectangle of height 1 over it, and we need to apply the inverse Fourier transform to find what time-domain signal would fit exactly into this frequency interval. Since the height of the rect is 1, the result is immediate

$$x(t) = \frac{1}{2\pi}\int_{-a}^{a} e^{i\omega t}d\omega = \frac{\sin(at)}{\pi t} = \frac{a}{\pi}\text{sinc}(at).$$

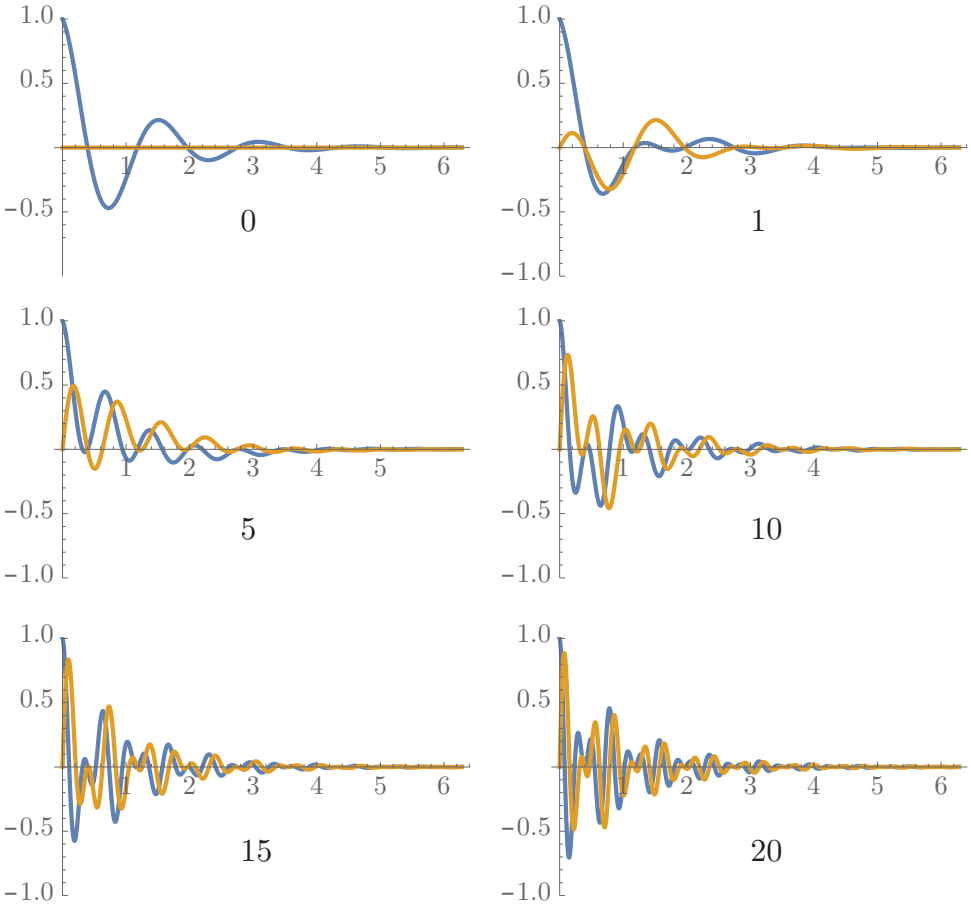
This duality can be summarized as follows

$$\begin{gathered} \text{time domain} \longleftrightarrow \text{frequency domain} \\ x(t) \longleftrightarrow X(\omega), \\ X(it) \longleftrightarrow 2\pi x(-\omega), \end{gathered}$$

which means that when any of these two functions is given in the time domain, its Fourier transform will be the other function in the frequency domain, and inversely, when any of these is given in the frequency domain, its Fourier transform will be the other function in the time domain. This duality is why the sinc filter is the ideal low-pass filter and is used to reconstruct an analog time signal from its digital samples (Page 136).

For example, when we start with $\text{sinc}(t)$ in the time domain and pass it through the Fourier transform, we end up with a function $X(\omega)$ in the frequency domain, which is a rectangle.

$$X(\omega) = \int_{-\infty}^{\infty} \text{sinc}(t)e^{-i\omega t}dt = \int_{-\infty}^{\infty} \text{sinc}(t)\cos(\omega t)dt.$$




```
Clear;(* plot amplitudes of e^{-1}cos(4t) *)
SetOptions[Plot, BaseStyle -> {FontFamily -> "cmr10", FontSize -> 10}];
omg = 20;
ampl := Exp[-t] Cos[4. t] Cos[omg t];
phas := Exp[-t] Cos[4. t] Sin[omg t];
Plot[{ampl, phas}, {t, 0, 2 Pi}, PlotRange -> {-1, 1}, ImageSize -> 72*2.45]
Integrate[{ampl, phas}, {t, 0, \[Infinity]}]

ap = {{0.0588235, 0}, {0.0692308, -0.0538462}, {0.256098, 0.304878},
 {0.0160516, 0.116614}, {0.00547958, 0.0713251}, {0.00281208, 0.0519256}};
For[i = 1, i < 7, i++,
 Print[{Sqrt[ap[[i, 1]]^2 + ap[[i, 2]]^2],
 ArcTan[ap[[i, 1]], ap[[i, 2]]]}]
```

Figure 4.7: Plots and Code for Six ω Values

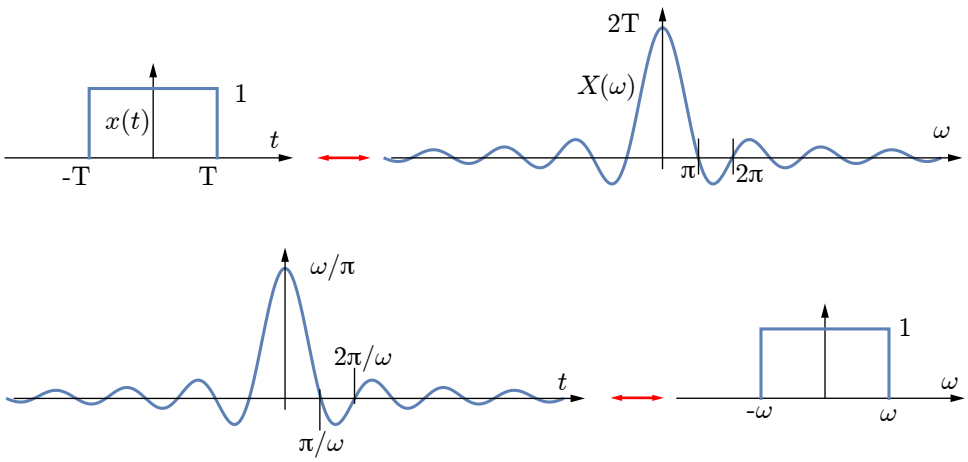


Figure 4.8: The Duality of rect and sinc.

Figure 4.9 shows the resulting approximations of the rectangle. In principle, the limits of the integration should be from $-\infty$ to ∞, but to save computation time, the four parts of the figure have limits in the intervals $[-t, t]$ for $t = 5$, 10, 15, and 50.

Digression: The beauty of sinc. The famous Euler's formulas $e^{ix} = \cos x + i \sin x$ and $e^{i\pi} + 1 = 0$, are often considered examples of beauty in mathematics. This short digression illustrates the much less known beauty hidden in the sinc function. Here are some examples, and the interested reader is directed to the Wikipedia article on sinc for more unusual properties of this important function.

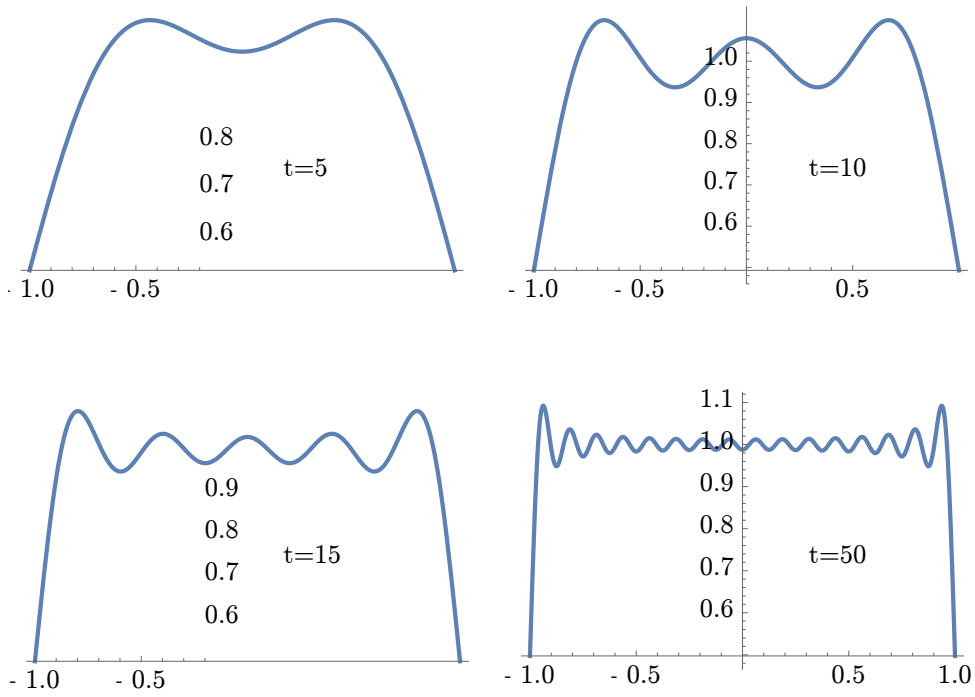
$$\sum_{n=1}^{\infty} \text{sinc}(n) = \sum_{n=1}^{\infty} \text{sinc}^2(n) = \frac{\pi - 1}{2}, \quad \sum_{n=-\infty}^{\infty} \text{sinc}(n) = \pi,$$

$$\sum_{n=1}^{\infty} (-1)^{n+1} \text{sinc}(n) = \text{sinc}(1) - \text{sinc}(2) + \text{sinc}(3) - \text{sinc}(4) \cdots = 1/2,$$

$$\sum_{n=1}^{\infty} (-1)^{n+1} \text{sinc}^2(n) = \sum_{n=1}^{\infty} (-1)^{n+1} \text{sinc}^3(n) = 1/2,$$

$$\int_{-\infty}^{\infty} \text{sinc}(x)dx = \int_{-\infty}^{\infty} \text{sinc}^2(x)dx = \pi.$$

Why does sinc feature these unusual, beautiful properties? It seems that the reason is the sinc-rect duality discussed earlier. In the frequency domain, sinc is simply a rectangle. All the complex infinite sums, squaring, and other operations above have corresponding operations on the rectangle in the frequency domain, but because a rectangle is such a simple figure, the final results are also simple. *End of digression.*




```
a = 1; (*sinc-rect duality*)
X[w_] := NIntegrate[Sinc[t] Cos[ w t]/Pi, {t, -50, 50}]
Plot[X[w], {w, -a, a}, BaseStyle -> {FontFamily -> "cmr10", FontSize -> 10},
 PlotRange -> All, AspectRatio -> 113/180, ImageSize -> 72*2.45]
```

Figure 4.9: A Rectangle as the Fourier Transform of a Sinc

4.4 The Discrete Fourier Transform

Sinusoids and many other mathematical functions are continuous, but 20th century physics and the widespread use of digital computers are based on discrete quantities. The former on elementary particles and the latter on binary digits, bits. There are two ways to look at how and why our lives now depend on discrete quantities and their manipulations. We can either say that modern technology must move with the times and convert the continuous to the discrete, or we can claim that data in the real world, which originally may have been continuous, has come to us—via our cameras, sensors, and scanners—in discrete form, and we have to process it in this form.

Whatever some may claim or believe in, we know from experience that the discrete form of the Fourier transform is important and is therefore discussed here in detail. We start with the one-dimensional discrete Fourier transform (DFT). This is a useful mathematical tool that analyzes an N-element time-series (or a time-domain signal) $x[n]$ and computes its frequency content as an N-element array $X[k]$ of frequency components. Each frequency component is a complex number whose two parts can be used to compute

the amplitude and phase of the component.

Intuitively, the DFT works by comparing $x[n]$ to sinusoidal basis functions of higher and higher frequencies (the familiar sin and cos functions). Each comparison is done by calculating (1) the inner-product (or dot product) of the N elements of $x[n]$ with N equally-spaced values of $\cos[t]$, and then (2) the inner-product of the same elements of $x[n]$ with N equally-spaced values of $i\sin[t]$, where $i = \sqrt{-1}$.

Here is a geometric explanation of this process. Given an N-point time signal, we consider its points the components of a vector T in an N-dimensional vector space, and we start a loop that iterates N times, where in iteration n we perform two steps:

1. We construct a vector S which is a complex sine wave of frequency $(n-1)$. The magnitude of S does not depend on its frequency and is the same in each iteration. The different frequency of each iteration affects only the direction of S.

2. We compute the inner product of T and S. The result is a complex number, a frequency coefficient, that we can always write in the form $a + bi$. Its absolute value (the amplitude of the coefficient) is $\sqrt{a^2 + b^2}$ and its phase (its direction in the Argand diagram) is $\arctan(b/a)$. Another representation, using Euler's notation, is $me^{i\theta}$ where m is the amplitude of the coefficient and θ is its phase. The final set of N frequency amplitudes is the Fourier spectrum of the original time signal. See also the discussion of complex numbers on Page 49.

This explanation of the DFT as an inner product of a given signal and a family of complex sine waves suggests why the DFT is so useful and universal. This transform links the values of a signal to the waves that constitute it, and waves are important because much of the work in signal processing and image processing involves electrical, sound, spatial, or light waves.

Dot products and inner products employ the same computations. The difference between them is that the former is defined for vector spaces over the reals, while the latter is defined on general vector spaces, such as the space of real functions.

Once this intuitive explanation of the DFT as an inner product of a signal and sinusoidal waves is clear, the equation of the DFT is easy to understand. We start with the continuous one-dimensional Fourier transform

$$X(\omega) = \int_{-\infty}^{\infty} x(t) e^{-2\pi i \omega t} dt,$$

which itself is based on the well-known Euler's formula $e^{ix} = \cos x + i \sin x$, and generalize it to the discrete one-dimensional case, where the only input is a sequence of N equally-spaced numbers x_n that are values of an unknown function $f(x_n)$

$$\begin{aligned} X_k &= \sum_{n=0}^{N-1} x_n \cdot e^{-2\pi i k n/N} \\ &= \sum_{n=0}^{N-1} x_n \cdot [\cos(2\pi k n/N) - i \cdot \sin(2\pi k n/N)]. \end{aligned} \tag{4.2}$$

This definition is often denoted by $\mathbf{X} = \mathcal{F}\{\mathbf{x}\}$ or simply $\mathcal{F}(\mathbf{x})$. The input to the DFT is a time series $x[n]$ of N real or complex values x_n. The output is an array $X[k]$ of N

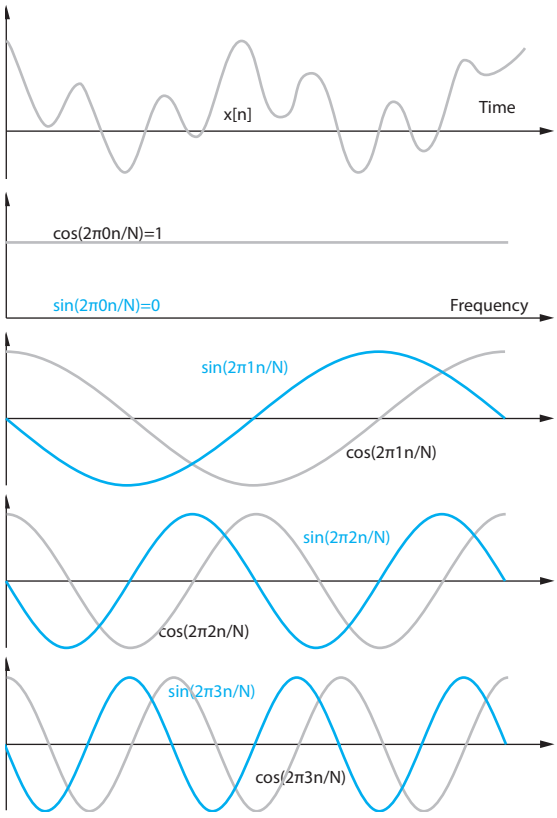


Figure 4.10: Four Sine and Cosine Waves and a Signal $x[n]$.

complex frequency coefficients X_k, for $k = 0, 1, \ldots, N-1$. Figure 4.10 shows an example of a short signal and four sine and cosine waves, corresponding to $k = 0$ through 3.

The following is a short example of computing the DFT in matlab.

```
clc
clear all
close all
x=[1 2-i -i -1+2i]
len=length(x)
xk=zeros(1,len)
for k=0:len-1
 for n=0:len-1
  xk(k+1)=xk(k+1)+x(n+1)*exp(((-i)*2*pi*k*n)/len);
 end
end
disp('transform')
disp(xk)
```

Figure 4.11: Matlab DFT Code.

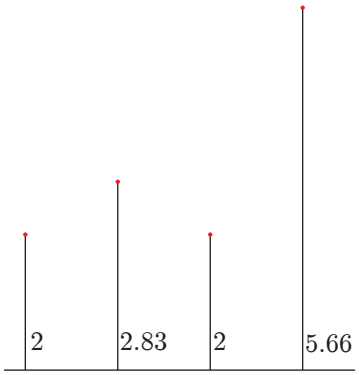


Figure 4.12: Frequency Components.

Running this code produces
`2.0000 + 0.0000i  -2.0000 - 2.0000i  0.0000 - 2.0000i  4.0000 + 4.0000i`,
or simply $(2, -2-2i, -2i, 4+4i)$. The absolute values of these complex numbers are $(2, 2.83, 2, 5.66)$ and these become the components of the frequency domain of the original array x (Figure 4.12).

The Fourier transform is reversible, i.e., a discrete signal x_n can be transformed and then reconstructed in full detail with the inverse transform whose form is

$$x_k = \frac{1}{N}\sum_{n=0}^{N-1} X_n \cdot e^{2\pi ikn/N}. \tag{4.3}$$

Two points should be mentioned here, in regard to the forward and inverse DFTs:

- Some authors prefer to use $e^{2\pi ikn/N}$ for the forward transform and $e^{-2\pi ikn/N}$ for the inverse transform.
- Many authors prefer to use the shorter notation $\omega_N \stackrel{\text{def}}{=} e^{2\pi i/N}$ which implies ω_N^{kn} instead of $e^{2\pi ikn/N}$ as well as ω_N^{-kn} instead of $e^{-2\pi ikn/N}$.

Speed is very important in algorithms and their practical implementations. The code of Figure 4.11 is slow because of the double loop it contains. If the length of the original signal is about 1000, then the double loop has to iterate about a million times. In general, we say that this code has a time complexity of the order of N^2, where N is the number of elements of the signal.

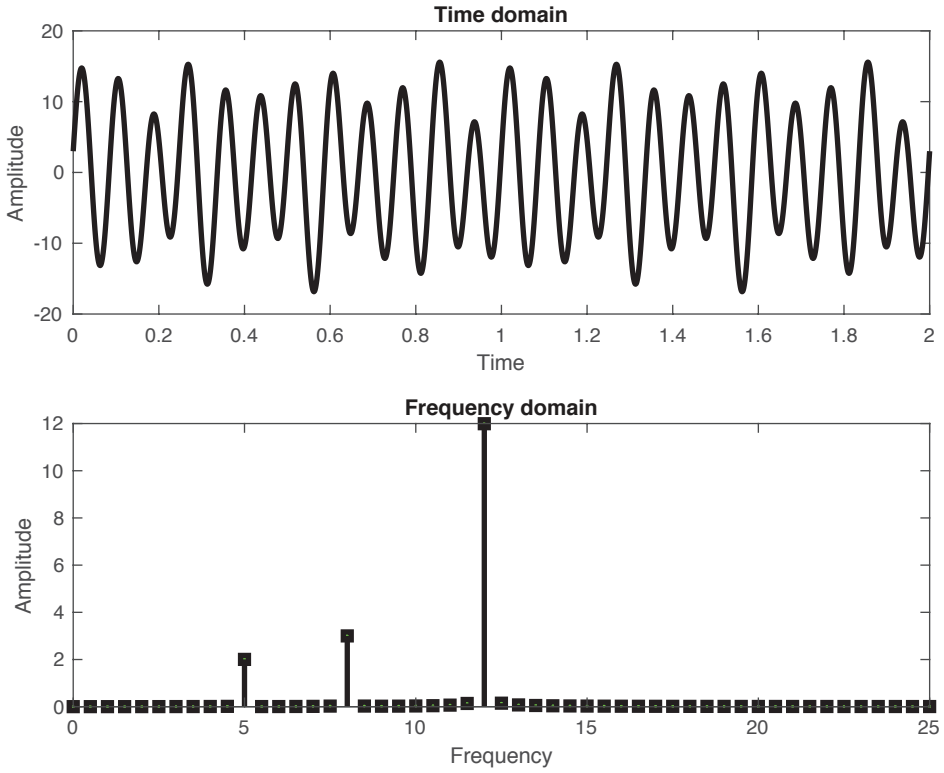
Fortunately, a much faster algorithm, known as the fast Fourier transform, or FFT, Section 4.8, has been developed and implemented as part of many mathematical software packages, such as Matlab and Mathematica. This algorithm was originated around 1805 by Gauss, and was rediscovered in 1965 by James Cooley and John Tukey. Its time complexity is on the order of $N \log_2 N$, which for $N = 1000$ implies about 10,000 operation. Compared to the million steps required by the straightforward method for $N = 1000$, the FFT reduces the computations time by a factor of 100!

Figure 4.13 illustrates the use of the FFT in Matlab. The signal to be transformed is the sum of sinusoidal waves, and the frequency plot shows clearly the three contributing frequencies and their relative strengths.

The matlab code that generated the figure is also listed. Notice the command `subplot(m,n,p)`, or `subplot(mnp)`. This Matlab command partitions the current figure window into an m-by-n matrix of plots and places the current plot in the p-th partition. Partitions are numbered row by row, top to bottom, and within each row, left to right. This is how several plots can be included in the same window. This command takes more parameters. The parameter `gca` in the `set` command refers to the style of the plot axes.

We use the term Discrete Fourier Transform (DFT) when the input signal data is finite. For infinite input data, the term DTFT (Discrete Time Fourier Transform) is used. Thus, the DFT becomes the DTFT as the length of the sample becomes infinite, and, similarly, the DTFT converges to the continuous Fourier transform as the sampling frequency approaches infinity. See also Section 6.3.

The following example of a one-dimensional Fourier transform is derived and computed by hand, step by step, in order to shed more light on this important process. We




```
time  = 0:1/1000:2;
n     = length(time);
hz    = linspace(0,1000,n);

%  signal = sinusoidal wave
signal = 2*sin(2*pi*5*time) + 3*cos(2*pi*8*time) + 12*sin(2*pi*12*time);

% transform it with fft
freq = 2*abs(fft(signal)/n);

% plot the time domain
figure(1), clf, subplot(211)
plot(time,signal,'k','linew',2)
xlabel('Time'), ylabel('Amplitude')
title('Time domain')

% plot the frequency domain
subplot(212)
stem(hz,freq,'k-s','linew',2,'markerfacecolor','g')
xlabel('Frequency'), ylabel('Amplitude')
title('Frequency domain')
set(gca,'xlim',[0 25])
```

Figure 4.13: The DFT of a General Sinusoidal Wave and Matlab Code.

start with Equation (4.2), duplicated here

$$\begin{aligned} X_k &= \sum_{n=0}^{N-1} x_n \cdot e^{-2\pi ikn/N} \\ &= \sum_{n=0}^{N-1} x_n[\cos(2\pi kn/N) - i \cdot \sin(2\pi kn/N)]. \end{aligned} \tag{4.2}$$

For the sake of compact notation we define $b_n \stackrel{\text{def}}{=} 2\pi kn/N$, and write the kth frequency bin explicitly

$$\begin{aligned} X_k &= x_0 e^{-b_0 i} + x_1 e^{-b_1 i} + \cdots + x_{N-1} e^{-b_{N-1} i} \\ &= x_0[\cos(-b_0) + i\sin(-b_0)] + x_1[\cos(-b_1) + i\sin(-b_1)] + \cdots \\ &\quad \cdots + x_{N-1}[\cos(-b_{N-1}) + i\sin(-b_{N-1})] \\ &= x_0\cos(-b_0) + x_1\cos(-b_1) + \cdots + i[x_0\sin(-b_0) + x_1\sin(-b_1) + \cdots] \\ &= A_k + iB_k. \end{aligned}$$

This produces the kth frequency bin X_k as a complex number whose magnitude and direction are $\sqrt{A_k^2 + B_k^2}$ and $\arctan(B_k/A_k)$, respectively (Figure 4.4). The magnitude of X_k is the magnitude of the sinusoid at frequency bin k and the phase of X_k is the phase of that sinusoid (how much it is shifted horizontally).

We now choose a simple sine wave of amplitude 1 and frequency 1 Hz. We sample it eight times to end up with the eight equally-spaced samples

$$x = [0, 0.7071, 1, 0.7071, 0, -0.7071, -1, -0.7071].$$

(This is because $\sin(45°) = \sin(135°) = \sqrt{2}/2 \approx 0.7071$.) The first frequency coefficient X_0 is simply the sum of the eight samples, which happens to be zero. The second coefficient takes longer to compute

$$\begin{aligned} X_1 &= 0 \cdot e^{-\frac{i2\pi\cdot 1\cdot 0}{8}} + 0.7071 \cdot e^{-\frac{i2\pi\cdot 1\cdot 1}{8}} + 1 \cdot e^{-\frac{i2\pi\cdot 1\cdot 2}{8}} + \cdots \\ &= 0 + 0.7071\big[\cos(-\frac{\pi}{4}) + i\sin(-\frac{\pi}{4})\big] + 1\big[\cos(-\frac{\pi}{2}) + i\sin(-\frac{\pi}{2})\big] + \cdots \\ &= 0 + (0.5 - 0.5i) + (-i) + (-0.5 - 0.5i) + (0.5 - 0.5i) + (-i) + (-0.5 - 0.5i) \\ &= 0 - 4i. \end{aligned}$$

Similar tedious computations result in $X_2 = X_3 = X_4 = X_5 = X_6 = 0$ and $X_7 = 0 + 4i$. Thus, the magnitudes of the two nonzero frequency coefficients are $\sqrt{0^2 + (\pm 4)^2} = 4$, and a plot of the frequency domain consists of two short vertical lines which correspond to coefficients X_1 and X_7, or frequencies of 1 Hz and 7 Hz.

The frequency resolution of the frequency bin is 1 Hz (it equals the sampling frequency of 8 Hz, divided by the number of samples, which is also eight). The frequency spectrum consists therefore of eight bins, each 1 Hz greater than its predecessor. We started with a simple, 1 Hz sine wave, so we expect to end up with a frequency of 1 Hz

in the frequency domain, but the 7 Hz frequency that was also obtained is a surprise. The explanation is that we ended up with a two-sided frequency plot, that is symmetric about the 4 Hz Nyquist frequency. The Nyquist frequency (Page 122) is the sampling frequency divided by 2, or in our case 4 Hz, and with 8 samples it is impossible to obtain any higher frequencies.

In general, the frequency diagram produced by the Fourier transform is organized as follows. It is symmetric about the Nyquist frequency, with the two parts on both sides of this frequency being mirror images of each other as shown in Figure 4.14. The extra part to the right of the Nyquist frequency is normally referred to as negative frequencies. This part is explained on Page 72 and is usually ignored. Even more! The energy of the DFT frequency diagram is spread over its two symmetric parts, which means that each is shown, including any peaks, at half its real height.

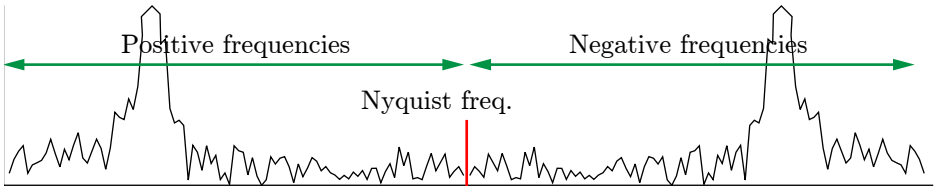


Figure 4.14: Positive and Negative Frequencies.

The reason for this behavior (see also an explanation on Page 72) is the famous Euler's formula $e^{ik} = \cos(k) + i\sin(k)$, from which is derived the cosine identity

$$\cos(k) = \frac{e^{ik} + e^{-ik}}{2}, \quad \rightarrow \quad \cos(2\pi ft) = \frac{e^{i2\pi ft} + e^{-i2\pi ft}}{2}, \tag{4.4}$$

which says that a real-valued cosine equals the average of two complex exponentials, one positive and the other negative. The negative exponential corresponds to the positive frequencies in the frequency diagram, and the positive exponential corresponds to the negative frequencies in this diagram. Because of the factor of 1/2 in Equation (4.4), each half of the diagram contains half the energy of the cosine. If we ignore the negative frequencies, we should double the height (i.e., amplitudes) of the positive frequencies in the diagram.

The following examples of the one-dimensional Fourier transform may shed more light on the meaning of the results (frequency coefficients and negative frequencies) of this important transform.

Figure 4.15 illustrates the DFT of a pure cosine wave with frequency range from 0 to 50 radians. Measured in Hz, this is $50/(2\pi) \approx 8$ Hz and the Fourier transform exhibits a spike at 8 Hz, as expected. In addition, there is another spike at 42 Hz, because 8 and 42 are equally-spaced about the Nyquist frequency of 25 (half of 50).

Figure 4.16 features damped oscillations of a pure sine wave which are obtained by multiplying it by a negative exponent. A small random perturbation is also added. The frequency of this function varies from 0 to 200 radians in steps of one radian. Expressed in Hz, this frequency is $200/(2\pi) \approx 32$. The Fourier transform exhibits spikes at 32 Hz (in

```
testData = Table[Cos[x], {x, 0, 50}];
ListLinePlot[testData, PlotRange -> Full]
ListLinePlot[Abs[Fourier[testData]], PlotRange -> Full]
```

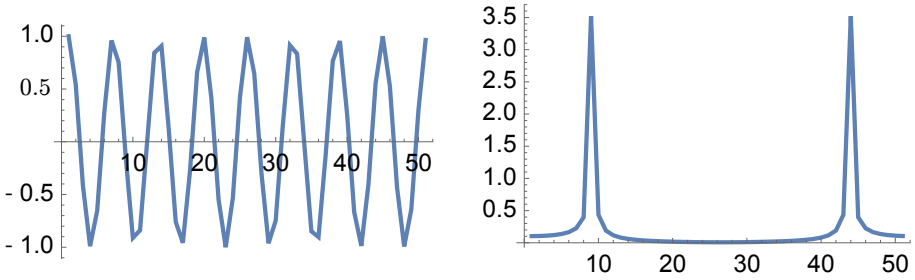


Figure 4.15: DFT of a Pure Cosine Wave.

the positive frequencies range) and 168 Hz (in the negative frequencies range), because 32 and 168 are equally-spaced about the Nyquist frequency of 100 (half of 200).

White noise is a random signal featuring equal intensity at every frequency in its entire frequency range. Constructed as a discrete time signal, white noise is a set of samples that are serially uncorrelated random numbers with zero mean and finite variance. In principle, the DFT of such a signal should be a horizontal line, because it is the same for all frequencies. In practice, any white noise, whether constructed by software or sampled from a real source, is imperfect and its DFT differs from a horizontal line to some extent. The sequence of 200 samples generated and used in Figure 4.17 is very short and is far from an ideal white noise. It was generated by a simple algorithm proposed in a textbook, and its DFT features variations between frequencies.

Robert Brown was a Scottish botanist and paleobotanist who used a microscope extensively and made important contributions to botany. Today, he is known mostly for his discovery of the random walk named Brownian motion after him. He spread pollen in water and discovered through his microscope that the individual pollen particles moved erratically and randomly in the water as if propelled by small, random collisions with invisible objects. This type of motion was later termed random walk and was finally explained by Einstein in 1905 as the result of collisions with water molecules.

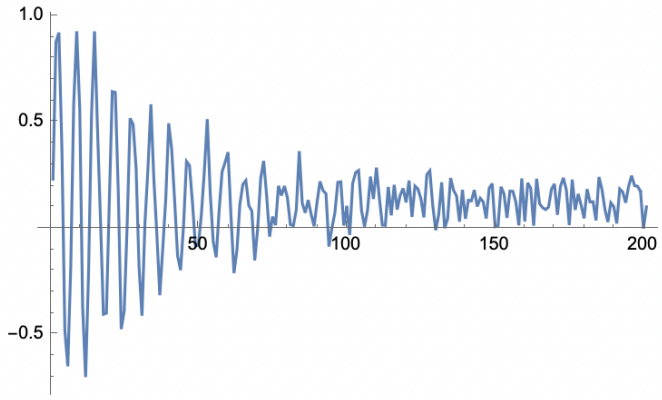
Figure 4.18 illustrates a typical sequence of 200 random walks, and it is obvious that the large-scale behavior of this sample is dominant and is much bigger than any small fluctuations. This is why the very low frequencies are so big (> 150) in the DFT, while the amplitudes of higher frequencies are much smaller (< 10).

4.4.1 Computational DFT

In theory, theory and practice are the same. In practice, as most programmers know, there is often a significant difference between them. When studying a theory, it is easy to skip and ignore "minor" practical considerations, but when writing software, those minor points tend to pop up suddenly, to confront the programmer, annoy him, place bugs in his work and doubt in his mind. Nothing works until the programmer has thoroughly understood the details of the problem being coded, as well as all its special

```
In[231]:= (* Damped oscillations *)
          testData = Table[Sin[x] Exp[-x / 40] + Random[] / 4, {x, 0, 200}];
          ListLinePlot[testData, PlotRange → Full]
          ListLinePlot[Abs[Fourier[testData]], PlotRange → Full]
```

Out[232]=



Out[233]=

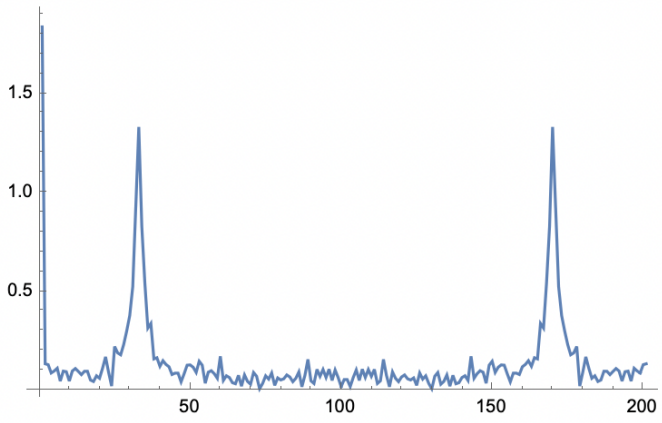
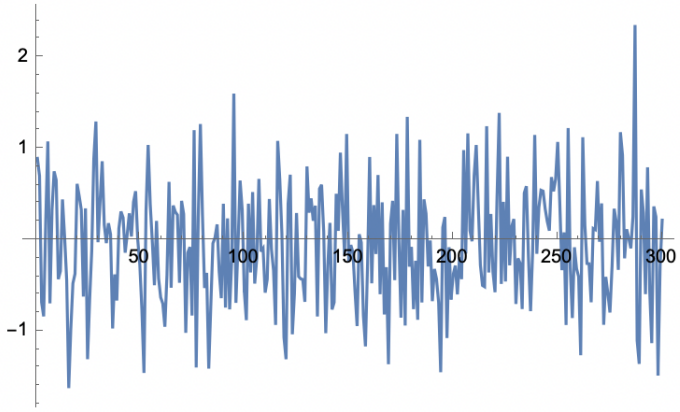


Figure 4.16: Damped Oscillations of a Pure Sine Wave.

```
(* Simulate white noise, Ref: Simulation of Communication
   Systems,  Michel C.Jeruchim, et al.*)
M = 300; X = Table[0, M];
Do[u1 = RandomReal[]; While[u1 == 0, u1 = RandomReal[]];
  u2 = RandomReal[];
  X[[i]] = Sqrt[-2. Log10[u1]] Cos[2. Pi u2];
  X[[i + 1]] = Sqrt[-2. Log10[u1]] Sin[2. Pi u2], {i, 1, M, 2}];
ListLinePlot[X]
ListPlot[Abs[Fourier[X]], PlotRange → Full, Filling → Axis]
```

Out[•]=



Out[•]=

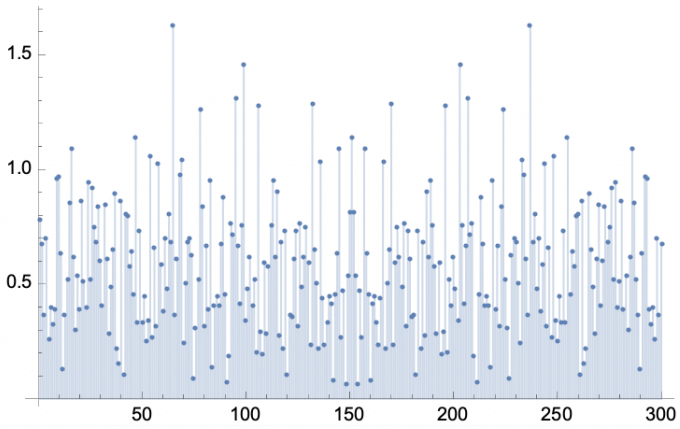


Figure 4.17: DFT of Simulated White Noise.

```
(* Brownian motion or random walk*)
data = Table[Accumulate[RandomReal[{-6, 6}, 200]], {i, 1, 1}];
ListLinePlot[%]
ListLinePlot[Abs[Fourier[data]], PlotRange → Full] ;
```

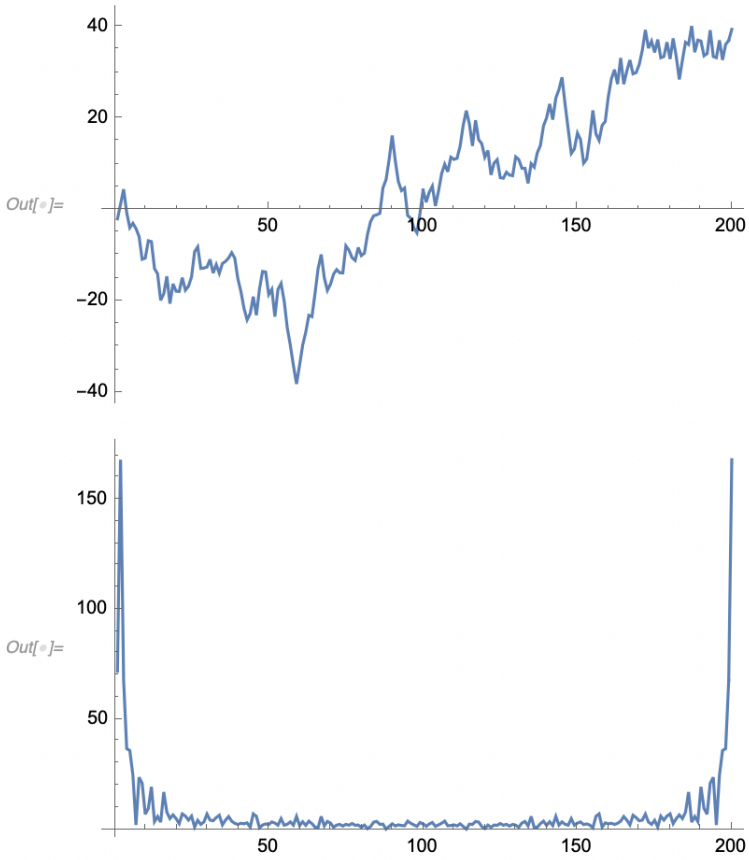


Figure 4.18: DFT of Simulated Brownian Noise (Random Walk).

cases. This section is concerned with ways of fully and precisely computing the DFT numerically, and it uses Mathematica as the software of choice.

Figure 4.19 lists Mathematica code that computes (on line 11) and displays (on line 12) 100 points of the sum of two cosine functions, one with amplitude 5 (line 10) and frequency $f1 = 20$ Hz (line 6) and the other with amplitude 1 and frequency $f2 = 15$ Hz (line 8). Line 2 specifies the total number of points of the data being transformed. Line 3 is the sampling rate, the number of samples per second. Line 4 computes the time increment ($1/50 = 0.02$ seconds) between samples. Line 6 selects a frequency of $40/(0.02 \cdot 100) = 20$ Hz, which later, on line 11, translates to exactly 40 periods during the two seconds "life" of the signal. Similarly, line 8 selects a frequency of $30/(0.02 \cdot 100) = 15$ Hz, which later, on line 11, translates to exactly 30 periods during two seconds.

Finally, line 10 selects an amplitude of 5 for the first cosine of our signal. The signal itself, a sum of two cosines, is computed, on line 11, as a table where the counter t starts at 0, ends at $\texttt{dt}(\texttt{np}-1) = (1/50)(100-1) = 99/50 = 1.98$ seconds, and is incremented in steps of $\texttt{dt} = 1/50 = 0.02$ seconds. Thus, the data to be transformed consists of 100 real values (the value of `np`).

This long preparation and selection of frequencies ensures a clean transform where only four frequency coefficients (two positive and two negative) are nonzero. We later deal with frequencies that result in a noninteger number of periods during the two-second signal.

Four tests are listed and their results displayed in Figure 4.20.

Test 1 (lines 13–17) computes the entire 100×100 DFT matrix in a double loop and performs a dot product of this matrix with the row of 100 data samples. This approach is shown for illustration purposes only, since there is no need to generate and save the entire matrix in memory. Also, for large sets of samples, this matrix may be too big, even for current computer memories.

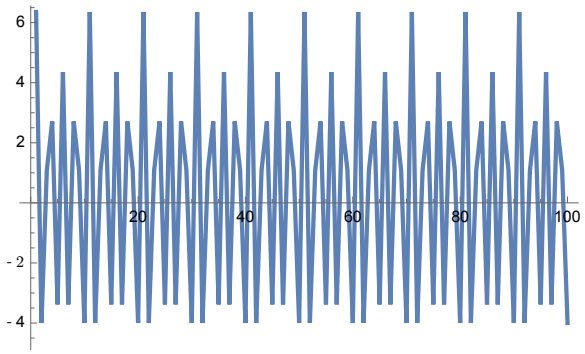
Test 2 (lines 18–20) is similar, except that the DFT matrix is computed by the single Mathematica command `FourierMatrix[]`. An important point is the length of the vertical bars in the figure. They are 50 and 10 units long, compared with the correct values of 5 and 1 in the other tests. Notice the factor of 2 that multiplies the `Abs` command on lines 16, 19, 23, and 32. This factor is needed because of the existence of negative frequencies, and has been discussed earlier. There is also a factor of `np` shown on lines 15, 16, and 31. It is needed in order to normalize the values of the frequency coefficients and bring them down to the values of the actual amplitudes.

Test 3 (lines 21–25) employs the `Fourier[data, FourierParameters->{-1, -1}]` command, which does all the work and places the Fourier frequency coefficients in `ft`, to be displayed. Line 25 prints the values of six elements of `ft`, just to make sure that the right ones (in locations 41 and 61) are nonzero and equal 2.5, half the value of the amplitude of the first cosine in the data.

Finally, test 4 (lines 29–34) works a bit harder. On line 28 it prepares an array `fourcoef` of zeros and in each iteration of the loop on lines 29–31 it computes a line of the DFT matrix (a complex sine wave), dot-products it with the data, and adds the result to the next element of `fourcoef`. Line 34 performs a test similar to that of line 25.

Following are several important notes.

- The x-axes of the four plots of Figure 4.20 seem wrong. They go from 0 to 100




```
(*Four ways to compute the 1D DFT in Mathematica*)
np = 100; (*Number of points*)
sr = 50; (*Sampling rate, # of samples/sec*)
dt = 1/sr;(*Time increment between samples*)
(*Frequency resolution = sr/np=1/(dt np) *)
f1 = 40/(dt np);(*1st freq. 40/2=20 Hz.
Generates exactly 40 periods in our 2 sec time interval*)
f2 = 30/(dt np); (*2nd freq. 30/2=15 Hz -> 30 periods*)
(*p=1/f1 is the period of f1 *)
a = 5;(*Amplitude*)
data = Table[a Cos[2Pi f1 t]+Cos[2Pi f2 t], {t, 0, dt(np-1), dt}];
ListLinePlot[data]
(*Test 1. Compute the entire DFT matrix explicitly and perform a \
dot-product with the signal*)
dftMat = Table[Exp[-2 Pi I k n/np], {n, 0, np-1}, {k, 0, np-1}];
ListPlot[2 Abs[dftMat.data]/np, Filling -> Axis,
 PlotStyle -> {Red, PointSize[0.015]}]
(*Test 2. Compute the DFT matrix with FourierMatrix[m] *)
ListPlot[2 Abs[FourierMatrix[np, FourierParameters->{-1,-1}].data],
 Filling -> Axis, PlotStyle -> {Blue, PointSize[0.015]}]
(*Test 3. Compute the DFT with the Fourier[data] command *)
ft = Fourier[data, FourierParameters->{-1,-1}];
ListPlot[2 Abs[ft], PlotRange -> All,
 PlotStyle -> {Orange, PointSize[0.015]}, Filling -> Axis]
ft[[{40,41,42,np-40, np-40+1, np-40+2}]] (*Print 6 strategic values*)
(*Test 4. For each frequency, compute a row of the DFT matrix, \
 multiply by the signal, and store in fourcoef*)
fourcoef = Table[0, {np}];
For[fq = 1, fq <= np, fq++,
  csw = Table[Exp[-I 2Pi (fq-1) t/np], {t, 0, np-1}];
  fourcoef[[fq]] = data.csw/np];
ListPlot[2 Abs[fourcoef], PlotRange -> All,
 PlotStyle -> {Gray, PointSize[0.015]}, Filling -> Axis]
ft[[{40,41,42,np-40, np-40+1, np-40+2}]] (*Check 6 strategic values*)
```

Figure 4.19: Four ways to compute the DFT in Mathematica

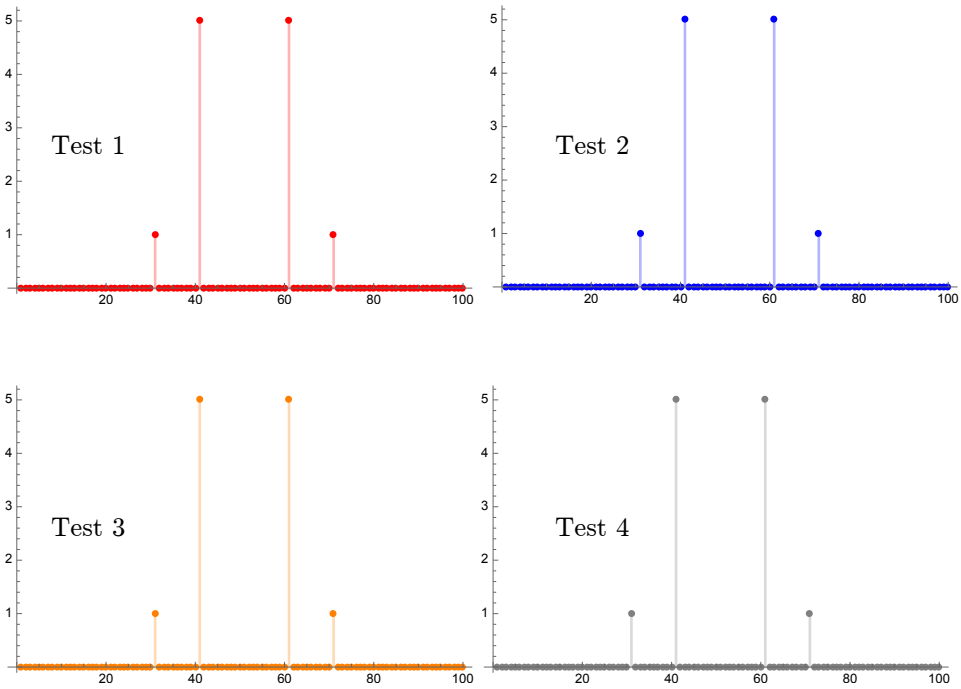


Figure 4.20: Four Tests of Computing the DFT in Mathematica.

(the number of points in the signal) because the Fourier transform outputs `np` frequency coefficients in response to an input signal with `np` points. In order to plot the correct x-axis, which corresponds to the frequencies, we should create a list `freq` with the correct frequencies from 0 to `sr`, and specify `freq` as the x-axis of the plot, as shown in this listing:

```
inc = sr/Length[data];(*increment*)
freq = Table[f, {f, 0, sr - inc, inc}]; (* x-axis of plot *)
ListPlot[Transpose[{freq, Abs[Fourier[data]]}], PlotRange -> All,
 Filling -> Axis, PlotStyle -> {Red, PointSize[0.015]}]
```

These lines should replace the `ListPlot` commands on lines 16–17, 19–20, 23–24, and 32–33. The result will show an x-axis from 0 to 50, whose center point corresponds to the Nyquist frequency of `sr`/2 = 25, and with nonzero frequency coefficients at 15 Hz and 20 Hz.

■ Frequencies `f1` and `f2` have specifically been chosen so that the length of the signal, 100 data points during two seconds, would correspond exactly to an integer number of periods. Frequency `f1` of 20 Hz implies 40 periods during the 100 points, while frequency `f2` of 15 Hz corresponds to 30 periods during the two-second signal length. This is why the tests results look so "clean" and show only two positive and two negative spikes. We now change the code of Figure 4.19 to find out what happens when other frequencies are used.

- Changing line 6 to `f1 = 40.4/(dt np)` results in Figure 4.21 for all four tests. The two frequencies of 20.2 and 15 are shown correctly, but other frequencies are nonzero, and the amplitudes seem wrong as well. The reason for this behavior is that the Fourier transform cannot accurately represent the time data with just one or two frequencies. An entire range of frequencies is needed, which may not seem elegant, but is correct.

  This is why it is important to sample the raw data so it contains an integer number of periods whenever possible. Otherwise, the computation returns blurred Dirac delta functions (see Subsection 5.2 for the delta function). Notice also the comment about the frequency resolution on line 5 of Figure 4.19. The parameters of this test were chosen such that the frequency resolution `sr/np = 1/(dt np)` would equal $1/(0.02 \cdot 100) = 0.5$. This implies that changing `f1` from `40/(dt np)` to `41/(dt np)` would change the frequency from $40/2 = 20$ to $41/2 = 20.5$, producing exactly 41 periods during the 2 second time interval. However, changing 40 to 40.4 (or any value other than an integer or half-integer) results in a number of periods that is noninteger.

- The term temporal resolution is the sampling rate, the number of samples per second, obtained from the hardware device that measures and samples the signal (50 Hz in our case). We assume that the sampling rate is fixed. Notice that the sampling rate is independent of the total time it takes to measure and sample the signal, nor is it dependent on the total number of samples collected (the number of points in the signal that's being transformed).

  In the listing of Figure 4.19, the sampling rate `sr` is defined on line 3 and is used on line 4 to compute the time gap `dt` between samples. From then on, it is `dt` that is used instead, first to specify the frequencies on lines 6 and 8, and then to actually sample the signal, on line 11.

- The term frequency resolution, sometimes also called spectral resolution, is understood as the distance between two consecutive frequency values, and is defined as the sampling rate divided by the signal length, the total number of points (100) in the signal. Line 29 of Figure 4.19 starts a loop that iterates `np` times, where `np` is the number of points in the signal (100). Thus, our frequency resolution is $50/100 = 0.5$ Hz, which means that we have a measurement of the spectral energy of the time signal every 0.5 Hz.

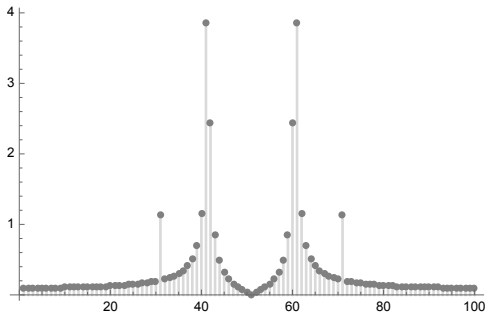


Figure 4.21: The DFT of Data With a Noninteger Number of Periods.

- There is an interesting relation between the sampling rate and the frequency resolution. Once we specify the sampling rate `sr` of a signal, its boundaries are automatically determined and can no longer be changed. On the left of the frequency axis there is zero frequency and on the right there is the Nyquist frequency. These boundaries are independent of the number of points `np` of the signal.

At the same time, the number of frequency coefficients (or frequency bins) in the output of the transform equals the number `np` of points, as indicated, for example, by line 29 of Figure 4.19. Thus, adding more points to the signal (i.e., increasing `np`) would increase the frequency resolution, while the sampling rate stays the same. Each point added to the signal increases the number of frequency coefficients by one, thereby helping to smooth the plot of the coefficients. In the absence of more data points, zeros can be appended to the signal, which would also increase the number of frequency coefficients and would result in a smooth plot. This is discussed in more detail in Subsection 4.5.1.

Line 22 of Figure 4.19 contains the option `FourierParameters`. Here is what the Wolfram literature says about it.

- `FourierParameters->{a,b}` modifies the Mathematica `Fourier` command as follows:

$$v_s = \frac{1}{N^{(1-a)/2}} \sum_{k=1}^{N} u_k e^{2\pi i b(k-1)(s-1)/N},$$

where u_k are the data samples being transformed and v_s is a frequency coefficient. Thus, the choice `FourierParameters->{0,1}` results in the standard form of the transform. The Wolfram literature recommends the following settings `{-1,1}` for data analysis and `{1,-1}` for signal processing.

- To ensure a unique inverse discrete Fourier transform, b must be relatively prime to N.

- The length of the list of data u_k supplied to `Fourier` does not have to be a power of two.

- The list given in `Fourier`[*list*] can be nested to represent an array of data in any number of dimensions.

- The array of data must be rectangular.

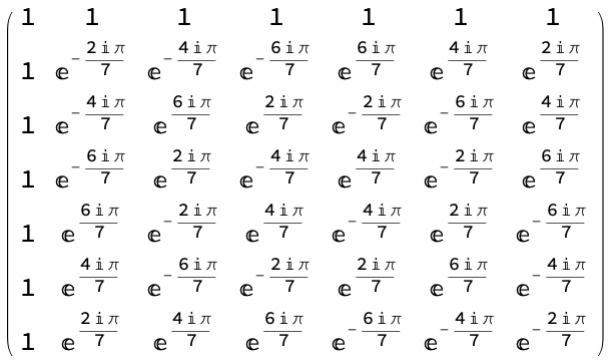
- If the elements u_k of the input list are exact numbers, `Fourier` begins by applying N to them.

Next, we examine the properties of the DFT matrix. Figure 4.22 lists two DFT matrices, an exponential and an ω, where the latter is defined by $\omega_N = e^{-2\pi i/N}$ and its elements are the Nth roots of unity. It is easy to see that the exponential matrix has the following features:

- The top row and leftmost column are all 1's, because they correspond to zero values of counters n and k.

- The trigonometric functions sin and cos have a period of 2π, which implies the following identity

$$\cos\left[\frac{-8\pi}{7}\right] = \cos\left[\frac{-8\pi}{7} + 2\pi\right] = \cos\left[\frac{(-8+14)\pi}{7}\right] = \cos\left[\frac{6\pi}{7}\right].$$



```
m = 7;
MatrixForm[Table[Exp[-2 Pi I k n/m], {n, 0, m-1}, {k, 0, m-1}]]
```

$$\begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & \omega_8^1 & \omega_8^2 & \omega_8^3 & \omega_8^4 & \omega_8^5 & \omega_8^6 & \omega_8^7 \\ 1 & \omega_8^2 & \omega_8^4 & \omega_8^6 & \omega_8^8 & \omega_8^{10} & \omega_8^{12} & \omega_8^{14} \\ 1 & \omega_8^3 & \omega_8^6 & \omega_8^9 & \omega_8^{12} & \omega_8^{15} & \omega_8^{18} & \omega_8^{21} \\ 1 & \omega_8^4 & \omega_8^8 & \omega_8^{12} & \omega_8^{16} & \omega_8^{20} & \omega_8^{24} & \omega_8^{28} \\ 1 & \omega_8^5 & \omega_8^{10} & \omega_8^{15} & \omega_8^{20} & \omega_8^{25} & \omega_8^{30} & \omega_8^{35} \\ 1 & \omega_8^6 & \omega_8^{12} & \omega_8^{18} & \omega_8^{24} & \omega_8^{30} & \omega_8^{36} & \omega_8^{42} \\ 1 & \omega_8^7 & \omega_8^{14} & \omega_8^{21} & \omega_8^{28} & \omega_8^{35} & \omega_8^{42} & \omega_8^{49} \end{pmatrix}$$

Figure 4.22: Exponential and Omega Matrices of DFT Frequency Coefficients.

With this and the famous Euler formula in mind, we conclude that $e^{6i\pi/7} = e^{-8i\pi/7}$ and similarly $e^{4i\pi/7} = e^{-10i\pi/7}$ and $e^{2i\pi/7} = e^{-12i\pi/7}$. This explains the last three items on the 2nd row of the matrix, as well as similar symmetries found in all the other rows.

- One result of the above point is that of the six non-unity elements of any row and column, three have positive exponents and the other three have negative exponents.
- The entire matrix is symmetric.
- Because of the properties above, it is easy to show that multiplying such a matrix with a data column of seven real numbers produces seven complex numbers featuring the following properties: (1) The dot product of the top row (all 1's) and the data is simply the sum of the data items, and is often referred to as the DC component of the transform (although sometimes the term DC refers to the average instead of the sum). (2) If the dot product of the second row and the data is the complex number $a+bi$, then the dot product of row 7 and the data would be the conjugate $a-bi$. (3) The same is true for the dot products of row 3 and row 6, and for row 4 and row 5. The resulting seven frequency components would therefore be

$$(\text{sum} + 0i,\ a+bi,\ c+di,\ e+fi,\ e-fi,\ c-di,\ a-bi), \tag{4.5}$$

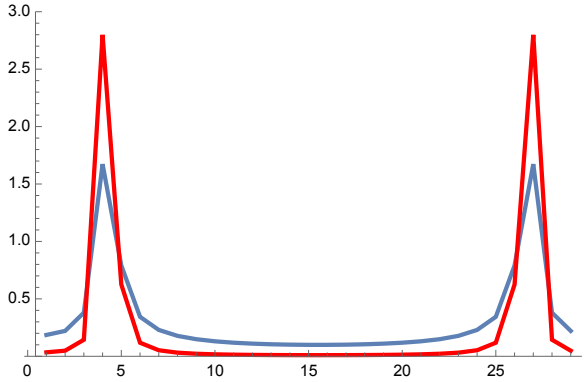
and it is this unexpected result that explains the existence of the positive and negative frequencies.

The following Mathematica lines (for $m = 7$) illustrate this result.

```
dftmt = Table[Exp[-2Pi I k n/m], {n, 0, m-1}, {k, 0, m-1}]
dftmt . {3., 7, 1, -8, 9, 0, 2}
```

◇ **Exercise 4.2:** The discussion of the DFT matrix above has assumed a 7×7 matrix. Explain what happens when the number of data samples is even. What are the results of the dot products between the rows of the Fourier matrix and the data samples.

A common term often used in connection with the DFT is power spectrum. Mathematically, power spectrum is the square of the amplitude spectrum. Each frequency coefficient is squared and the set of squared coefficients is plotted. Squaring a number greater than unity yields a bigger result, while the square of a number $n < 1$ is smaller than n. Thus, squaring a set of numbers increases the details of the large ones and shrinks the differences between the small ones. It amplifies the prominent features of the signal (which often are the important ones) in contrast with the amplitude spectrum which is concerned with all the signal's features, large and small, equally. Figure 4.23 is a quick demonstration of a power spectrum plotted on top of the corresponding amplitude spectrum.




```
signal = Table[2 Sin[ t], {t, 0, 200, 7}];
m = Length[signal];
ListLinePlot[signal];
ft = Fourier[signal, FourierParameters -> {-1, -1}];
g1 = ListLinePlot[2 Abs[ft], PlotRange -> {All, {0, 3}}];
po = (2 Abs[ft])^2;
g2 = ListLinePlot[po, PlotRange -> {All, {0, 3}}, PlotStyle -> {Red}];
Show[g1, g2, PlotRange -> {All, {0, 3}}]
```

Figure 4.23: A Plot of Amplitude Spectrum and Power Spectrum.

4.5 The Inverse DFT

The DFT is perfect in the sense that it is lossless as well as reversible. Its inverse is given by Equation (4.3), duplicated here

$$x_k = \frac{1}{N} \sum_{n=0}^{N-1} X_n \cdot e^{2\pi i k n/N}, \qquad (4.3)$$

which shows that the Fourier matrix of the inverse transform is the inverse (up to a factor of N) of the Fourier matrix of the forward transform, Equation (4.2). This can also be checked by the following code

```
np = 7;
fm = Table[Exp[-2. Pi I k n/np], {n, 0, np - 1}, {k, 0, np - 1}];
ifm = Table[Exp[2. Pi I k n/np], {n, 0, np - 1}, {k, 0, np - 1}];
Chop[fm . ifm] // MatrixForm
```

Figure 4.24 is a simple example of the forward and inverse DFT. The signal, constructed on line 1, is a sum of two cosines, sampled at 100 points and plotted on line 3. The entire 100×100 DFT matrix is computed on line 4 and multiplied by the signal on line 5, where the result (the row of frequency coefficients) is saved in `dft`. The inverse DFT matrix is similarly computed on line 7 and multiplied by `dft` on line 8 where (1) only the real part of the product is saved, and (2) a small value of 25 is added to the result before it is saved in `idft`. Line 9 displays the inverse transform, and line 10 displays both the original signal and the inverse transform. The following comments are important:

- The value 25 added to `idft` on line 8 results in shifting the display of the inverse transform (the red part of the figure) up a bit, so the original signal (in blue) would be visible under it. Otherwise, the two curves are identical.

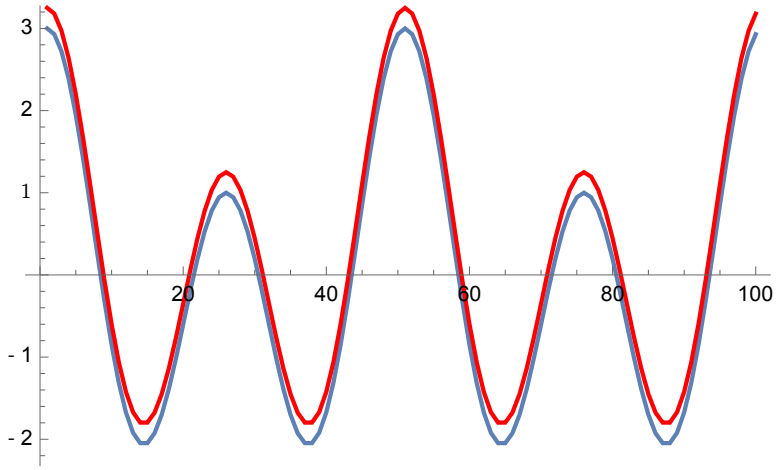
- The product `idftMat.dft` on line 8 should be real. All the imaginary parts should cancel out because the forward and inverse DFT matrices differ only by a minus sign in the complex exponent and are therefore inverses (up to a factor of `np`) of each other. However, because of the finite precision of computer arithmetic, very small imaginary values are often left after so many computations. These are illustrated in Figure 4.25, itself produced by line 11. Even though these tend to be on the order of 10^{-16} to 10^{-14}, they may significantly corrupt the inverse transform.

Equation (4.4), duplicated here

$$\cos k = \frac{e^{ik} + e^{-ik}}{2}, \quad \rightarrow \quad \cos(2\pi f t) = \frac{e^{i2\pi f t} + e^{-i2\pi f t}}{2}, \qquad (4.4)$$

explains why the sum of two complex exponentials, one positive and the other negative, results in a real-valued cosine.

Thus, the inverse DFT consists of a loop where in each iteration the row `dft` of (complex) frequency coefficients is multiplied by a (complex valued) row of the inverse Fourier matrix `idftMat` to create a (real valued) row of numbers. The rows of matrix




```
signal = Table[2 Cos[2 Pi 2 t] + Cos[2 Pi 1 t], {t, 0, 1.98, 0.02}];
np = Length[signal];
g1 = ListLinePlot[signal];
dftMat = Table[Exp[-2 Pi I k n/np], {n, 0, np-1}, {k, 0, np-1}];
dft = dftMat.signal;
ListPlot[2 Abs[dft]/np, Filling->Axis, PlotStyle->{Red, PointSize[0.015]}];
idftMat = Table[Exp[2 Pi I k n/np], {n, 0, np-1}, {k, 0, np-1}];
idft = Re[idftMat.dft] + 25;
g2 = ListLinePlot[idft/np, PlotStyle -> Red];
Show[g1, g2]
ListLinePlot[Im[idftMat.dft], PlotStyle -> Red]
```

Figure 4.24: DFT and Inverse DFT: A Simple Example.

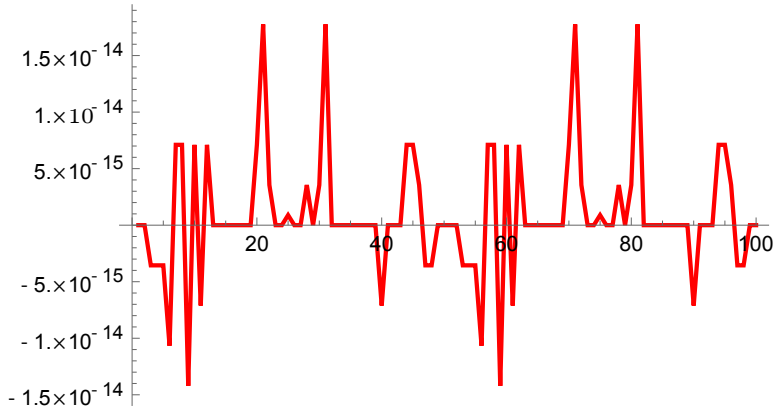


Figure 4.25: Traces of Imaginary Numbers.

`idftMat` contain complex sine waves of higher and higher frequencies, which is why each product of `dft` and a row of `idftMat` results in an array of numbers that contributes the next higher-frequency component to the final result `idft`, which is the original time series `signal`.

Another unexpected result of this relation between the forward and inverse Fourier matrices is that they are interchangeable. If we move the minus sign from the exponent on line 4 to the exponent of line 7, the graph of Figure 4.24 would remain the same.

The following question is at the heart of the inverse transform: What is the use of this? Why would we want to transform a set of data items only to transform it back to its original state? The forward Fourier transform yields the frequencies hidden in the original signal, and this information may be useful, but the applications of the inverse transform are not immediately clear.

It turns out that the inverse transform is useful if we perform it after we edit the frequency coefficients. One example of useful, practical editing is removing noise. Imagine trying to record sound under noisy conditions. A nearby motor automatically starts from time to time, corrupting parts of the sound that's being recorded. The result is a large array of audio samples, some of which are mostly due to noise. Often, noise is high-frequency sound, and this is easy to identify when the DFT is applied to the audio samples. Regions of noise would appear as large high-frequency coefficients, and those may be easy to recognize and set to zero. Once done, the inverse transform constructs a set of audio samples that may sound much better than the original. Such a process is referred to as bandstop filtering and can also be used, as in Figure 4.42, to clean digital images of high-frequency spatial "dirt" that came from the original image.

4.5.1 DFT Zero-Padding

The input to a DFT is an array of N samples from a signal, and the output of the transform is an array of N frequency coefficients, sometimes referred to as frequency bins. If N is small, a plot of the (absolute values) of the coefficients is jagged and rough. The technique of zero-padding a time-domain signal can be used to get more frequency bins and a smooth plot. We start with the definitions of the DFT and its inverse, Equations (4.2) and (4.3), respectively. In cases where the input x_n is infinite, we use the discrete-time Fourier transform (DTFT), where the frequency ω is a continuous variable

$$\tilde{X}(\omega) = \sum_{n=0}^{N-1} x_n e^{-in\omega}. \tag{4.6}$$

Comparing Equations (4.6) and (4.2) verifies that the DFT equals the DTFT when the latter is sampled at frequency values k that satisfy

$$X[k] = \tilde{X}(2\pi k/N), \quad k = 0, 1, \ldots, N-1.$$

Now imagine that we extend the original signal x_n from N points to L points by appending zeros, and then compute its length-L DFT that we denote by $X_L[l]$.

$$X_L[l] = \sum_{n=0}^{N-1} x_n e^{-i2\pi nl/L} = \tilde{X}(2\pi l/L), \quad l = 0, 1, \ldots, L-1. \tag{4.7}$$

It turns out that the length-L DFT equals a denser version of $\tilde{X}(\omega)$, one where ω consists of L instead of N frequency bins.

Next, we write Equation (4.6) with the help of Equation (4.3). This yields an expression for $\tilde{X}(\omega)$ in terms of the N frequency bins $X[k]$

$$\begin{aligned}\tilde{X}(\omega) &= \sum_{n=0}^{N-1} \frac{1}{N} \sum_{k=0}^{N-1} X[k] e^{i2\pi nk/N} e^{-in\omega} \\ &= \sum_{k=0}^{N-1} X[k] \frac{1}{N} \sum_{n=0}^{N-1} e^{-in(\omega - 2\pi k/N} \\ &= \sum_{k=0}^{N-1} X[k] \frac{1}{N} G(\omega - 2\pi k/N), \end{aligned} \tag{4.8}$$

Where the interpolation function $G(\omega)$ is defined by

$$G(\omega) = \frac{1}{N} e^{-in\omega}. \tag{4.9}$$

The last step is to substitute the value $2\pi l/L$ for ω in Equation (4.8) and compare the result to Equation (4.7), from which we can conclude that the length-L DFT $X_L[l]$ can be computed from the shorter, length-N DFT using the interpolation function $G(\omega)$ from Equation (4.9).

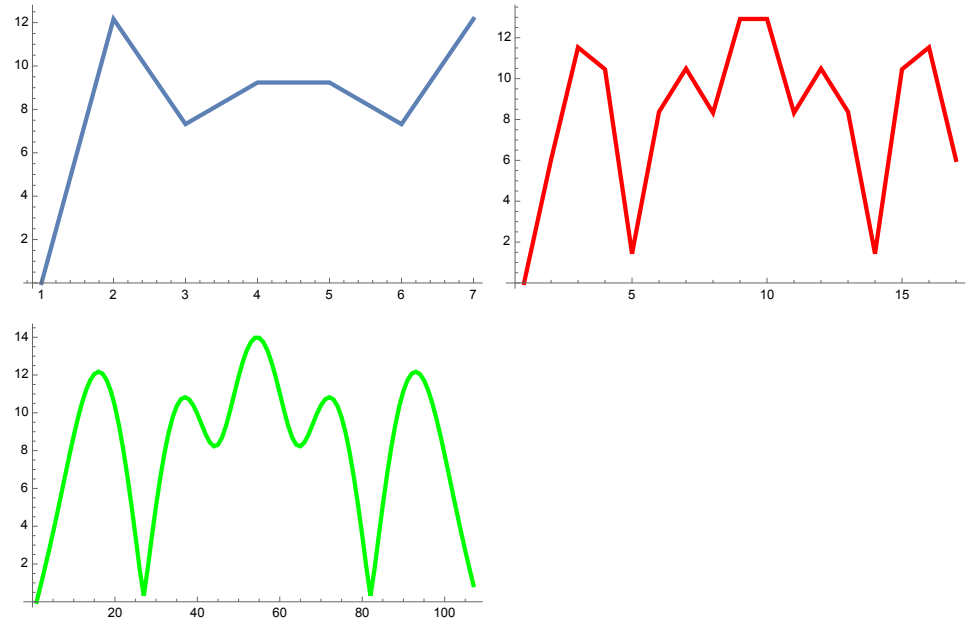
$$X_L(l) = \tilde{X}(2\pi l/L) = \sum_{k=0}^{N-1} X[k] G(2\pi l/L - 2\pi k/N).$$

This is the justification for the process of zero padding in the time domain. The advantage of this process is the increase in the number of frequency bins (and therefore in the frequency resolution) from N to L. The effect is a smoother curve of the frequency bins, as illustrated in Figure 4.26.

The figure starts with a 7-point signal `sig0` to which are appended first 10 zeros and then 100 zeros, ending up with signals `sig1` and `sig2`. It is obvious that the resulting curves of the frequency coefficients become smoother. Also note that the x-axes show the different number of points, 7, 17, and 107.

Frequency-domain zero padding. One of the fundamental principles of discrete signals is that zero padding in one domain results in an increased sampling rate in the other domain. Thus, zero padding in the frequency domain results in upsampling (increased sampling rate) in the time domain.

Zero padding in the time domain corresponds to frequency interpolation in the frequency domain, while zero padding in the frequency domain corresponds to temporal interpolation in the frequency domain. Both interpolations are done by the sinc function (more accurately, its normalized version). This all-important function and its applications to signal reconstruction by way of the sampling theorem are discussed in Subsection 5.2.




```
sig0 = {-4, 6, 0, 3, 1 , -2, -4};
ListLinePlot[Abs[Fourier[sig0, FourierParameters -> {1, 1}]],
 PlotRange -> Full]
sig1 = PadRight[sig0, 17];
ListLinePlot[Abs[Fourier[sig1, FourierParameters -> {1, 1}]],
 PlotRange -> Full, PlotStyle -> Red]
sig2 = PadRight[sig0, 107];
ListLinePlot[Abs[Fourier[sig2, FourierParameters -> {1, 1}]],
 PlotRange -> Full, PlotStyle -> Green]
```

Figure 4.26: Zero Padding in the Time Domain

4.5.2 Aliasing

The Sampling Theorem, Section 5.1, states that we can scan a time signal, convert it into samples, and still be able to reconstruct the original, analog signal perfectly if the sampling rate is at least twice the maximum frequency of the signal. This section deals with aliasing, a phenomenon caused by insufficient sampling, also referred to as undersampling or sub-sampling. The following are two causes of aliasing:

- The sampling rate is exactly twice the maximum frequency of the signal, but it is insufficient because the signal happens to be sampled at certain strategic points, as illustrated in Figure 4.27. The figure shows a pure sine wave sampled at twice its frequency. Each period contains exactly two samples, but the samples are taken at such points that their reconstruction is very different from the original signal.

- A signal is sampled at a little more than twice its maximum frequency, but because

of vibrations, temperature gradients, or slight inaccuracies in the sampling device, small fluctuations are introduced in the analog signal and are big enough to slightly increase its theoretical maximum frequency, thereby resulting in undersampling.

In both cases, the solution is to rescan the signal, if possible, at a higher sampling rate. Figure 4.27 shows how aliasing may cause the reconstruction process to end up with a signal (in green) that is completely different from the original signal (in blue) and always has lower frequency. This effect of low frequency aliasing is so noticeable that Marc Levoy, in his outstanding lectures on photography, defines aliasing as high frequencies masquerading as low frequencies due to insufficiently closely-spaced samples (lecture 7 of [Levoy 16]). See also Exercise 5.1 for examples of aliasing.

Here is what wikipedia has to say about aliasing "In signal processing and related fields, aliasing is an effect that causes different signals to become indistinguishable (or aliases of one another) when sampled." Thus, when a sound wave is sampled at a rate that is too low, the reconstructed sound is different from the original sound and it features low frequencies. When an image is sampled (scanned or photographed) at a low rate, its reconstruction also features low pixel frequencies. Such frequencies are associated with important image features, which is why aliasing causes significant distortions.

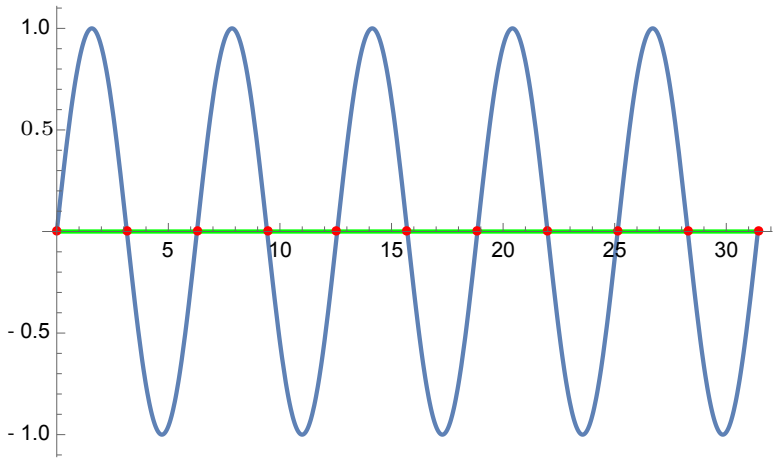
A good example of avoiding aliasing is sound sampling for CDs and DVDs. The range of human hearing is typically from 16–20 Hz to 20,000–22,000 Hz, depending on the person and on age. When sound is digitized at high fidelity, it should therefore be sampled at a little over the Nyquist rate of $2\times 22{,}000 = 44{,}000$ Hz. This is why high-quality digital sound is based on (at least) a 44,100-Hz sampling rate. Anything lower than this rate may result in distortions, while higher sampling rates do not improve the reconstruction (playback) of the sound. We can consider the sampling rate of 44,100 Hz a lowpass filter, since it effectively removes all the frequencies above 22,000 Hz.

Signal stationarity. A time signal can be stationary or non-stationary. A typical dictionary definition of the term stationary is "1. fixed in a station, course, or mode: immobile. 2. Unchanging in condition; a stationary population." This term is often confused with stationery, whose meaning is "writing and other office materials."

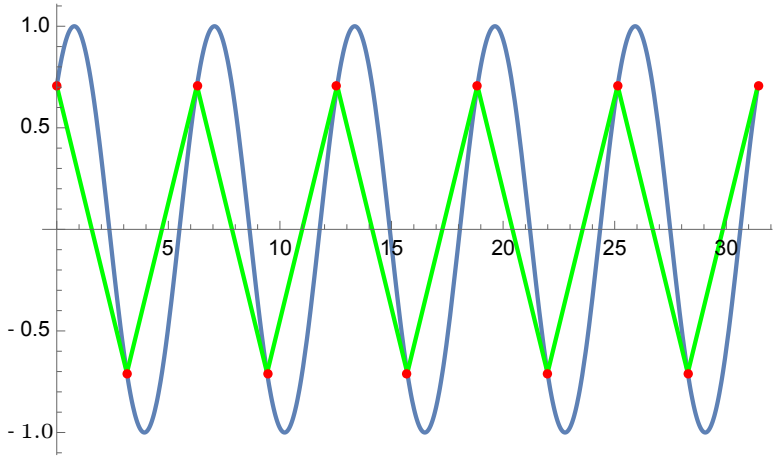
The stationarity of a signal is often ambiguous, but this term is nevertheless important because of its use in interpreting the results of the Fourier transform of a signal. Up to now, we have dealt with a few features of a time signal such as its amplitude, frequency, and phase, but there are many more statistical features that are important when trying to interpret the results of a transform. Examples are mean (the average or expected value of a signal), variance (the distribution of a signal around its mean), kurtosis (the shape of a statistical distribution), dominant frequency, spectral shape, and phase stability.

Those who work with signals often have to compute some features of a given signal, and the concept of stationarity arises when we try to compute features of just part of a signal. Imagine placing a window on top of a signal and measuring the mean of the signal inside the window. We then move the window to other regions of the signal and may also change its width in the process. If the mean of the signal is the same in every window, the signal is said to be mean-stationary.

Thus, in the field of digital signal processing, signal stationarity is defined as follows:




```
phi = Pi/4;
g1 = Plot[Sin[t], {t, 0, 10 Pi}];
pn = Table[{t, Sin[t]}, {t, 0, 10 Pi, Pi}];
g2 = ListLinePlot[pn, PlotStyle -> Green];
g3 = ListPlot[pn, PlotStyle -> Red];
Show[g1, g2, g3]
```




```
phi = Pi/4;
g1 = Plot[Sin[t + phi], {t, 0, 10 Pi}];
pn = Table[{t, Sin[t + phi]}, {t, 0, 10 Pi, Pi}];
g2 = ListLinePlot[pn, PlotStyle -> Green];
g3 = ListPlot[pn, PlotStyle -> Red];
Show[g1, g2, g3]
```

Figure 4.27: Aliasing Caused by Slightly Low Sampling Rates

A signal is stationary when its statistical characteristics do not significantly vary over time.

The top part of Figure 4.18 is a good example of a mean non-stationary signal, because its value rises from -40 to $+40$ over its 200-point life. Similarly, the signal on the bottom of the same figure is amplitude non-stationary because its amplitude is very high at its two extremes. Finally, the top signal of Figure 4.16 is non-stationary in both its amplitude and frequency, the latter because of the random contributions.

In spite of its importance, stationarity is an ambiguous concept because of the following:

- A signal may be stationary in several features and non-stationary in other features.
- When moving windows over a signal to measure a certain feature, the measurements may differ slightly. We therefore need a measure of uncertainty, a threshold or leniency, in deciding whether two windows should be considered equal.
- The complexity of a signal may lead to wrong measurements of a feature. The frequency of a signal may look very similar in two windows, but when we zoom on the signal, small frequency details that had been hidden earlier may now be visible and indicate that the windows feature different frequencies.

Because of these ambiguities, those who measure features of signals must use their experience to determine equality of windows and the stationarity of a signal. This is an important point because non-stationarities in a time signal can introduce distortions in its Fourier transform and make it more difficult to interpret. The following examples are illustrative. Figures 4.28 and 4.29 show an amplitude non-stationary signal and its transform. The top part of the latter figure is the time signal (a product of two sine waves with frequencies 1/2 and 3), the middle part is its Fourier transform (already looks weird), and the bottom part is a vertical enlargement of the x-axis of the transform. It is clear that most of the frequency coefficients are concentrated around 3 Hz, but lots of the energy is spread all over the spectrum, with many small coefficients.

```
sr = 30;
f1 = 1/2; f2 = 3;
s1[t_] := 0.7 + 0.5 Sin[2 Pi f1 t];
s2[t_] := Sin[2 Pi f2 t];
Plot[s1[t] s2[t], {t, 0, 3}]
pn = Table[s1[t] s2[t] , {t, 0, sr, 1/sr}];
inc = sr/Length[pn];(*increment*)
freq =  Table[f, {f, 0, sr - inc, inc}]; (* x-axis of plot *)
ListPlot[ Transpose[{freq, 2 Abs[Fourier[pn, FourierParameters -> {-1, 1}]]/pn}],
 PlotRange -> All, Filling -> Axis, PlotStyle -> {Red, PointSize[0.001]}]
```

Figure 4.28: Mathematica Code for Figure 4.29

Figure 4.30 illustrates a frequency non-stationary signal and its transform. The signal is a cosine whose frequency is determined by a sine wave. This signal features increased frequency during its short window (it is a chirp), which is why it is reasonable to expect frequency coefficients of about the same size at higher and higher frequencies.

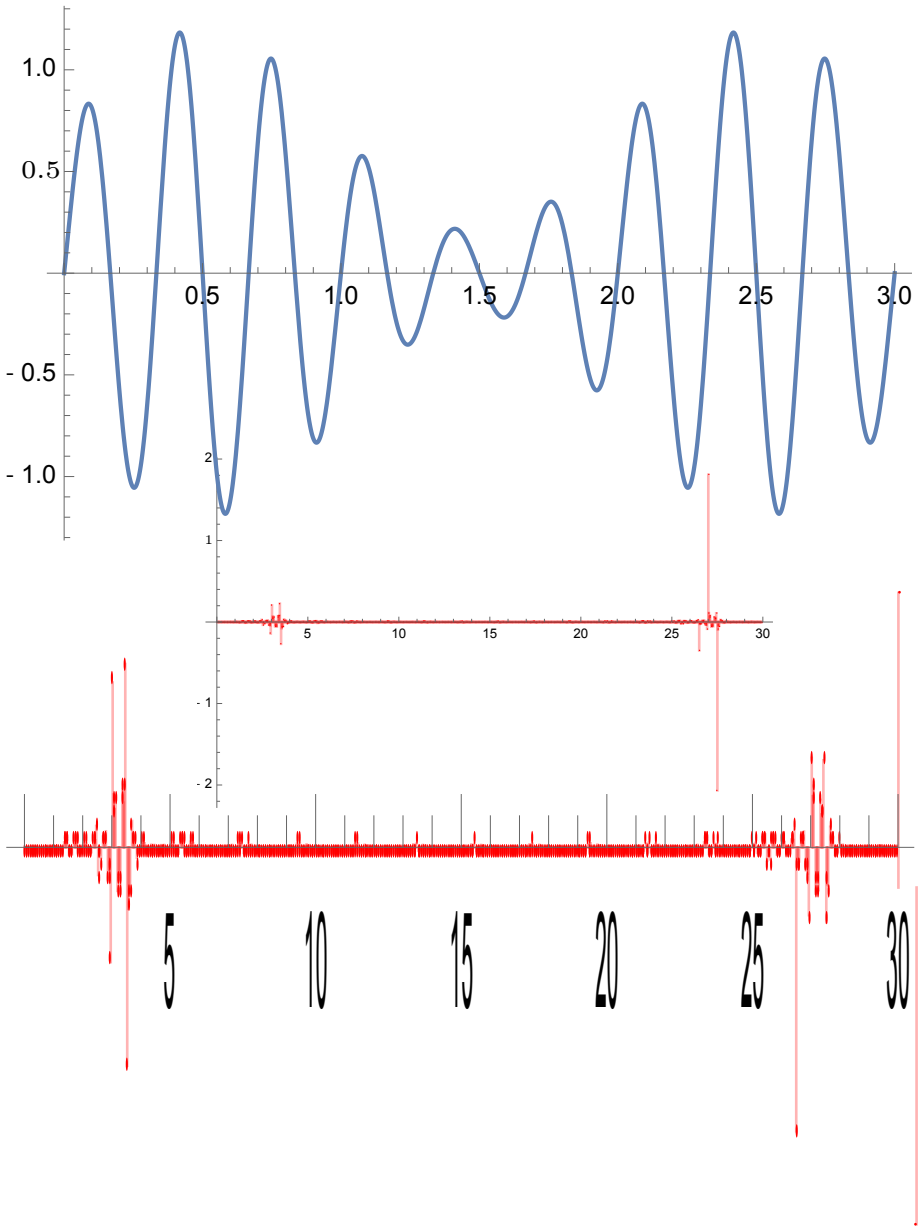


Figure 4.29: An Amplitude Non-Stationary Signal and its Transform

What the figure actually shows is a mixture of many coefficients with magnitudes that seem random and defy interpretation.

These examples are typical and they suggest that a time signal should be checked for stationarity before it is Fourier transformed.

4.6 The Two-Dimensional DFT

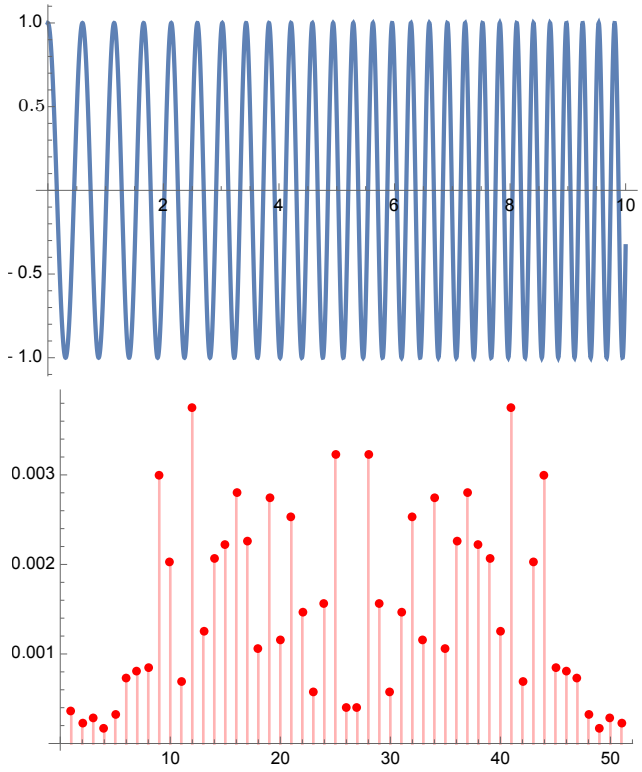
We live in the digital age, and are surrounded by digital images, which is why the two-dimensional DFT is relevant to our discussion and is now introduced. Once we understand this version of the transform and witness its power, we realize the following, surprising fact; all visual patterns, in fact the entire visual world, can be represented as a sum of two-dimensional sine waves, which resemble corrugated two-dimensional metal or plastic sheets. That's all. Nothing else is needed to mathematically represent any of the images around us, even those that look random and are far removed from any wavy form.

This conclusion is unintuitive and goes against our everyday's experience. How can a large symphony—with its many variations of volume, frequency, and tone color—be represented as the sum of periodic, regular, one-dimensional sine waves? Even worse. How can an oil painting by a master—with its complex irregular gradations of shade and color, attention to detail, and variance of brightness—be represented by such a sum of regular, periodic two-dimensional sine waves? It is like declaring that the world is nothing but waves! It is no wonder that Fourier's ideas faced stiff resistance during his lifetime.

The waves that constitute a Fourier sum can have different amplitudes (corresponding to the brightness of the image), frequencies (according to the amount of detail in the scene), and phases (to increase the fitness of the waves to the shape of the image). They can also "run" in different directions. The furrows between the crests may point in the north-south, east-west, or in any other direction. The infinite richness of our visual world has been reduced in this way to a bunch of corrugated plastic sheets. Poets and nature lovers may resent and deny this idea, but mathematicians love it and engineers use it in practice.

Many images and three-dimensional objects have colors, and color can also be implemented in Fourier's sine waves. A colored wave may be a set of three component waves with the same frequency and with different amplitudes, which correspond to the three RGB color components of the wave. Thus, the first and second component waves (corresponding to the red and green RGB colors) of a yellow wave would have a large amplitude, while the waves of the third component (for RGB blue) would have zero amplitude.

We already know, from the one-dimensional case, that the low-frequency waves in the two-dimensional Fourier transform of an image correspond to the important image features, while the high-frequency waves are mainly responsible for the fine details of the image, as well as for any edges in it. As a result, an engineer who examines the transform of an image may claim "look, there are so many high frequencies here," as his way of saying "this image has much fine detail and may have sharp edges."




```
f[t_] := 2 + 2 Sin[t/12];
Plot[Cos[5 f[t] t], {t, 0, 10}] (* ft[t] is the frequency *)
tb = Table[Cos[f[t] t], {t, 0, 50}];
ListPlot[Abs[Fourier[tb, FourierParameters -> {-1, 1}]/Length[tb]],
 PlotRange -> All, Filling -> Axis, PlotStyle -> {Red, PointSize[0.015]}]
```

Figure 4.30: A Frequency Non-Stationary Signal and its Transform

◇ **Exercise 4.3:** The presence of many high-frequency waves in the transform of an image indicates small details, but why does it also imply the existence of sharp edges?

The two-dimensional DFT is the product of two one-dimensional DFTs. A glance at Equation (4.2) makes it easy to see why such a product has the form

$$\begin{aligned} X_{k,l} &= \sum_{n=0}^{N-1}\sum_{m=0}^{M-1} x_{n,m}\, e^{-2\pi i k\frac{n}{N}} e^{-2\pi i l\frac{m}{M}} \qquad (4.10)\\ &= \sum_{n=0}^{N-1}\sum_{m=0}^{M-1} x_{n,m}\big[\cos(2\pi kn/N) - i\cdot\sin(2\pi kn/N)\big]\big[\cos(2\pi lm/M) - i\cdot\sin(2\pi lm/M)\big],\\ &\text{for } k\in(0,N-1) \text{ and } l\in(0,M-1), \end{aligned}$$

Given a digital image `IMG` of N rows and M columns of pixel values $x_{n,m}$, Equation (4.10) transforms it into a matrix X of $N\times M$ complex numbers $a+ib$ which are the DFT transform coefficients, and whose magnitudes $\sqrt{a^2+b^2}$ are frequency coefficients (amplitudes of two-dimensional sinusoids of frequencies k and l) and their angles $\arctan(b/a)$ are the phases of those sinusoids.

The inverse two-dimensional discrete Fourier transform is given by Equation (4.11), which should be compared to the straight transform, Equation (4.10)

$$x_{n,m} = \frac{1}{N\cdot M}\sum_{k=0}^{N-1}\sum_{l=0}^{M-1} X_{k,l}\, e^{2\pi i n\frac{k}{N}} e^{2\pi i m\frac{l}{M}}. \qquad (4.11)$$

It is also useful to interpret this transform as the sum of two matrices, both with real numbers. The first one, with the cosines, contains information about the amplitudes of the Fourier transform components, and the second matrix, with the sines, contains phase information for the coefficients. The second matrix is multiplied by the imaginary i which can now be ignored.

An alternative way of looking at this process is to say that given a digital image `IMG`, the double sum of Equation (4.10) computes its two-dimensional DFT by (1) computing the one-dimensional DFT of each column, and then (2) computing the one-dimensional DFT of each row of the result of step 1. The result of step 1 is a matrix of frequency coefficients where the frequency increases as we go down the rows. The top row is the DC offsets. The result of step 2 is again a matrix of frequency coefficients where the low-frequency coefficients, the ones that correspond to dominant image features, are located in the corners, and the center elements are the coefficients that correspond to the Nyquist frequency. The following paragraphs explain why the dominant matrix elements are located in the corners.

We apply Equation (4.10) to six important corner elements of matrix X, starting with the top-left element

$$X_{0,0} = \sum_{n=0}^{N-1}\sum_{m=0}^{M-1} x_{n,m} = N\cdot M\cdot\bar{x}_{n,m}.$$

This element of X is simply the sum of all the pixels of the image, but we can also interpret it as the average value $\bar{x}_{n,m}$ of all the pixels, times the total number of pixels, which explains why this element is the dominant value in the amplitude matrix of any two-dimensional DFT. In addition, the following elements of $X_{k,l}$ also tend to be much bigger than the rest:

The second element of the top row of X is

$$\begin{aligned} X_{0,1} &= \sum_{n=0}^{N-1}\sum_{m=0}^{M-1} x_{n,m}\big[\cos(0) - i\cdot\sin(0)\big]\big[\cos(2\pi m/M) - i\cdot\sin(2\pi m/M)\big] \\ &= \sum_{n=0}^{N-1}\left[\sum_{m=0}^{M-1} x_{n,m}\big[\cos(2\pi m/M) - i\cdot\sin(2\pi m/M)\big]\right]. \end{aligned}$$

This element is therefore the sum of N one-dimensional sinusoids, each of which is the lowest-frequency Fourier component of the M pixels of one row of `IMG`. It is big, because the lowest-frequency Fourier components contain the significant image information. Similarly, element $X_{1,0}$ (the first element of the second row of X) is the sum of M one-dimensional sinusoids, each of which is the lowest-frequency Fourier component of the N pixels of a column of `IMG`.

Next, we examine the top-right and bottom-left elements of X. The former is the last element of the top row of X

$$\begin{aligned} &X_{0,M-1} \\ &= \sum_{n=0}^{N-1}\left[\sum_{m=0}^{M-1} x_{n,m}\big[\cos(0) - i\cdot\sin(0)\big]\big[\cos(2\pi m(M-1)/M) - i\cdot\sin(2\pi m(M-1)/M)\big]\right] \\ &= \sum_{n=0}^{N-1}\left[\sum_{m=0}^{M-1} x_{n,m}\big[\cos(2\pi m(M-1)/M) - i\cdot\sin(2\pi m(M-1)/M)\big]\right]. \end{aligned}$$

Similar to $X_{0,1}$, element $X_{0,M-1}$ is the sum of N one-dimensional sinusoids each of which is the *highest*-frequency Fourier component of the M pixels of a row of `IMG`. Finally, element $X_{N-1,0}$ is the sum of M one-dimensional sinusoids each of which is the *highest*-frequency Fourier component of the N pixels of a column of `IMG`.

High-frequency Fourier components of an image contain the details of the image, which is why they tend to be smaller the higher the frequency. However, the highest-frequency components are an exception. They are large, because the DFT has the following periodicity property. In the one-dimensional case, Equation (4.2), each coefficient X_k satisfies $X_k = X_{k+pN}$ for any integer p. Similarly, in the two-dimensional DFT, Equation (4.10), $X_{k,l} = X_{k+p1\cdot N,l} = X_{k,l+p2\cdot M} = X_{k+p1\cdot N,l+p2\cdot M}$, for any integers $p1$ and $p2$.

Periodicity follows directly from the definition of the DFT, and it is illustrated here

for the one-dimensional case, Equation (4.2)

$$X_{k+pN} = \sum_{n=0}^{N-1} x_n \cdot e^{-2\pi i \frac{k+pN}{N} n} = \sum_{n=0}^{N-1} x_n \cdot e^{-2\pi i \frac{k}{N} n} \underbrace{e^{-2\pi i \frac{pN}{N} n}}_{=1}$$
$$= \sum_{n=0}^{N-1} x_n \cdot e^{-2\pi i \frac{k}{N} n} = X_k.$$

(The equality $e^{-2\pi p\, i\, n} = 1$ for any integer p is based on the periodicities of the sine and cosine.)

Finally, element $X_{N-1,M-1}$, located at the bottom-right corner of matrix X, equals

$$X_{N-1,M-1} = \sum_{n=0}^{N-1} \sum_{m=0}^{M-1} x_{n,m}\; e^{-2\pi i n \frac{N-1}{N}} e^{-2\pi i m \frac{M-1}{M}}$$
$$= \sum_{n=0}^{N-1} \sum_{m=0}^{M-1} x_{n,m} \big[\cos(2\pi n(N-1)/N) - i \cdot \sin(2\pi n(N-1)/N)\big]$$
$$\big[\cos(2\pi m(M-1)/M) - i \cdot \sin(2\pi m(M-1)/M)\big].$$

This element is also relatively large, because of periodicity.

These six dominant elements increase the dynamic range (the difference between large and small matrix elements) of the two-dimensional DFT significantly, which is why a grayscale image of matrix X often looks dark, with only very small bright areas in the four corners.

Example. A quick two-dimensional DFT is performed on the elements (real numbers) of a small, 7×7 matrix listed in Figure 4.32 (which also includes all the results). Figure 4.31 is a listing of the Mathematica code.

```
ml = {{12, 23, 41, 49, 41, 23, 12}, {23, 24, 56, 72, 56, 22, 73},
 {52, 25, 75, 92, 75, 58, 32}, {42, 24, 85, 10, 95, 57, 12},
 {39, 23, 15, 12, 75, 50, 32}, {23, 24, 26, 22, 26, 20, 23},
 {12, 22, 41, 79, 41, 83, 12}};
Fourier[ml, FourierParameters->{1,-1}]; (*The 2D transform of ml*)
col = Table[Table[0, {7}], {7}];
For[fq = 1, fq <= 7, fq++, (*Loop over the columns*)
 col[[;; ,fq]] = Fourier[ml[[;; ,fq]], FourierParameters->{1,-1}]]
NumberForm[Re[col], {5, 2}];
MatrixPlot[Re[col], ColorFunction -> "GrayTones"]
NumberForm[Im[col], {5, 2}];
row = Table[Table[0, {7}], {7}];
For[fq = 1, fq <= 7, fq++, (*Loop over the rows*)
 row[[fq]] = Fourier[col[[fq]], FourierParameters->{1,-1}]]
NumberForm[Re[row], {5, 2}];
MatrixPlot[Abs[row], ColorFunction -> "GrayTones"]
NumberForm[Im[row], {5, 2}];
```

Figure 4.31: Mathematica Listing of a 2D Fourier Transform

The code of Figure 4.31 is straightforward. Lines 1 through 4 are the original data. Line 5 is the DFT, where the computations are done by the Mathematica `Fourier` command. The command recognizes `ml` as a two-dimensional matrix and computes the DFT accordingly. The remainder of the listing illustrates how the same transform is done explicitly in two loops, for the columns and for the rows, respectively. Line 6 prepares matrix `col` to receive the results of step 1 of the DFT, and the loop of lines 7–8 computes the transforms of the seven individual columns. Line 9 displays the results as five-digit numbers, with two digits to the right of the decimal point. Line 10 prepares a graphics representation of the real part of matrix `col`, which is displayed on the bottom-left of Figure 4.32. Finally, lines 12–17, similar to lines 6–11, perform step 2 on the rows of matrix `col` and store the results in matrix `row`.

203.00	165.00	339.00	336.00	409.00	313.00	196.00		0.00	0.00	0.00	0.00	0.00	0.00	0.00
-55.85	-1.57	-11.09	97.96	-74.16	-25.29	13.12		-38.17	-2.97	-89.87	-61.90	-68.18	7.61	-47.79
-12.86	-2.08	-9.23	-73.59	34.41	-3.93	-29.03		4.20	-0.73	61.36	35.63	22.27	81.43	-71.20
9.20	1.65	-5.67	-20.86	-21.25	-46.78	-40.08	$+i$	14.98	-1.06	-36.44	59.72	12.30	49.35	0.07
9.20	1.65	-5.67	-20.86	-21.25	-46.78	-40.08		-14.98	1.06	36.44	-59.72	-12.30	-49.35	-0.07
-12.86	-2.08	-9.23	-73.59	34.41	-3.93	-29.03		-4.20	0.73	-61.36	-35.63	-22.27	-81.43	71.20
-55.85	-1.57	-11.09	97.96	-74.16	-25.29	13.12		38.17	2.97	89.87	61.90	68.18	-7.61	47.79

Real part Imaginary part

Step 1. Transforming the rows

1961.00	-388.23	-0.26	118.49	118.49	-0.26	-388.23		0.00	30.56	-15.57	104.95	-104.95	15.57	-30.56
-56.89	-119.26	70.29	7.54	-196.00	-91.88	-4.72		-301.28	-11.36	121.18	-165.12	135.54	-188.91	142.72
-96.32	47.30	48.45	74.98	-43.60	-85.47	-35.35		132.97	-93.63	-185.36	234.62	55.71	40.68	-155.56
-123.80	-29.10	37.81	133.31	-92.31	39.70	98.81		98.91	-126.27	25.92	21.55	-5.76	71.01	19.47
-123.80	98.81	39.70	-92.31	133.31	37.81	-29.10	$+i$	-98.91	-19.47	-71.01	5.76	-21.55	-25.92	126.27
-96.32	-35.35	-85.47	-43.60	74.98	48.45	47.30		-132.97	155.56	-40.68	-55.71	-234.62	185.36	93.63
-56.89	-4.72	-91.88	-196.00	7.54	70.29	-119.26		301.28	-142.72	188.91	-135.54	165.12	-121.18	11.36

Real part Imaginary part

Step 2. Transforming the results of step 1

12	23	41	49	41	23	12
23	24	56	72	56	22	73
52	25	75	92	75	58	32
42	24	85	10	95	57	12
39	23	15	12	75	50	32
23	24	26	22	26	20	23
12	22	41	79	41	83	12

Original data

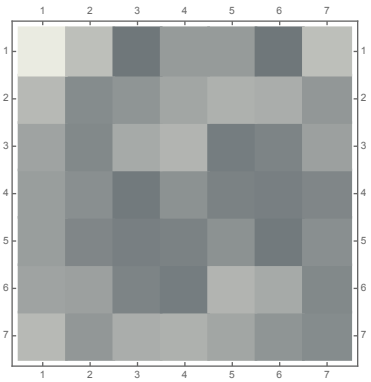


Figure 4.32: The results of Figure 4.31.

The low-frequency Fourier coefficients are located in the corners of the final matrix of Figure 4.32 and the highest frequency coefficients, which correspond to the Nyquist frequency (Section 5.1), are at the center. Those who use the DFT often, have long ago

discovered that the real part of the two-dimensional DFT matrix (`row` in our example) becomes visually clearer when it is rearranged such that it features the four dominant corner elements at the center and the remaining high-frequency coefficients closer and closer to the corners. Figure 4.33 illustrates the simple shifting rules that achieve this useful result. The main advantage of shifting the coefficients matrix is that this brings all its dominant elements together, around its center. Once this is done, it is possible to get a good idea of the original image by inverse transforming a small area around the center of the matrix. To put this in perspective, given a $256 \times 256 = 65536$-pixel image, it is enough to isolate its 7×7 central area and inverse transform it. This quick operation, performed on only 0.075% of the image, normally produces a recognizable, albeit blurred version of the original image.

◇ **Exercise 4.5:** Implement the shifting rules in Mathematica and apply them to the 7×7 matrix `row` of our example (the real part of step 2).

In addition to the shifting, the dynamic range of X can be reduced by scaling its elements to their logarithms according to $X_{k,l} \leftarrow c \cdot \log(1 + X_{k,l})$ where c is often chosen as 1. Figure 4.38 is an example of this practice.

Figure 4.34 shows a 7×7 real matrix together with four images produced by its two-dimensional DFT. The image labeled "Amplitude Spectrum, Unshifted" is the real part of the DFT, the image labeled "Phase Spectrum" is the imaginary part of the DFT, and the image labeled "Transformed and Shifted" is the rearranged version of the real part of the DFT. The red arrow in this latter part shows how to interpret each of the DFT coefficients (the small squares). The distance r of the coefficient from the center of the image corresponds to the frequency of the coefficient; closer to the center implies lower frequencies. The angle θ indicates the direction of the Fourier sinusoids in the vicinity of the coefficient. The Matlab code that generated the figure is also listed.

A few words about the Nyquist frequency of an image. Figure 4.35 demonstrates how we can intuitively associate pixel values in an image with waves. We can visualize how a string of adjacent pixels can be used to compute a smooth curve that oscillates according to the values of adjacent pixels. It is also possible to associate such a string of pixels with several smooth curves, all having different frequencies but passing close to all the pixels. However, the frequencies of such waves are limited. The maximum frequency of a wave that is defined by a string of pixels is one cycle per a pair of adjacent pixels, i.e., a pair of adjacent pixels is the smallest image unit that can be associated with the frequency of a wave. Therefore, if this maximum frequency is ν, then $\nu/2$ is the Nyquist frequency of the image.

Figure 4.36 illustrates the relation between an image in image space and the shifted version of its discrete Fourier transform. Each of the two original images has dominant lines oriented in a certain direction, and it is possible to see how the dominant features in the corresponding shifted transforms (located under the original image and to the right) are oriented in the perpendicular directions.

Figure 4.37 is similar to Figure 4.34 and shows the results of transforming a small 256×256-pixel image. The original image shown in the figure is in color, but the transform was computed on a grayscale version of it. It is clear that the real coefficient matrix (amplitude spectrum unshifted) is complicated and cannot be easily interpreted (examine carefully its corners). However, there is no need to examine individual transform

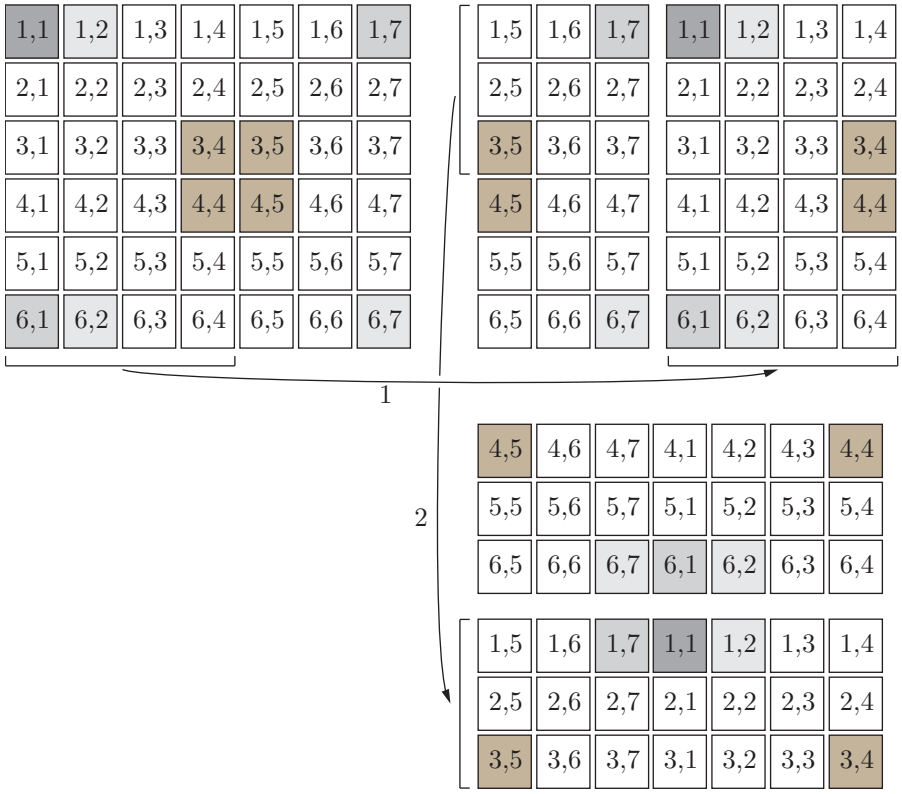
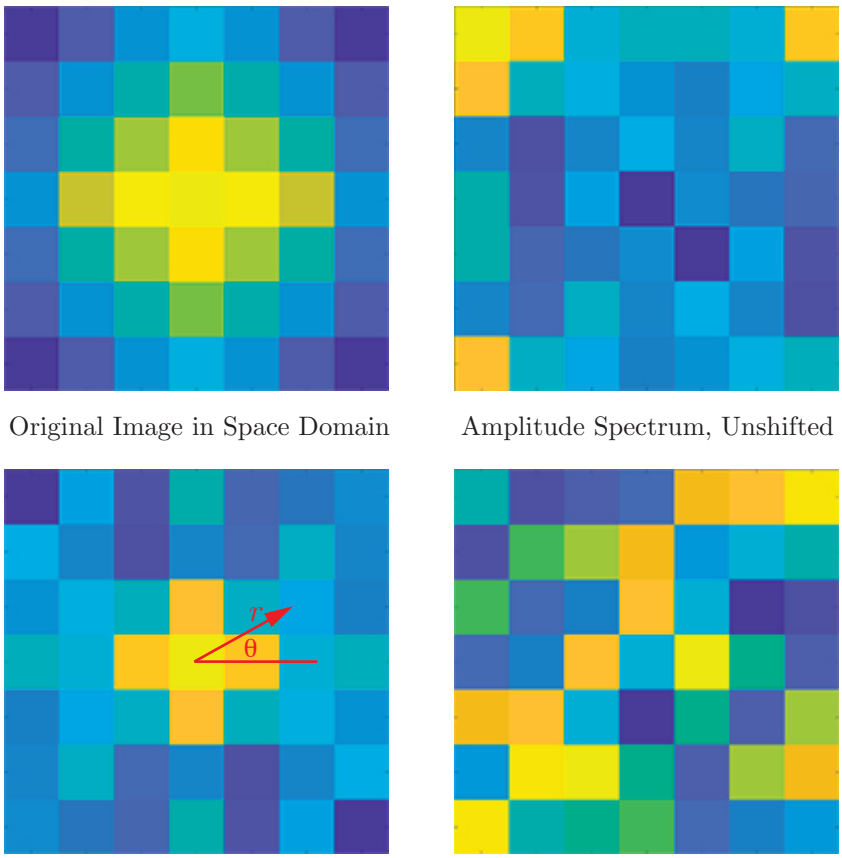


Figure 4.33: Shifting the 2D DFT Amplitudes.

The rules for shifting the DFT amplitudes are simple and are illustrated by the 6×7 array of Figure 4.33, where successive positions of several array elements are indicated by their background colors. First, swap the left and right halves of the array. If the length of the rows is odd, then include the middle element in the left half. Next, swap the top and bottom of the result of the previous step, following the same rule if the columns have an odd length. This will bring the corner elements (in gray) to the center, while also moving the center elements (in brown) to the new corners.

These rules can be stated compactly as follows: Swap the 1st and 4th quadrants and swap the 2nd and 3rd quadrants of the matrix. The inverse shift is done by following the same rule, and this is true even of the dimensions of the matrix are odd, which results in quadrants of slightly different sizes.

◇ **Exercise 4.4:** Prove this claim.



Original Image in Space Domain | Amplitude Spectrum, Unshifted

Transformed and Shifted | Phase Spectrum

```
image =[12 23 41 49 41 23 12; 23 42 56 72 56 42 23; 32 58 75 92 75 58 32;
    42 77 95 100 95 77 42; 32 58 75 92 75 58 32; 23 42 56 72 56 42 23;
    12 23 41 49 41 23 12];
FT = fftshift(fft2(image));
imagesc(abs(log2(FT))) % real part (amplitude)
%imagesc(imag(FT)) % imaginary part (phase)
axis image
```

Figure 4.34: The 2D DFT of the 7×7 Image.

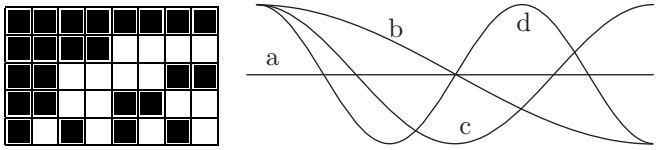


Figure 4.35: Image Frequencies.

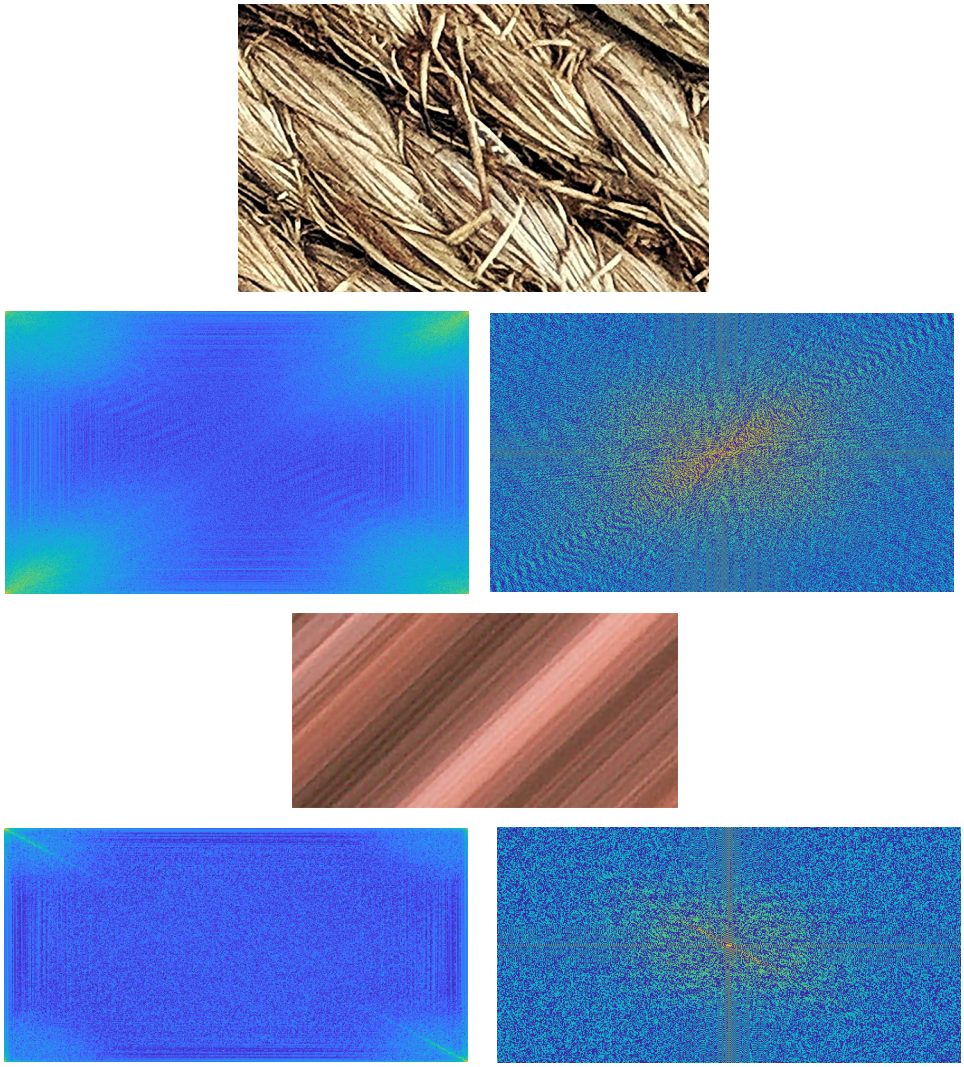


Figure 4.36: Directions of Sinusoidals.

coefficients, because the usefulness of the DFT to image processing lies in various statis-

tical operations on the DFT coefficients, not in manual changes to specific coefficients.

Figure 4.38 is similar to 4.37 but is computed by Mathematica. The following should be noted about the code. Line 2 imports the image. The entire path of the image is specified, and variable i becomes an image, not the same as a numeric matrix. Converting it to such a matrix is done on line 6. Line 3 extracts a 256×256-part of the image. Lines 9–14 perform the shifting of rows and columns, as is also illustrated in Figure Ans.4. Logarithmic versions of the matrices are displayed because there is a big difference between the largest elements (about 29000 at the top-left corner) and the rest of the matrices. The logarithm reduces that difference and enhances the small details of the images.

The two-dimensional DFT results in matrices for the amplitudes and phases of the Fourier coefficients. The amplitude matrix is easy to understand. A large amplitude A implies that the sinusoid associated with A contributes more to the reconstruction of the image. Reference [Live FT 14] is a video that illustrates the amplitude matrices for a variety of images. The phases of the second matrix are harder to interpret but are as important. We know that the phase of a sinusoid is a measure of its displacement from its "natural" position (the position for zero phase), which is why we can state that the phase matrix contains information about where discernible objects are located in the image. This is similar to saying that the phase matrix can help in determining edges of the image.

An important application of the DFT to image processing is filtering, which is done by modifying (amplifying or attenuating) the amplitudes of certain transform coefficients. while maintaining the same average over all the transform coefficients.

Low-pass filtering is done by attenuating the high-frequency transform coefficients (Figure 4.40a). This results in image blurring, but the important image features are retained. High-pass filtering is done by attenuating the low-frequency coefficients (Figure 4.40b). This produces the opposite result, it sharpens the image, while losing some of its major features.

Figure 4.39 is an example of blurring an image by clearing the elements of the two-dimensional Fourier coefficients matrix, except a small $(2\text{par}+1) \times (2\text{par}+1)$ square in the middle. The matrix has to be shifted before this operation, shifted back following the operation, and then passed through the inverse DFT. Figure 4.41 is the opposite example, where the dominant, low-frequency transform coefficients which are located at the corners of the DFT matrix, are set to zero. The result is an image of the edges of the original. It has the original details, but is lacking its main features.

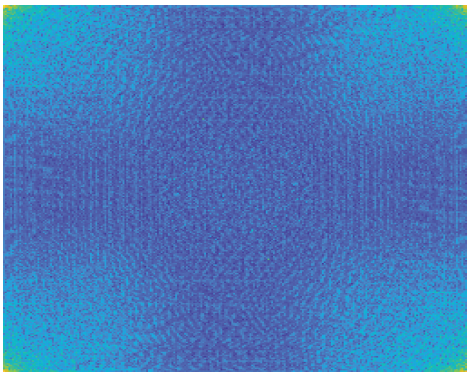
◇ **Exercise 4.6:** Use the Mathematica command `UnitBox` to come up with a sophisticated version of function `lpFiltr` of Figure 4.39. Your function should be able to generate matrices of 0's and 1's for low-pass, high-pass, and bandpass filters.

The DFT can also be used to clean "dirty" or noisy images from the effects of a blurry background, a background full of regular dithering (as is often found in old photographs), the remains of old fingerprints on a picture, or annoying dark or bright bands on an image as a result of combining partial scans of a large image, as is common with lunar and other astronomical images.

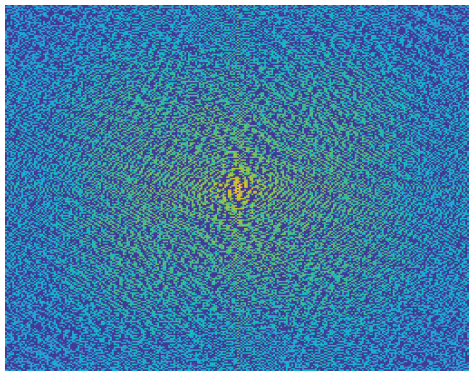
Figure 4.42 illustrates how the DFT of a noisy image may feature small bright regions that are the result of the noise. In such a case, the image can be cleaned

Original image
256x256 pixels

Amplitude Spectrum
Unshifted



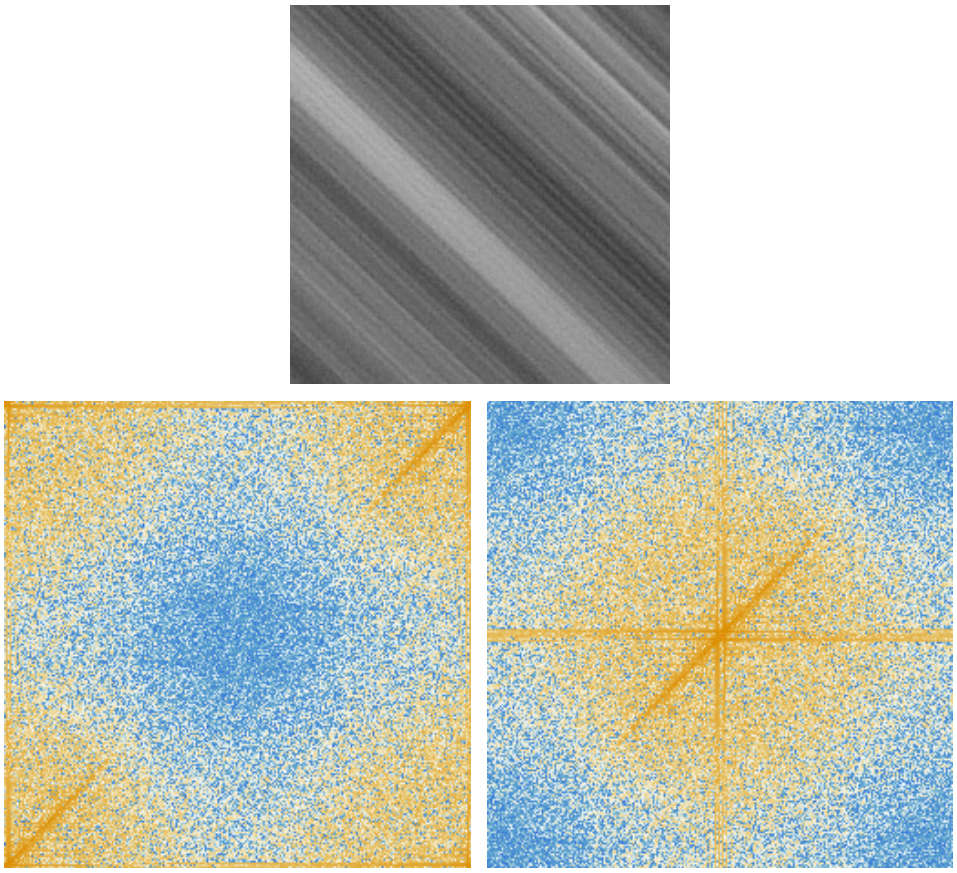
Amplitude Spectrum
Shifted



```
% unshifted image transform
image = double(imread('pot.tif'))/255.0;
FT = fft2(image);
imagesc(abs(log2(FT)))
% the log2 enhances details
axis image

% shifted image transform
image = double(imread('pot.tif'))/255.0;
fourier = fftshift(fft2(ifftshift(image)));
fftimage = log(max(real(fourier),0.0))/20.0;
imagesc(fftimage)
```

Figure 4.37: The Two-Dimensional DFT in Matlab.



```
(*2D DFT of an image*)
(*i=Import["/Users/davidsalomon/Desktop/mi.tif"];*)
img = ImageTake[i, {3101, 3356}, {2270, 2525}]
(*Take a 256x256 part of the image and
Convert it from image to numeric matrix*)
ml = ImageData[img];
fc = Fourier[ml, FourierParameters -> {1, -1}];
rel = Re[fc];
MatrixPlot[Log2[1 rel]] (*Plot the base2 log*)
dim =  Dimensions[rel] (*Start shifting*)
If[OddQ[dim[[2]]], c = Ceiling[dim[[2]]/2], c = dim[[2]]/2];
If[OddQ[dim[[1]]], r = Ceiling[dim[[1]]/2], r = dim[[1]]/2];
rel = RotateLeft[rel, {0, c}];
rel = RotateLeft[rel, r];
MatrixPlot[Log2[1 rel]] (*Plot shifted version*)
```

Figure 4.38: A Two-Dimensional DFT in Mathematica.

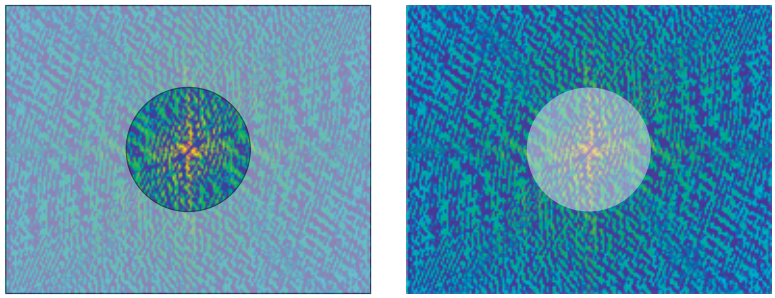
```
(*Low-pass filter to blur an image
Matrix 'fo' entries are 0 except a
(2par x 2par) square in its middle*)
lpFiltr[par_, c_, r_] := Module[{i, j, c2, r2, fo},
  (*Construct matrix fo with zeros*)
  c2 = Floor[c/2]; r2 = Floor[r/2];
  fo = Table[0, {i, 1, c}, {j, 1, r}];
  For[i = c2 - par, i <= c2, i++, (*Spread (2par x 2par) ones*)
   For[j = r2 - par, j <= r2, j++,(* around the center of fo*)
    fo[[i, j]] = fo[[i, r + 1 - j]] =
    fo[[c + 1 - i, j]] = fo[[c + 1 - i, r + 1 - j]] = 1;]];
  fm = fo;]

matShift[mm_] := Module[{dim, c, r},
  dim = Dimensions[mm];
  If[OddQ[dim[[2]]], c = Ceiling[dim[[2]]/2], c = dim[[2]]/2];
  If[OddQ[dim[[1]]], r = Ceiling[dim[[1]]/2], r = dim[[1]]/2];
  m1 = RotateLeft[mm, {0, c}];
  ml = RotateLeft[m1, r];]

matShiftBack[mm_] := Module[{dim, c, r},
  (*Shift mm back*)
  dim = Dimensions[mm];
  If[OddQ[dim[[2]]], c = Ceiling[dim[[2]]/2], c = dim[[2]]/2];
  If[OddQ[dim[[1]]], r = Ceiling[dim[[1]]/2], r = dim[[1]]/2];
  m3 = RotateLeft[mm, -r];
  ml = RotateLeft[m3, {0, -c}];]

(*Test*)
img = Import["/Users/davidsalomon/Desktop/mayaBW.tif"]
ml = ImageData[img];
ml = Fourier[ml, FourierParameters -> {1, -1}];
matShift[ml];
dim = Dimensions[ml];
c = dim[[1]]; r = dim[[2]];
lpFiltr[20., c, r]
MatrixPlot[fm];  (*all zeros except ones in center*)
ml = ml fm; (*ElementWise multiplication*)
matShiftBack[ml];
ml = InverseFourier[ml, FourierParameters -> {1, 1}];
ImageReflect[ImageReflect[Image[Abs[ml]]], Left]
```

Figure 4.39: Blurring An Image with a Low-Pass Filter.



(a) (b)

Figure 4.40: Filtering the DFT Coefficients.

```
img = Import["/Users/davidsalomon/Desktop/mayaBW.tif"]
ml = ImageData[img];
ml = Fourier[ml, FourierParameters -> {1, -1}];
dim = Dimensions[ml];
r = dim[[1]]; c = dim[[2]];
par = 7;
For[i = 1, i <= par, i++,
 For[j = 1, j <= par, j++,
  ml[[i, j]] = ml[[r - i + 1, j]] =
  ml[[i, c - j + 1]] = ml[[r - i + 1, c - j + 1]] = 0;
  ]]
MatrixPlot[ml]
ml = InverseFourier[ml, FourierParameters -> {1, 1}];
MatrixPlot[Reverse[Abs[ml]], ColorFunction -> "GrayTones"]
```

Figure 4.41: The Results of a High-Pass Filter.

by erasing or deleting those regions and inverse transforming the two-dimensional fre-

quency coefficients matrix. A good reference, with Mathematica code and examples, is `https://tinyurl.com/423za65j`

4.7 2D Graphics Transform Example

We are now ready to look at a complete example of a 2D DFT. Figure 4.43 (compare with Figure 2) is a listing of the Mathematica code, and the three parts of Figure 4.44 display the results.

The code of Figure 4.43 imports a small 65×65-pixel grayscale image, transforms it, shifts it, and then performs a double loop where it scans the resulting 65×65 matrix of transform coefficients. For each coefficient it computes an amplitude and phase and generates a 65×65 matrix of 2D sine waves. These matrices are accumulated and they gradually form the final image. The three parts of Figure 4.44 show the original image, seven pairs of intermediate results generated every 10 iterations, and the final, reconstructed image. Each of the seven pairs consists of an intermediate image and the current 2D sine wave. Here are detailed notes referring to the line numbers of Figure 4.43. Notice that 65 is an odd number. It was chosen here as the dimension of the image in order to simplify the code. With an odd-size image, there is a center row, center column, and center pixel, whereas with an even dimension, the most important pixel is shifted from the top-left corner to the row and column immediately following the image center. For even dimensions, the code listed here may have to be modified, but the intention here was to end up with simple, readable code.

Lines 4–10 list the shift procedure `matShift`. This has been discussed in an earlier section. The 2D transform is done on line 14. The final, reconstructed image will be generated in matrix `ts`, which is initialized to all zeros on line 18. Procedure `sg`, defined on line 19, will compute the 2D sine waves. The computations are done in a double loop (lines 21–33), where the `r` and `c` counters run from 1 to 65 each. However, the 2D sine waves, which are computed on line 25, employ row and column variables `row` and `col` (line 19) which run from -32 to $+32$ each. This is because the shifting has changed the original numbers of the rows and columns. The dominant value, which was at the top-left (row 0) is now at the center of the coefficient matrix, and the computations have to assign it its original row number. The amplitude and phase for each 2D sine wave are computed on lines 23 and 24, respectively. Each iteration calculates an intermediate image which is added to `ts` on line 25. Lines 29 through 32 print intermediate results every 10 iterations. The `ArrayPlot` on line 30 prints the current intermediate image, while the `DensityPlot` on line 31 prints the current 2D sine wave. Once the double loop is over, matrix `ts` is printed (with only two digits after the decimal point) and is finally displayed with a final `ArrayPlot` on line 35.

Seeing and "seeing."

There is seeing and there is seeing, and the two are different. This short discussion concentrates on the difference between the two types of seeing and explains why the Mathematica code of Figure 4.43 had to compute both an `Array` (a numerical matrix to be added to `ts` on line 25) and a `DensityPlot` (an image to be displayed on lines 31–32).

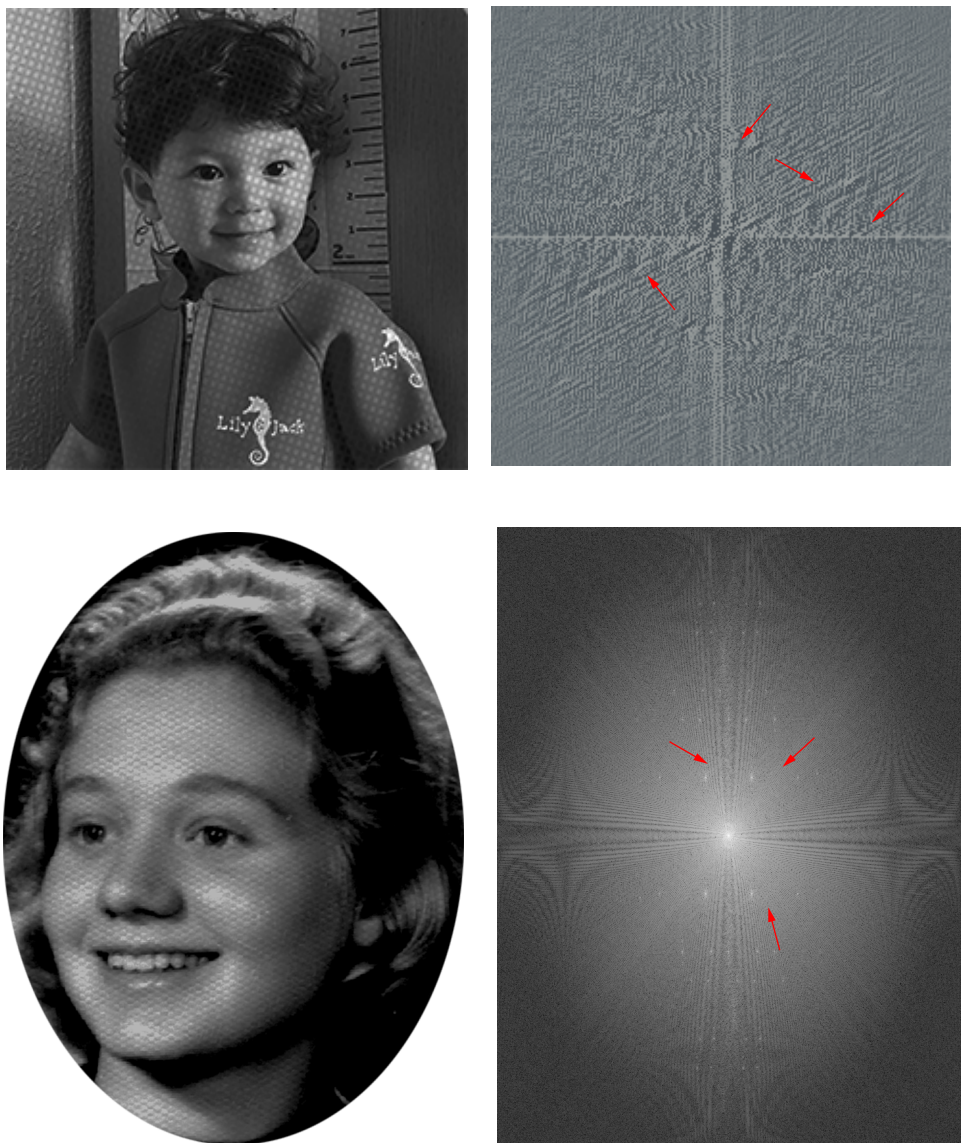


Figure 4.42: Image Texture and Noise as Seen in the DFT Matrix.

```
1 (*Transform a 65x65 image and then reconstruct it by computing the
2  amplitude and phase from the transformed and shifted matrix ml*)
3
4 Clear["Global`*"];
5 matShift[mm_] := Module[{dim, c, r},
6   dim = Dimensions[mm];
7   If[OddQ[dim[[2]]], c = Ceiling[dim[[2]]/2], c = dim[[2]]/2];
8   If[OddQ[dim[[1]]], r = Ceiling[dim[[1]]/2], r = dim[[1]]/2];
9   m1 = RotateLeft[mm, {0, c}];
10  ml = RotateLeft[m1, r];]
11
12 img = Import["/Users/davidsalomon/Desktop/PemaIn.tif"]
13 mm = ImageData[img];
14 ft = Fourier[mm, FourierParameters -> {1, -1}];
15 matShift[ft];
16 Dimensions[ml]
17 dim = 65;
18 ts = Table[Table[0, {dim}], {dim}];
19 row = -Floor[dim/2]; col = -Floor[dim/2];
20 sg[x_, y_] := am Sin[ row x + col y + phs];
21 Do[
22  Do[a = Re[ml[[r, c]]]; b = Im[ml[[r, c]]];
23   am = Sqrt[a^2 + b^2];
24   phs = Pi/2 - ArcTan[a, -b];
25   ts = ts + Array[sg, {dim, dim}, {{0, 2 Pi}, {0, 2 Pi}}];
26   row++,
27    {r, 1, dim}];
28  row = -Floor[dim/2]; col++;
29  If[Mod[col, 10] == 0,
30   Print[col, " , ", ArrayPlot[ts, ColorFunction -> "GrayTones"], " , ",
31    DensityPlot[sg[x, y], {x, 0, 2 Pi}, {y, 0, 2 Pi},
32     ColorFunction -> "GrayTones"]]],
33  {c, 1, dim}]
34 NumberForm[ts, {5, 2}];
35 ArrayPlot[ts, ColorFunction -> "GrayTones"]
```

Figure 4.43: A Complete 2D DFT Transform: Code.

The principle of reconstructing an image from its 2D DFT is to scan the Fourier coefficient matrix and employ each of its entries (a complex number) to compute an amplitude and a phase, which are then used to generate a 2D sine wave. As these sine waves are added, the reconstructed image takes shape and approaches the original image. However, in practical coding, using Mathematica, there is a difference between an image and the matrix of its pixels, which is why the code of Figure 4.43 had to generate an array `ts` and accumulate in it, on line 25, not images of the 2D sine waves, but arrays containing the individual pixels of those waves.

Imagine that you print images of the 2D sine waves computed by the double loop of Figure 4.43 on transparent sheets. Each sheet is a clear, sharp image of such a sine wave, with clear areas, black areas, and gray areas of many graycales. You then try to view the final image simply by stacking those sheets on top of one another. You may be surprised to find out that the result is a dark picture. Each sheet has clear parts and dark parts, so stacking sheets tends to cover more and more clear areas with dark areas.

Original 65x65 Image

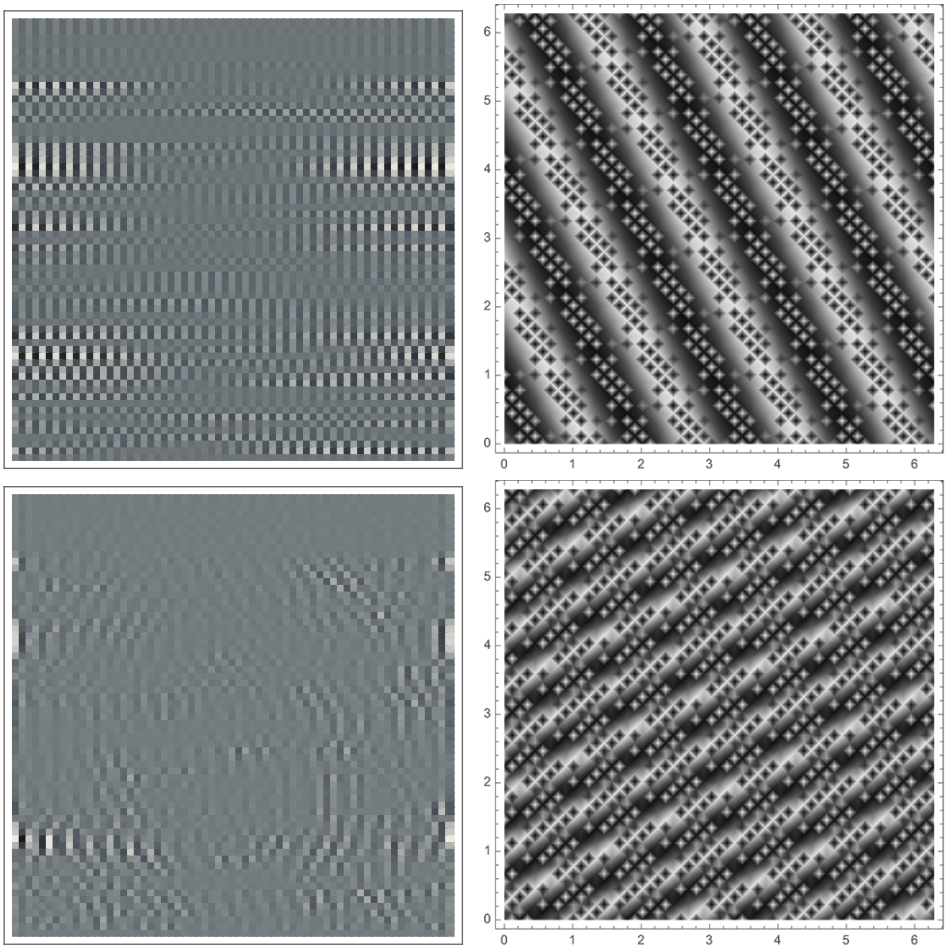


Figure 4.44: A Complete 2D DFT Transform: Part 1 of 3.

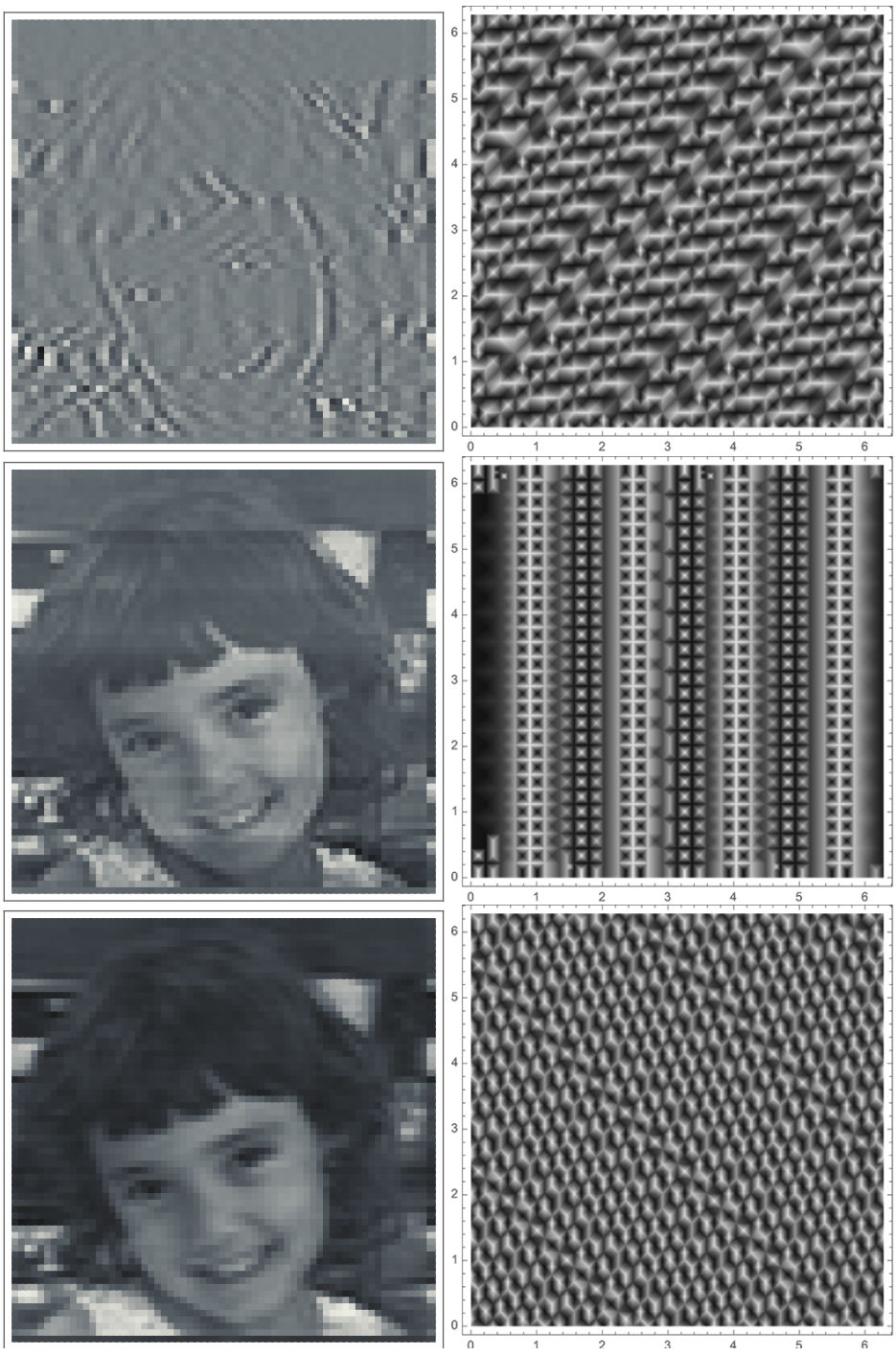
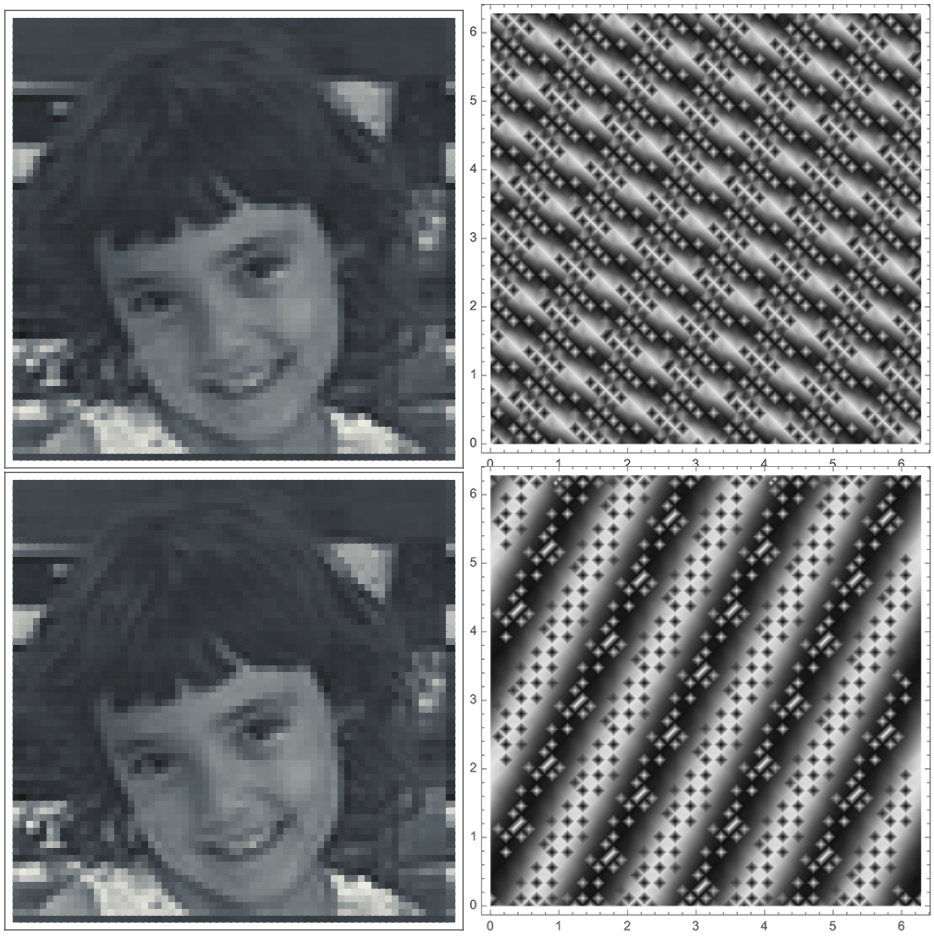


Figure 4.44: A Complete 2D DFT Transform: Part 2 of 3.



Final Reconstructed Image

Figure 4.44: A Complete 2D DFT Transform: Part 3 of 3.

Given enough sheets, no clear areas remain.

On the other hand, adding intermediate arrays (matrices) to variable `ts` on line 25 is different, because each array contains signed numbers, not just zeros and ones. The 65×65 numbers that constitute such an array can be negative, zero, or positive. Thus, adding the next array tends to increase some numbers, but also to decrease others. This is why the final array, printed on line 34 and displayed on line 35, consists of small, large, negative, and positive numbers, and is not fully black.

This discussion also explains why adding several intermediate arrays of numbers, each of them regular, may result in a final array which is irregular. Here is a simple example:

```
0 5 0 0 0        0 -2 0  0 0        0  3 0  0 0
0 0 5 0 0        0 -2 0 -4 0        0 -2 5 -4 0
3 0 0 5 0   +    0 -2 0 -4 0   =    3 -2 0  1 0
0 3 0 0 5        0 -2 0 -4 0        0  1 0 -4 5
0 0 3 0 0        0 -2 0  0 0        0 -2 3  0 0
```

References for this section are [fingerprints 20], [Session 3 15], and [DFT demo 18].

4.8 The Fast Fourier Transform

Called "the most important numerical algorithm of our lifetime," the fast Fourier transform is based on ideas due to Gauss in the early 1800's (even preceding Fourier's own ideas). The first generic FFT algorithm was published in 1965 by James Cooley and John Tukey and it reduced the complexity of the DFT from $O(N^2)$ to the much smaller $O(N \log_2 N)$. Even though surprisingly simple, this algorithm is not obvious. Today, many FFT algorithms are known which are based on different principles, and the discussion here is only an outline of the basic concepts, which are based on the periodicity of the sine and cosine functions.

Gauss and the FFT
The eminent mathematician (and all-round scientist) Carl Friedrich Gauss was active in many fields, among them astronomy. When the first asteroid, Ceres, was discovered in 1801, Gauss decided to compute its orbit, and to speed up his work he developed two ingenious computational techniques. One (least squares curve fitting) has immediately become popular, but the other (the recursive FFT, even before Fourier himself) was forgotten and had to be rediscovered in 1965.

Readers of this book already know that the inverse DFT is very similar to the DFT, which is why the principles behind the various FFT algorithms also apply to the inverse FFT.

The main idea behind the various versions of the FFT is to identify those parts of the computations that occur several times, save them, and reuse them. Such parts are the values of various exponentials, as well as sums and differences of the form $x_i \pm x_j e^{-2\pi i \cdots}$ and $e^{-2\pi i \cdots}(x_i \pm x_j)$.

the basic DFT is defined by Equation (4.2), duplicated here

$$X_k = \sum_{n=0}^{N-1} x_n e^{-2\pi i k n/N} = \sum_{n=0}^{N-1} x_n \, \omega_N^{kn}, \tag{4.2}$$
$$\text{for } k = 0, 1, 2, \ \ldots \ , N-1, \text{ and } \omega_N \stackrel{\text{def}}{=} e^{-2\pi i/N}.$$

The DFT starts with an array x_n of N data items and computes an array X_k of N frequency coefficients or bins. Computing any coefficient X_k requires N iterations of the sum of Equation (4.2), so computing all N coefficients requires a total of N^2 iterations. For $N = 10^3$, the FFT represents a saving factor of $N^2/(N \log_2 N) \approx 10^6/(10^3 \times 10) = 100$, whereas for $N = 10^6$ the ratio of the number of operations is approximately $10^{12}/(10^6 \cdot 20) = 50000$, or 1 sec instead of 13.9 hours, very significant!

◇ **Exercise 4.7:** Derive an estimate of the time complexity of the straight DFT, considering that it requires arithmetic operations on complex numbers.

The simplest version of the FFT requires N to be a power of 2, and its computations can proceed in one of two ways, which are referred to as Decimation in Time (DIT) and Decimation in Frequency (DIF).

Does time have a frequency?

We intuitively feel that time is a one-dimensional phenomenon. It moves constantly, but it seems uniform and smooth; there are no changes and there is no frequency.

However, since time is important to us, both in science and technology (GPS is one of many examples) and in everyday life, we are concerned with measuring time, which is done by cutting it into equal-size slices, a process which introduces frequencies.

The movement of a pendulum is periodic, but is not regular enough for precise time measurement. The same is true for the regular movements of celestial bodies, which is why modern science uses the internal frequency of atoms to achieve unprecedented accuracy in measuring time.

The conclusion is that time does not have an internal frequency, but its measurement introduces a frequency, which is why we often think of time in discrete units.

4.8.1 Decimation in Time

The DIT method is described here by means of a detailed example for $N = 8$.

> Decimation
>
> To kill, destroy, or remove a large percentage or part of. This is derived from the Latin meaning "removal of a tenth."
>
> —Dictionary definition of decimation.

The FFT problem is to compute the DFT fast. We therefore start by writing the DFT sum, Equation (4.2), explicitly, for $N = 8$, and then partitioning it into sums of

even and odd indices

$$\begin{aligned}X_k &= x_0 + x_1 e^{-i\frac{2\pi}{8}k} + x_2 e^{-i\frac{2\pi}{8}2k} + x_3 e^{-i\frac{2\pi}{8}3k} \\ &\quad + x_4 e^{-i\frac{2\pi}{8}4k} + x_5 e^{-i\frac{2\pi}{8}5k} + x_6 e^{-i\frac{2\pi}{8}6k} + x_7 e^{-i\frac{2\pi}{8}7k} \\ &= \left[x_0 + x_2 e^{-i\frac{2\pi}{8}2k} + x_4 e^{-i\frac{2\pi}{8}4k} + x_6 e^{-i\frac{2\pi}{8}6k}\right] \\ &\quad e^{-i\frac{2\pi}{8}k}\left[x_1 + x_3 e^{-i\frac{2\pi}{8}2k} + x_5 e^{-i\frac{2\pi}{8}4k} + x_7 e^{-i\frac{2\pi}{8}6k}\right].\end{aligned}$$

The sums in the square brackets are rearranged and are split again

$$\begin{aligned}X_k &= \left[\left(x_0 + x_4 e^{-i\frac{2\pi}{8}4k}\right) + e^{-i\frac{2\pi}{8}2k}\left(x_2 + x_6 e^{-i\frac{2\pi}{8}4k}\right)\right] \\ &\quad + e^{-i\frac{2\pi}{8}k}\left[\left(x_1 + x_5 e^{-i\frac{2\pi}{8}4k}\right) + e^{-i\frac{2\pi}{8}2k}\left(x_3 + x_7 e^{-i\frac{2\pi}{8}4k}\right)\right] \\ &= \left[\left(x_0 + x_4 e^{-i\pi k}\right) + e^{-i\frac{\pi}{2}k}\left(x_2 + x_6 e^{-i\pi k}\right)\right] \\ &\quad + e^{-i\frac{\pi}{4}k}\left[\left(x_1 + x_5 e^{-i\pi k}\right) + e^{-i\frac{\pi}{2}k}\left(x_3 + x_7 e^{-i\pi k}\right)\right]. \qquad (4.12)\end{aligned}$$

The original, 8-item DFT, has been rewritten as three levels of summation. The top (or outer) level consists of two parts, one with data items x_n with even indices n and the other with odd indices. Each lower (or inner) level similarly consists of two parts that we also refer to as even and odd. In general, the number of levels is $\log_2 N$. In the expression above, the innermost level is surrounded by parentheses, the middle level has square brackets, and the top level has no brackets. Each level has its own type of exponential, and the first step toward speeding up the calculations is to employ the periodicity of our exponentials. We use the identities $e^{i(\phi+2\pi)} = e^{i\phi}$ and $e^{i(\phi+\pi)} = -e^{i\phi}$, which result in the following sets of values for each summation level. The set for the inner level is

$$e^{-i\pi k} = \begin{cases} 1 & k = 0, 2, 4, 6, \\ -1 & k = 1, 3, 5, 7. \end{cases}$$

For the middle level the set of exponentials we need is

$$e^{-i\frac{\pi}{2}k} = \begin{cases} 1 & k = 0, 4, \\ -i & k = 1, 5, \\ -1 & k = 2, 6, \\ i & k = 3, 7. \end{cases}$$

The eight exponentials for the top level use the values of sine and cosine of $\pi/4 = 45°$, which are $\pm\frac{1}{\sqrt{2}} \approx \pm 0.7071$. In order to shorten our expressions we will simply use

the rough approximation ± 0.7 for those numbers, which are

$$e^{-i\frac{\pi}{4}k} = \begin{cases} 1 & k=0, \\ 0.7-0.7i & k=1, \\ -i & k=2, \\ -0.7-7i & k=3, \\ -1 & k=4, \\ -0.7+0.7i & k=5, \\ i & k=6, \\ 0.7+7i & k=7. \end{cases}$$

However, in the computations of $X[k]$ that follow, as well as in the final diagrams, the better value, 0.7071, is used, even when it is written as 0.7.

Notice that our exponentials are mostly complex numbers and (most important) each of the exponential tables above can be divided into a top and bottom halves, which differ by their signs. Also, the total number of exponentials needed for $N=8$ is $N+N/2+N/4=8+4+2$, and for an arbitrary N it is

$$N+\frac{N}{2}+\frac{N}{4}+\cdots = N\left[1+\frac{1}{2}+\frac{1}{4}+\cdots\right],$$

A number that approaches $2N$ for large values of N. This means that all the exponentials needed by the algorithm can be precomputed once before the main body of the algorithm starts and saved for the duration, since the maximum storage can never exceed $2N$ locations, each large enough for a complex number.

The next step toward speeding up the calculations is to identify those parts of Equation (4.12) that are used more than once. They would have to be saved and reused, thereby saving much time. We start with the first term of the inner level of Equation (4.12), namely $(x_0+x_4e^{-i\pi k})$. The exponent used here has been precomputed and equals either 1 (for $k=0,2,4,6$) or -1 (for $k=1,3,5,7$). The algorithm should therefore add (x_0+x_4) and use this sum to compute output X_k for the even values of k. Then subtract (x_0-x_4) and use this difference to compute output X_k for the odd values of k. This simple process is repeated for the three other sums of the inner level. Each is computed by a single addition and a single subtraction, resulting in six more numbers, for a total of eight numbers as follows

$$\begin{aligned} &a=(x_0+x_4),\ b=(x_0-x_4),\ c=(x_2+x_6),\ d=(x_2-x_6),\\ &e=(x_1+x_5),\ f=(x_1-x_5),\ g=(x_3+x_7),\ h=(x_3-x_7). \end{aligned}$$

These numbers can be stored in the locations so far occupied by the original data x_n, since this data will no longer be needed. These eight numbers are also sent to the next, middle level to be processed further.

The middle level receives the eight numbers a through h and employs them to compute its own eight results. Depending on k, these results are sums and differences of pairs of a through h, some of which are multiplied by $e^{-i\frac{\pi}{2}k}$. Before being sent to the top level, these eight results can be stored in the same locations that have recently occupied a through h.

The top level receives the eight results and computes each final X_k value by multiplying, when needed, by an exponent of the form $e^{-i\frac{\pi}{4}k}$, and performing one addition or subtraction. Here are the details of computing the first four outputs $X[0]$ through $X[3]$, assuming the 8-item original data 1, -2 , 5, 7, -3, 8, 4, and 6.

$$\begin{aligned}
X[0] &= \left[\left(x_0 + x_4 e^{-i\pi 0}\right) + e^{-i\frac{\pi}{2}0}\left(x_2 + x_6 e^{-i\pi 0}\right)\right] \\
&\quad + e^{-i\frac{\pi}{4}0}\left[\left(x_1 + x_5 e^{-i\pi 0}\right) + e^{-i\frac{\pi}{2}0}\left(x_3 + x_7 e^{-i\pi 0}\right)\right] \\
&= \left[(x_0 + x_4) + e^{-i\frac{\pi}{2}0}(x_2 + x_6)\right] \\
&\quad + e^{-i\frac{\pi}{4}0}\left[(x_1 + x_5) + e^{-i\frac{\pi}{2}0}(x_3 + x_7)\right] \\
&= [(x_0 + x_4) + (x_2 + x_6)] + [(x_1 + x_5) + (x_3 + x_7)] \\
&= [(1 - 3) + (5 + 4)] + [(-2 + 8) + (7 + 6)] = 26.
\end{aligned}$$

$$\begin{aligned}
X[1] &= \left[\left(x_0 + x_4 e^{-i\pi 1}\right) + e^{-i\frac{\pi}{2}1}\left(x_2 + x_6 e^{-i\pi 1}\right)\right] \\
&\quad + e^{-i\frac{\pi}{4}1}\left[\left(x_1 + x_5 e^{-i\pi 1}\right) + e^{-i\frac{\pi}{2}1}\left(x_3 + x_7 e^{-i\pi 1}\right)\right] \\
&= \left[(x_0 + x_4(-1)) + e^{-i\frac{\pi}{2}1}(x_2 + x_6(-1))\right] \\
&\quad + e^{-i\frac{\pi}{4}1}\left[(x_1 + x_5(-1)) + e^{-i\frac{\pi}{2}1}(x_3 + x_7(-1))\right] \\
&= [(x_0 - x_4) + (-i)(x_2 - x_6)] \\
&\quad + (0.7 - 0.7i)[(x_1 - x_5) + (-i)(x_3 - x_7)] \\
&= [(1 + 3) + (-i)(5 - 4)] \\
&\quad + (0.7 - 0.7i)[(-2 - 8) + (-i)(7 - 6)] = -3.7781 + 5.3639i.
\end{aligned}$$

$$\begin{aligned}
X[2] &= \left[\left(x_0 + x_4 e^{-i\pi 2}\right) + e^{-i\frac{\pi}{2}2}\left(x_2 + x_6 e^{-i\pi 2}\right)\right] \\
&\quad + e^{-i\frac{\pi}{4}2}\left[\left(x_1 + x_5 e^{-i\pi 2}\right) + e^{-i\frac{\pi}{2}2}\left(x_3 + x_7 e^{-i\pi 2}\right)\right] \\
&= [(x_0 + x_4) + (-1)(x_2 + x_6)] \\
&\quad + (-i)[(x_1 + x_5) + (-1)(x_3 + x_7)] \\
&= [(1 - 3) + (-1)(5 + 4)] + (-i)[(-2 + 8) + (-1)(7 + 6)] = -11 + 7i.
\end{aligned}$$

$$\begin{aligned}
X[3] &= \left[\left(x_0 + x_4 e^{-i\pi 3}\right) + e^{-i\frac{\pi}{2}3}\left(x_2 + x_6 e^{-i\pi 3}\right)\right] \\
&\quad + e^{-i\frac{\pi}{4}3}\left[\left(x_1 + x_5 e^{-i\pi 3}\right) + e^{-i\frac{\pi}{2}3}\left(x_3 + x_7 e^{-i\pi 3}\right)\right] \\
&= [(1 - (-3)) + i(5 - 4)] \\
&\quad + (-0.7 - 0.7i)[(-2 - 8) + i(7 - 6)] = 11.7781 + 7.3639i.
\end{aligned}$$

◇ **Exercise 4.8:** Derive the remaining outputs $X[4]$ through $X[7]$.

A careful examination of the eight outputs shows that six of them form pairs of the form $a + bi$ and $a - bi$. These correspond to positive and negative frequenies and are explained in the text found around Equation (4.5).

It is now easy to estimate the number of computations required for each output. In our example there are three levels, and in general the number of levels is $\log_2 N$. In our

example, each level requires $4+2+1$ multiplications and the same number of additions or subtractions. In general this number is

$$\frac{N}{2}+\frac{N}{4}+\frac{N}{8}+\cdots+4+2+1=N\left[\frac{1}{2}+\frac{1}{4}+\frac{1}{8}+\cdots+\frac{1}{N/2}+\frac{1}{N}\right]<N.$$

The total time complexity is therefore $N\log_2 N$.

In spite of the complexity of this algorithm, it is possible to summarize it in a single diagram. We start with Figure 4.45 which shows the complete, three-level path from eight inputs to one output. This is shown for outputs $X[3]$ (in red) and $X[6]$ (in green). Readers who follow these paths may find it possible, although perhaps intimidating, to understand the entire process, which is summarized in the butterfly diagram of Figure 4.46.

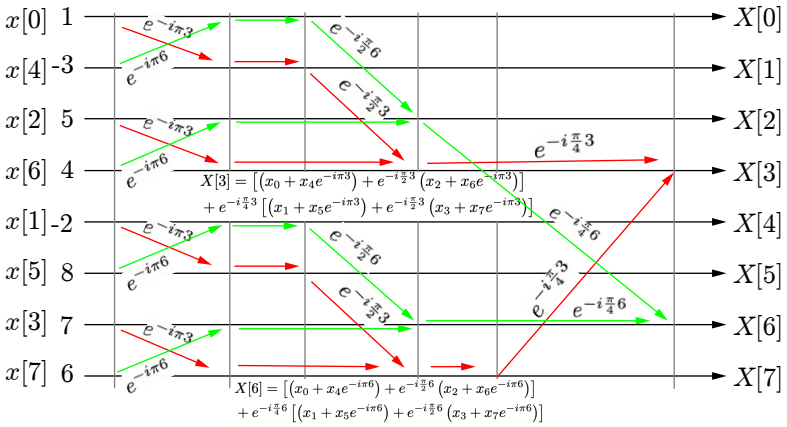


Figure 4.45: FFT by Decimation in Time. Two paths from Inputs to Outputs.

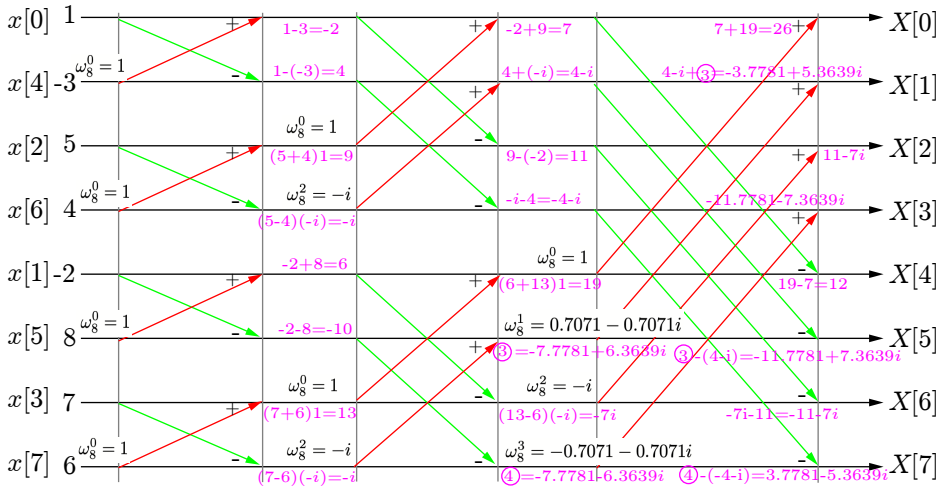
The short notation $\omega_N \stackrel{\text{def}}{=} e^{-2\pi i/N}$ is used in Figure 4.46 instead of $e^{-2\pi ikn/N}$. The reader will also encounter the notation ω_N^{kn}, as well as $\omega_N^{k+N/2}=-\omega_N^k$.

Note the numbers 3 and 4 in circles in Figure 4.46. Their values were too long to include in the figure, so they are listed here

$$\text{Circle 3}=(-10+(-i))(0.7-0.7i)=-7.7781+6.3639i.$$
$$\text{Circle 4}=(-i-(-10))(-0.7071-0.7071i)=-7.7781-6.3639i.$$

The bottom (innermost) level deals with the pairs of inputs with indices $(0,4)$, $(2,6)$, $(1,5)$, and $(3,7)$, which is why the original input to the algorithm must be arranged in this order before computations can start. Fortunately, this is easy and it involves only bit reversal, an operation that changes, for example, $110_2=6$ to $011_2=3$. Thus, the sequence 0, 1, 2, 3, 4, 5, 6, 7 becomes 0, 4, 2, 6, 1, 5, 3, and 7.

Once this example of eight inputs and eight outputs is clear, it is easy to see how the general case of an arbitrary N is implemented in an algorithm. The main operations




```
data={1,-2,5,7,-3,8,4,6};
Fourier[data,FourierParameters->{1,1}]
```

Figure 4.46: FFT by Decimation in Time. Complete $N = 8$ Example.

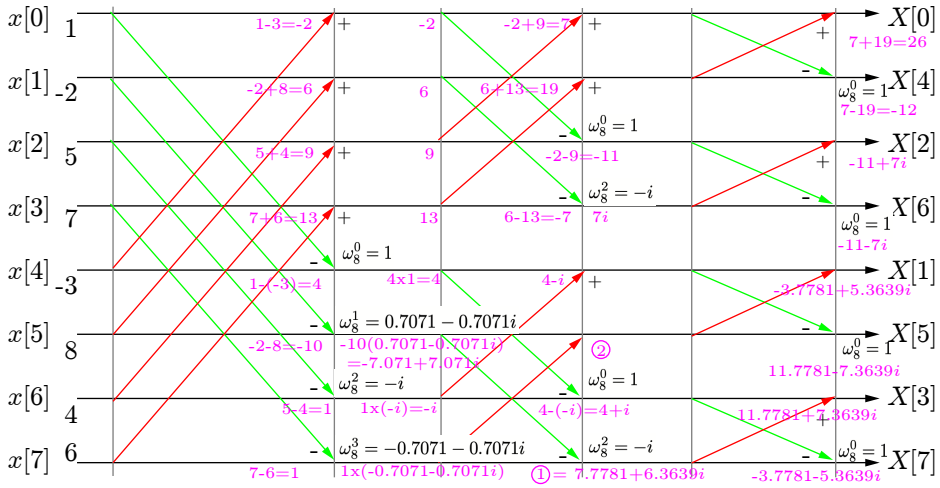
are partitioning (splitting or decimating) and recursion. The code starts at the bottom (innermost) level with the sequence of N original inputs. It computes N results by performing one addition and one subtraction for each pair of data items. These N results are sent to the next higher level, where similar operations are performed (with some multiplications), resulting in a new set of N numbers that are sent to the next level.

In summary, the DIT version of the basic FFT starts by arranging the input sequence according to the indices (even or odd) of the N data items. It then proceeds by considering consecutive pairs of items (there are $N/2$ pairs) and computing two results from each pair. Each such result comes from two original data items. The N results are sent to the next level, where a pair of results (each coming from two data items) is used to produce new results, each of which comes from four original data items. This continues up the levels. Each level produces N new results, each of which comes from more original data items. The last level produces N final results, each of which comes from all N original data items.

In practice, such a loop is implemented with recursion. The pseudocode on Page 111 shows how this is done. Procedure `fft` invokes itself twice, first with the data items with even indices, and then with those items with odd indices. When both recursive calls return, the procedure performs one butterfly (multiply, add, subtract) and assigns its two results to two output locations. This is short and elegant, but confusing when seen for the first time.

The other version of the basic FFT, Decimation in Frequency (DIF), starts by partitioning the original sequence of data items into two sequences containing the first half and second half of the original items, respectively. The first level selects an item

from each sequence, computes two results, and repeats for $N/2$ pairs of items. The N new results are sent to the next level which also computes N results, each coming from four original items, and so on. This process is summarized, for $N = 8$, in Figure 4.47. Notice how similar this figure is to Figure 4.46. The two are essentially mirror images of each other, even in the positions of the twiddle factors ω_N^k. (The dictionary definition of twiddle is "to twist, move, or fiddle with (something), typically in a purposeless or nervous way.")




```
data={1,-2,5,7,-3,8,4,6};
Fourier[data,FourierParameters->{1,-1}]
```

Figure 4.47: FFT by Decimation in Frequency. $N = 8$ Example.

The main differences between DIT and DFT are the following:

- DIT partitions the input sequence x_n according to the parity of the indices n, while DIF partitions the same sequence into its first half and second half.
- The input sequence of DIT must first be rearranged in bit-reversed order and the output sequence comes out in natural order. DIF employs the inverse scheme where the input is in natural order and the output is generated in bit-reverse.

Computer Generated Text

The fast Fourier transform (FFT) is an efficient algorithm for computing the discrete Fourier transform (DFT) of a sequence. The DFT is a mathematical operation that transforms a sequence of n complex numbers $x[0]$, $x[1]$, ... , $x[n-1]$ into another sequence of n complex numbers $X[0]$, $X[1]$, ... , $X[n-1]$, where each $X[k]$ is given by:

$$X[k] = \sum_{i=0}^{n-1} x[i]\, e^{-2\pi ik/n}.$$

The FFT is a divide-and-conquer algorithm that recursively breaks down the DFT into smaller DFTs, using the fact that the DFT can be computed by summing two DFTs of half the size. Here is some pseudocode for the FFT algorithm:

```
def fft(x):
  n = len(x)
  if n == 1:
    return x
  even = [x[i] for i in range(0, n, 2)]
  odd = [x[i] for i in range(1, n, 2)]
  X_even = fft(even)
  X_odd = fft(odd)
  X = [0] * n
  for k in range(n//2):
    t = cmath.exp(-2*cmath.pi*k/n) * X_odd[k]
    X[k] = X_even[k] + t
    X[k + n//2] = X_even[k] - t
  return X
```

The FFT algorithm has a time complexity of $O(n \log_2 n)$, which is much faster than the naive algorithm for computing the DFT, which has a time complexity of $O(n^2)$. The FFT is widely used in signal processing and other applications where the DFT is needed.

Generated by ChatGPT, 22 Dec 2022.

4.8.2 A Different Look at the FFT

We now change our viewpoint and use references [reducible 20a,b] to look at the FFT from a different and (to some) more familiar direction, that of polynomial multiplication. The assumption is that a reader who wants to understand Fourier methods in general and the FFT in particular would likely be familiar with polynomials and arithmetic operations on them. It turns out that the FFT can be understood by looking at polynomial multiplication, and especially at the problem of converting a polynomial from its traditional coefficient representation to the completely different value representation.

The traditional method of multiplying polynomials relies on their distributive property, which is illustrated by the identity $a(b+c) = ab + ac$. Given the two polynomials $A(x) = -x^2 + 4x - 1.5$ and $B(x) = 2x^2 + 3$, we can write

$$A(x) \cdot B(x) = (-x^2+4x-1.5)(2x^2+3) = -x^2(2x^2+3)+4x(2x^2+3)-1.5(2x^2+3) = \ldots,$$

and continue to distribute items until we have a long sum where each term is a product of the form $c_i x^i$, where c_i is a constant. It is easy to see that this method of multiplication is slow. Each term in $A(x)$ must be multiplied by every term of $B(x)$. If the two polynomials have degrees a and b, the multiplication requires ab operations (except when certain terms are missing, such as the term x in our $B(x)$). If both polynomials have degree d, the time complexity of the multiplication is denoted by $O(d^2)$, very slow.

When manipulating polynomials in a computer, the most natural way of storing them is to store the $n+1$ coefficients of a degree-n polynomial in an array according to increasing powers of the variable. Thus, the two polynomials above would be stored in the arrays $(-1.5, 4, -1)$ and $(3, 0, 2)$. This way, the coefficient of x^k is stored in location $k+1$ of the array. We can call this representation of a polynomial its coefficient representation.

We now consider the problem of speeding up polynomial multiplication. We start with the simplest, degree-1 polynomial, whose expression is $P(x) = p_0 + p_1 x$. This mathematical object can also be considered a linear function and can be graphed as a straight line with slope p_1 (the degree-1 term) and y-intercept p_0 (the degree-0 term). Those familiar with analytic geometry (the study of geometry using a coordinate system) know that such a line can also be represented in other ways, such as the following:

- Polar coordinates. Given a straight line inclined at θ degrees relative to the x-axis and passing through point $P = (h, k)$, its polar equation is $Q(x, y) = (h + r\cos\theta, k + r\sin\theta)$, where r is the distance of point Q from P, and it can vary in any desired interval.

- Parametrically. The expression $\mathbf{P}(t) = (1-t)\mathbf{A} + t\,\mathbf{B}$ is the parametric (or two-point) equation of a straight line, where $\mathbf{A}$ and $\mathbf{B}$ are two distinct points on the line. When varying t from 0 to 1, the line progresses from $\mathbf{A}$ to $\mathbf{B}$.

- The Lagrange polynomial extends the basic two-point representation of a straight line to arbitrary polynomial curves. Given $n+1$ distinct data points $\mathbf{P}_0 = (x_0, y_0)$, $\mathbf{P}_1 = (x_1, y_1), \ldots, \mathbf{P}_n = (x_n, y_n)$, we search for a function $y = f(x)$ that will pass through all of them. We first try an expression of the form $y = \sum_{i=0}^{n} y_i L_i^n(x)$. This is a weighted sum of the individual y_i coordinates where the weights L_i^n depend on the x_i coordinates. This sum will pass through the points if

$$L_i^n(x) = \begin{cases} 1, & x = x_i, \\ 0, & \text{otherwise.} \end{cases}$$

A good mathematician can easily guess that such functions are given by

$$L_i^n(x) = \frac{\Pi_{j\neq i}(x - x_j)}{\Pi_{j\neq i}(x_i - x_j)} = \frac{(x-x_0)(x-x_1)\cdots(x-x_{i-1})(x-x_{i+1})\cdots(x-x_n)}{(x_i-x_0)\cdots(x_i-x_{i-1})(x_i-x_{i+1})\cdots(x_i-x_n)}.$$

(Note that $(x - x_i)$ is missing from the numerator and $(x_i - x_i)$ is missing from the denominator.) The function $y(x) = \sum_{i=0}^{n} y_i L_i^n(x)$ is called the Lagrange polynomial, reference [Lagrange 77], and it is a polynomial of degree n, denoted by LP.

Example. Given the four points $(1, 1)$, $(2, 8)$, $(3, 27)$, and $(4, 64)$, the Lagrange polynomial that passes through them is

$$\begin{aligned} L(x) &= 1 \times \frac{(x-2)(x-3)(x-4)}{(1-2)(1-3)(1-4)} + 8 \times \frac{(x-1)(x-3)(x-4)}{(2-1)(2-3)(2-4)} \\ &\quad + 27 \times \frac{(x-1)(x-2)(x-4)}{(3-1)(3-2)(3-4)} + 64 \times \frac{(x-1)(x-2)(x-3)}{(4-1)(4-2)(4-3)} \\ &= x^3. \end{aligned}$$

Given a set of points, there are many curves that pass through the points, but the Lagrange method produces only one of them. This may be a serious disadvantage in practical computer graphics work, which is why the LP is not used much in practice.

◇ **Exercise 4.9:** Show that the LP is unique.

The existence of the Lagrange polynomial suggests another representation of polynomials, the value representation. Given a degree-d polynomial $P(x)$, its value representation is a list, any list, of n distinct points, where $n \geq d+1$, through which $P(x)$ passes, something of the form

$$\big[\big(x_0, \mathbf{P}(x_0)\big), \big(x_1, \mathbf{P}(x_1)\big), \ldots, \big(x_d, \mathbf{P}(x_d)\big)\big] .$$

Our interest in the value representation is that it simplifies the multiplication of polynomials. Imagine two univariate quadratic polynomials $A(x) = x^2 + 3x + 1$ and $B(x) = x^2 - 3x + 1$. (Univariate means there is no binary term such as xy.) The graphs of such functions are parabolas, an infinite curve where x varies in the interval $[-\infty, \infty]$. For any x value, such a curve has a corresponding y value. The product $C(x) = A(x) \cdot B(x)$ is a degree-4 polynomial, so its value representation requires five points. We cleverly choose the five x values -2, -1, 0, 1, and 2, and compute their corresponding y values for $A(x)$ and $B(x)$. The results are

$$\big((-2,-1),(-1,-1),(0,1),(1,5),(2,11)\big) \text{ and } \big((-2,12),(-1,6),(0,2),(1,0),(2,0)\big),$$

respectively.

We now multiply the polynomials by first computing five new points. The x coordinates of these are the same as of the points above, going from -2 to 2. Each y coordinate is the product of the y coordinates of the corresponding pair of the points above. Thus, the y coordinates of the first points of the two lists above are multiplied to produce $(-1) \cdot 12 = -12$. The resulting five points are

$$\big((-2,-12),(-1,-6),(0,2),(1,0),(2,0)\big),$$

and the important fact is that this process required only five multiplications. So far, the time complexity has been $O(d)$, instead of the traditional polynomial multiplication, based on the coefficient representation, which requires $O(d^2)$ individual multiplications.

We still want to derive the actual expression of the product polynomial $C(x)$, and for now, we do this with the Lagrange polynomial. In our example, the result is the expression $C(x) = x^4 - 6x^2 + 3x + 2$. The Mathematica code for this process is listed here.

```
pa[x_]:=x^2+3x+1;
qa=Table[{x,pa[x]}, {x,-2,2}]
pb[x_]:=x^2-3x+2;
qb=Table[{x,pb[x]}, {x,-2,2}]
pq=Table[{x,pa[x]pb[x]},{x,-2,2}]
Simplify[InterpolatingPolynomial[pq,x]]
```

We finally get to the FFT. We will show how to use the FFT instead of the Lagrange polynomial. We start with two degree-d polynomials A and B, choose $2d+1$ points on each, multiply them as shown earlier to obtain a set of $2d+1$ points that are the value representation of the polynomial product $C = A \cdot B$. Finally, we employ the FFT to derive the expression for polynomial C from the points that constitute the value representation.

A clever way to choose $2d+1$ points is to take advantage of symmetry. If our polynomial is $P(x) = x^2$, then it is an even function which satisfies $P(-x) = P(x)$ and we can choose points with x coordinates ± 1, ± 2, and so on (a set of $\pm$ paired points). For each pair, we only need one calculation to determine both $P(x)$ and $P(-x)$. Similarly, if our polynomial is $P(x) = x^3$, then it is an odd function, which satisfies $P(-x) = -P(x)$ and we again can do one computation for each pair of paired points.

Most functions are neither odd nor even, but polynomials are simple functions. A polynomial with only even (positive or negative) exponents of the variable x is an even function, and similarly for a polynomial with only odd exponents of x. This leads to the first of the several simple, ingenious ideas (some of which had originally been proposed by Gauss) that make the FFT possible. This idea is partition. Given an arbitrary, degree-n polynomial such as $P(x) = 8x^5 - 6x^4 + x^3 + 4x^2 - 5x + 1$, simply partition it into two parts, even and odd, as follows:

$$\begin{aligned} P(x) &= (-6x^4 + 4x^2 + 1) + (8x^5 + x^3 - 5x) \\ &= (-6x^4 + 4x^2 + 1) + x(8x^4 + x^2 - 5) \\ &= P_e(x) + xP_o(x). \end{aligned} \tag{4.13}$$

Notice that the new, smaller-degree polynomials have only even exponent terms, which is why we can consider them functions of x^2 instead of x and write them as $P_e(x^2) = -6x^2 + 4x + 1$ and $P_o(x^2) = 8x^2 + x - 5$, both of degree $n/2 - 1$. (For now, let's assume that n is even and discuss this point later.) Another point to keep in mind is that now, that these two small polynomials are functions of x^2, we have to compute each at the points x_1^2, x_2^2, ..., $x_{n/2}^2$, which are all nonnegative numbers and therefore unpaired.

◇ **Exercise 4.10:** Given the degree-6 polynomial $P(x) = 9x^6+8x^5-6x^4+x^3+4x^2-5x+1$, partition it into two parts, even and odd, as shown in the text.

At this point we conclude that partitioning has transformed the original polynomial $P(x)$ to the sum $P_e(x^2) + xP_o(x^2)$, i.e., a polynomial that is almost even, because for any particular parameter x_i it satisfies

$$P(x_i) = P_e(x_i^2) + x_i P_o(x_i^2) \quad \text{and} \quad P(-x_i) = P_e(x_i^2) - x_i P_o(x_i^2), \tag{4.14}$$

a relation which saves computations in two ways as follows:

- Once we compute $P_e(x_i^2)$ and $P_o(x_i^2)$ in order to evaluate $P(x_i)$, we can use them to also evaluate $P(-x_i)$.
- Both $P_e(x_i^2)$ and $P_o(x_i^2)$ are degree 2, compared to the degree 5 of the original $P(x)$. Recall that the time complexity of computing the values of a polynomial in the

coefficient representation is $O(n^2)$, which is why we benefit from the fact that twice $O(2^2)$ is less than a single $O(5^2)$ and that this disparity gets bigger with n.

Example. We choose the degree-5 polynomial $P(x) = 8x^5 - 6x^4 + x^3 + 4x^2 - 5x + 1$, and compute its values at the six paired points ± 1, ± 2, and ± 3. Partitioning $P(x)$ as in Equation (4.13), we compute its values at the six paired points as shown in the following code. First (on line 4) directly and then (lines 5–6) according to Equation (4.14). The result, produced by the given Mathematica code, is $(3, -5, 175, -333, 1507, -2405)$ in both cases.

```
1 pp[x_] := 8 x^5 - 6 x^4 + x^3 + 4 x^2 - 5 x + 1;
2 pe[x_] := -6 x^2 + 4 x + 1;
3 po[x_] := 8 x^2 + x - 5;
4 Table[pp[x], {x, {1, -1, 2, -2, 3, -3}}]
5 Table[pe[x^2] + x po[x^2], {x, {1, 2, 3}}]
6 Table[pe[x^2] - x po[x^2], {x, {1, 2, 3}}]
```

In the general case, we start with an even integer n and a degree-$(n-1)$ polynomial $P(x)$ that we want to evaluate at n paired points $\pm x_1$, $\pm x_2$, $\ldots$, $\pm x_{n/2}$. We partition $P(x)$ into two, degree $n/2-1$ polynomials $P_e(x^2)$ and $P_o(x^2)$. We then compute $P_e(x_1^2)$ and $x_1 P_o(x_1^2)$ and apply Equation (4.14) to evaluate $P(x_1) = P_e(x_1^2) + x_1 P_o(x_1^2)$. Once this is done, we can evaluate $P(-x_1) = P_e(x_1^2) - x_1 P_o(x_1^2)$ at the extra cost of one addition and one subtraction. The total cost to evaluate $P(x)$ at the two points $\pm x_1$ is therefore (1) the costs to evaluate the two, smaller-degree $P_e(x_1^2)$ and $x_1 P_o(x_1^2)$ once, and (2) an addition and a subtraction. This is repeated for the remaining pairs of points.

The idea of partitioning the original, high-degree polynomial into two smaller polynomials leads naturally to the second of the simple, ingenious ideas that make the FFT fast, namely recursion. What if we partition each of the two, degree $n/2-1$ polynomials into smaller, degree $n/4-1$ children polynomials and repeat this process recursively until only polynomials of degree 1 remain? (An important note! In order for all those smaller polynomials to have integer degrees, n itself must be a power of 2. Thus, the recursive FFT version described here is limited to such values of n.)

Now that we decided to use recursion, our problem can be stated as follows: We start with a degree-n polynomial $P(x)$ that we want to evaluate at n points in order to find its value representation. Our thinking went through the following three stages

1. Partition $P(x)$ into two, degree-$n/2$ children polynomials $P_e(x^2)$ and $P_o(x^2)$, both of which are even.

2. Evaluate each of these children on a set of $\pm$ paired points of the form ($\pm x_1$, $\pm x_2$, $\ldots$, $\pm x_{n/2}$). This results in two sets of values

$$\left[P_e(x_1^2), P_e(x_2^2), \ldots, P_e(x_{n/2}^2)\right] \quad \text{and} \quad \left[P_o(x_1^2), P_o(x_2^2), \ldots, P_o(x_{n/2}^2)\right]. \tag{4.15}$$

Now employ Equation (4.14) on each pair of values $P_e(x_i^2)$ and $P_o(x_i^2)$ twice, to compute points $P(x_i)$ and $P(-x_i)$ at the cost of one multiplication, one addition, and one subtraction.

3. This process can be speeded up if instead of evaluating each of the children $P_e(x^2)$ and $P_o(x^2)$ on a set of $n/2$ paired points, we partition each child into two smaller, degree-$n/4$, grandchildren polynomials, and then come back and use those four to evaluate the children $P_e(x^2)$ and $P_o(x^2)$. This naturally introduces the idea of recursion. Instead

of evaluating the four new, degree-$n/4$, grandchildren polynomials, we can partition each into smaller, degree-$n/8$ polynomials, and continue partitioning recursively, without evaluating, until we end up with n degree-1 polynomials. Those are trivial to evaluate, and then the recursion can return step by step, evaluating higher-degree polynomials at each step with one multiplication, one addition, and one subtraction

These are three stages of algorithm development that seemed promising, but once we try implementing these ideas we run into a problem. Consider the stage where we partition the two degree-$n/2$ children polynomials $P_e(x^2)$ and $P_o(x^2)$ into four smaller ones. Once the recursion goes all the way to the end (degree-1 polynomials) and arrives back at the children $P_e(x^2)$ and $P_o(x^2)$, we realize that each needs to be evaluated at the $n/2$ points x_1^2, x_2^2, up to $x_{n/2}^2$ and those points are unpaired.

This may be confusing, so here is a short summary. The original polynomial $P(x)$ had to be evaluated at the set of paired points $\pm x_i$ and this was fast and easy to do with Equation (4.14) from the children $P_e(x^2)$ and $P_o(x^2)$. Each child was evaluated at the $n/2$ points x_i^2, and the final result was the set of n points listed in Equation (4.15). At that time, we could have evaluated $P_e(x^2)$ and $P_o(x^2)$ at all the x_i^2 directly, but we decided to try the recursive approach. With this approach, we plan to evaluate the two children from their four grandchildren, and the problem is that points x_i^2 are unpaired; in fact, they are nonnegative! We can benefit from the ideas of stage 2 above only if we evaluate a polynomial at a set of paired points, so our new problem is to find a set of $\pm x_i$ paired points that will remain paired (identical but with opposite signs) when they become polynomial values such as $P_e(x_i^2)$ and $P_o(x_i^2)$.

A good way to visualize such an initial set of points is to examine a simple example. Let's consider the case $n = 4$, i.e., a degree-3 polynomial, such as $P(x) = x^3 + x^2 + x - 1$. We need to compute the values of this $P(x)$ on a set of four paired points, so let's denote them by $\pm x_1$ and $\pm x_2$. In the first recursive step we partition $P(x)$ into an even and odd pair $P_e(x^2)$ and $P_o(x^2)$ of polynomials, which is done by

$$P(x) = (x^2 - 1) + (x^3 + x) = (x^2 - 1) + x(x^2 + 1) = P_e(x) + xP_o(x).$$

Thus, $P_e(x^2) = x - 1$ and $P_o(x^2) = x + 1$. The two should later be evaluated (from their children) both at x_1^2 and x_2^2, and in order to do so with Equation (4.14), x_1^2 and x_2^2 must be a pair, $x_2^2 = -x_1^2$. In the next and last step of the recursion we end up with the single point x_1^4. This structure is illustrated by Figure 4.48(a).

We now try to assign numeric values to this structure of seven points, and the simplest initial choice is to choose 1 as the value of x_1. Figure 4.48(b) shows the results of this choice. Point $-x_1$ must be -1, causing the pair x_1^2 and $-x_1^2$ to also be 1 and -1. At the bottom level there is only one point, so its value is also 1. This leaves the initial pair x_2 and $-x_2$ and the figure shows that x_2^2 must be -1, which implies the obvious choice $x_2 = i$. This is how complex numbers enter the FFT.

To generalize this example to arbitrary n, we examine part (c) of Figure 4.48. The bottom level, which is 1, is the value of x_1^4. In the formal language of mathematics, this means that the four numbers at the top must be four solutions of the equation $x_1^4 = 1$. As long as we are limited to real numbers, this equation has only the two solutions $x_1 = \pm 1$, which is why we have to look at a wider number field for our four solutions.

Those familiar with complex numbers may now realize that the four numbers at the

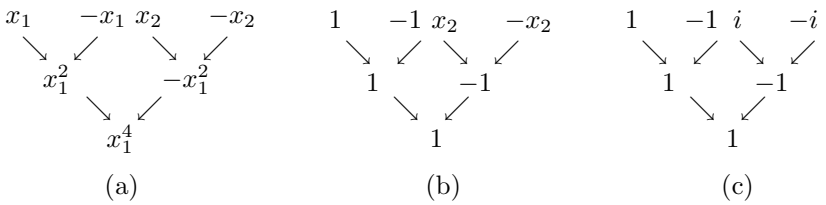


Figure 4.48: Tentative Points for a Two-Step Recursion.

top level are the 4th roots of unity, as discussed on Page 7. It has been mentioned earlier that the recursive version of FFT described here requires n to be a power of 2. It is also known that for each power 2^n there are 2^n nth roots of unity. These facts are what is needed to choose the initial set of $\pm$ paired points. Given a polynomial of degree d, we first choose the smallest n that satisfies $n \geq (d+1)$ as well as $n = 2^k$ for some integer k, and compute the set of nth roots of unity to become the initial points for the first recursion step. (The reader is invited to review the three sets of exponentials that start on Page 105, since those are also roots of unity.)

As explained on Page 7, those roots are the powers $(1, \omega^1, \omega^2, \ldots, \omega^{n-1})$ of $\omega = e^{2\pi i/n}$ and they are $\pm$ paired because $\omega^{j+n/2} = -\omega^j$. The pairing is shown in Figure Intro.4, where lines connect the two points of each pair. In the first recursion step, we select half the initial points and use the sequence $(1, \omega^2, \omega^4, \ldots, \omega^{n/2-1})$ of paired points to evaluate polynomials $P_e(x^2)$ and $P_o(x^2)$. The FFT recursion continues in this way until the last step, where only one point remains.

The discussion so far has explained each of the three main ideas—partition, recursion, and roots of unity—behind the basic, recursive version of the FFT. We are now ready for a summary of the main steps of the algorithm.

We start with (1) choosing a power of 2 as the value of n and (2) a degree-$(n-1)$ polynomial $P(x)$. We define $\omega = e^{2\pi i/n}$ (or, equivalently $\omega = e^{-2\pi i/n}$) and start the recursion. Each recursion step invokes itself recursively twice, once for the even-degree terms of the current polynomial and one for its odd-degree terms. Each of these smaller, degree $n/2$ polynomials is evaluated at $(n/2)$-roots-of-unity points, ending up with the following values

$$y_e = \big[P_e(\omega^0), P_e(\omega^2), \ldots, P_e(\omega^{n-2})\big], \text{ and } y_o = \big[P_o(\omega^0), P_o(\omega^2), \ldots, P_o(\omega^{n-2})\big]. \quad (4.16)$$

These values are saved and are now used to compute the n values of the original, degree-$(n-1)$ polynomial. This is done with the value representation, Equation (4.14), duplicated here with a slight change of index and with a root ω^j instead of an arbitrary, unknown point x_j

$$\begin{aligned}
&P(\omega^j) = P_e(\omega^{2j}) + \omega^j P_o(\omega^{2j}) \quad \text{and} \quad P(-\omega^j) = P_e(\omega^{2j}) - \omega^j P_o(\omega^{2j}),\\
&P(\omega^j) = P_e(\omega^{2j}) + \omega^j P_o(\omega^{2j}) \quad \text{and} \quad P(\omega^{j+n/2}) = P_e(\omega^{2j}) - \omega^j P_o(\omega^{2j})\\
&P(\omega^j) = y_e(j) + \omega^j y_o(j) \quad \text{and} \quad P(\omega^{j+n/2}) = y_e(j) - \omega^j y_o(j),\\
&\quad j = 0, 1, \ldots, (n/2-1).
\end{aligned} \quad (4.14)$$

The second line above is based on the identity $-\omega^j = \omega^{j+n/2}$, that is satisfied by the roots of unity. The third line comes from Equation (4.16).

Once the n values of $P(x)$ at the roots of unity have been evaluated, the algorithm returns, with the value $y = \left[P(\omega^0), P(\omega^1), \ldots, P(\omega^{n-1})\right]$ as its output.

The excellent references [reducible 20a,b] cover this material in video form and include detailed flowcharts and a step by step example of recursion.

4.9 Postscript

Fourier methods prove that, at the lowest level, one-dimensional signals, two-dimensional images, and even three-dimensional objects consist of sinusoidal waves, but why? A possible explanation is that the laws of nature can very often be expressed in terms of partial differential equations. Such equations are difficult to solve, but they contain functions of physical properties such as size, mass, density, and electrical conductivity. These functions have coefficients, and if those coefficients are constant in space and time, the solutions of such an equation are in the form of exponential and sinusoidal functions which describe waves propagating in space and time. This may be the reason why our universe can be expressed in terms of waves.

—Jon Claerbout, *Fourier Transforms and Waves*, 1999.

Fourier's theorem is not only one of the most beautiful results of modern mathematics, but it may be said to furnish an indispensable instrument in the treatment of nearly every recondite question in modern physics.

—Ron Bracewell.

5
The Sampling Theorem

Many historians agree that the first industrial revolution started in 1776, when James Watt introduced his design of the steam engine commercially. This allowed manufacturers to gradually switch from manual production to machine fabrication using steam and water power. The use of machines was first adopted by textile manufacturing, followed by the iron industry, agriculture, and mining.

Similarly, it makes sense to claim that the second industrial revolution started in 1869, when the first transcontinental railroad was completed, joining the two coasts of the United States. The transcontinental telegraph, which went together with the railroad, is also a reason to consider the period 1869 to 1914, revolutionary.

The modern electronic computer was first envisioned by Charles Babbage around 1837. The first steps toward building such a machine came during World War II, when Alan Turing and Tommy Flowers constructed special-purpose computers for enemy code breaking. After the war, several researchers started designing and building digital computers, but the third industrial revolution, the digital revolution, had to wait until around 1968–1969. It was at that time that two little-known developments gave us computer networks and personal computers, and in this way started a revolution that took the world by surprise as well as by storm.

The story of computer networks and the Internet has been well documented. See, for example, Wikipedia article "History of the Internet." The personal computer came about when Busicom (Bijikon Kabushiki-gaisha), a small Japanese company specializing in computer-related products, decided to make a very small calculator, a novelty item. Not knowing how to squeeze a screen, keypad, battery, and electronics into a small package, in 1968 Busicom executives went to Intel, a semiconductor company in Santa Clara, California, asking for help. The job was assigned to "Ted" Hoff who decided to design the electronic circuits as a (microprogrammed) computer. The resulting device, known as the Intel 4004, became the first microprocessor and its more powerful successors started powering the personal computers that immediately followed.

This is why computer networks, combined with our powerful personal computers and with mathematical theories, have changed the way we process information from analog to digital and gave meaning to the term digital revolution or the third industrial revolution.

5.1 The Sampling Theorem

This section introduces the sampling theorem, one of those magical, unexpected mathematical results that affect and improve our lives even though most are unaware of their existence. The sampling theorem claims that we can scan an image, convert it into pixels, and still be able (if we do it right) to compute the colors of image points between pixels. This is possible if the image is scanned at a high-enough resolution. Images are two dimensional, but the sampling theorem holds for any number of dimensions, which is why it is considered the most important idea that made our vast digital media world possible.

Because of its importance, we try to state the essence of the sampling theorem in various ways as follows: (1) It proves that the digital world of bits and pixels (which we cannot see or hear) can accurately represent the real world, which is basically analog and is visible. (2) It shows that discrete data (a finite set of numbers) can faithfully represent the solid, smooth, and uncountably ($\aleph_1$) continuous objects we are familiar with, regardless of their number of dimensions. (3) It implies that we can get rid of a vast amount of information without losing anything, because we can always reconstruct the original information perfectly; it's like getting something from nothing. (4) When we carry a CD with music on it, we do not carry the music itself. We carry a large set of numbers, the audio samples (or a compressed version of an audio file) that can be processed to produce and play the music. In contrast, when we carry an old, analog vinyl record, we carry the infinite analog representation of the sound (actually, a small subset of it, due to the limited precision of the track on vinyl). (5) The sampling theorem is not just an existence theorem, because it shows exactly how to achieve the surprising, magical goal of ideal sampling and perfect reconstruction.

We first illustrate the sampling theorem with audio, which is one dimensional. The following exchange is typical:

John: "No, officer, I didn't say that. I said I *thought* it was him, but it was dark and I couldn't be sure."

Officer: "I heard you say you saw him."

Anna: "No, officer. I'm sure he only said he may have seen him."

Sound is fleeting. You say something and it is immediately gone. People hear you, they think they understand you, but just a few seconds later they may not agree on exactly what they have heard. This is why the invention of the phonograph (by Thomas Edison in 1877), was so revolutionary. For the first time in human history it became possible to record sound, store it indefinitely, and play it back. Following Edison, many engineers and inventors developed advanced techniques and media for sound recording and constantly improved the results. Sound was recorded on wax cylinders, electrical wires, steel tape, magnetic disks, and magnetic tapes, but all these devices were analog.

The first successful digital sound recording was the CD format, developed and adopted by a consortium of several manufacturers in the 1980's. For the first time, it became possible to record sound at a high fidelity and copy it many times without degradation of quality.

The main shortcoming of any analog representation is the gradual degradation of copies. You write something on paper, place it in a copy machine and copy it easily, but the copy is not as sharp as the original, and copies of this copy feature more and more degradation. On the other hand, if you write text on the computer, and save it as a text file, you can copy it and copy its copies indefinitely without any loss of quality. Each copy is identical to the original file. Thus, a chief advantage of a digital representation of any data is the ability to make true copies.

Sound is a wave in the air. When we generate sound, air molecules move back and forth, forming a wave in the air. This sound wave is an analog representation of the sound. When a microphone picks up a sound wave, it converts it to an electrical wave, a voltage that varies with time in accordance with the sound. This wave is another analog representation of sound. Digitizing sound, converting its representation from analog to digital, is done by measuring the amplitude (the voltage) of this wave many times each second. Each voltage measurement is converted to a number, an audio sample, and those numbers constitute an audio file.

(The terms voxel and soxel are derived from pixel. The former is a small unit of volume, while the latter, which is introduced in the excellent book [Smith, Alvy Ray 21], is a contraction of sonic element, and denotes an audio sample.)

Figure 5.1 illustrates a simple example. A very small part of a sound wave is sampled and is converted to about two dozen audio samples. The magnitude of each sample is indicated in red. It is clear that there are not enough samples to reconstruct the original sound wave. Many parts of the wave (some are marked by green boxes) cannot be predicted from our samples. Thus, the most important question in digitizing sound is how often to sample a given sound wave.

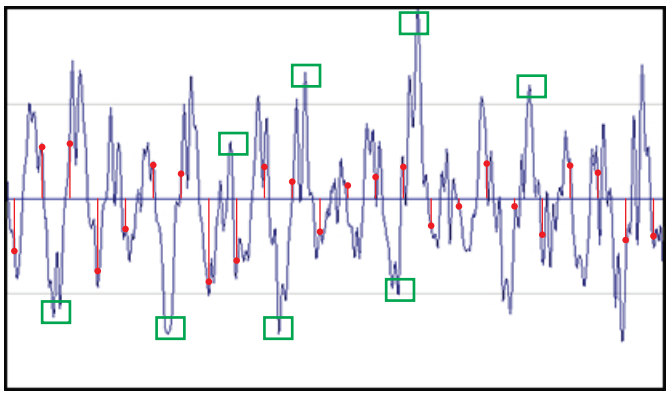


Figure 5.1: Sampling a Sound Wave.

It is intuitively clear that the sampling rate must depend on the frequency of the sound, as illustrated by Figure 5.2. The figure shows a simple sine wave that is sampled

several times. In part (a) of the figure, the wave is sampled once per period, the samples are identical, and the simplest reconstruction (in green) is a constant signal. Naturally, this is very different from the original sine wave. In part (b) of the figure, three samples are taken for every two periods of the wave, and the most obvious reconstruction is a sine wave with a frequency that is much lower than that of the original. Part (c) shows what happens when two samples are taken in each period of the wave. The result is a sine wave with the same frequency but a much smaller amplitude. Only part (d), where eight samples are taken, seems promising. At least, it is not immediately clear how anything other than the original sine wave can be reconstructed from those samples.

A digression. The black and green waves of parts (a) through (c) of Figure 5.2 are different, but because of the low sampling rate they cannot be distinguished. Given the set of samples, we can use it to reconstruct either wave. This is a case of *aliasing*. Here is what wikipedia has to say about this term "In signal processing and related fields, aliasing is an effect that causes different signals to become indistinguishable (or aliases of one another) when sampled." Thus, when a sound wave is sampled at a rate that is too low, the reconstructed sound is different from the original sound and features low frequencies. When an image is sampled (scanned or photographed) at a low rate, its reconstruction also features low pixel frequencies. Such frequencies are associated with important image features, which is why aliasing causes significant distortions. In his outstanding lectures on photography, Marc Levoy defines aliasing as high frequencies masquerading as low frequencies due to insufficiently closely-spaced samples (lecture 7 of [Levoy 16]). **End of digression**.

◇ **Exercise 5.1:** Show simple examples of spatial and temporal aliasing (aliasing in space and in time). The former should be a simple image where high frequencies combine to form low frequencies. The latter should be a simple system that moves one way in time, but seems to slowly go back in time when we sample it at a low rate.

We are now ready for the sampling theorem. Most references that discuss this topic refer to it as the Shannon-Nyquist sampling theorem [Nyquist 15] and claim that it is due to Harry Nyquist and Claude Shannon. However, the excellent book [Smith, Alvy Ray 21] corrects this tradition and informs us that the originator of this important theorem, in 1933, was the Soviet engineer and scientist Vladimir Kotelnikov (Figure 5.3).

Later researchers put this result on a more solid theoretical foundation, but its basic claim hasn't changed. The sampling theorem states that any wave can be reconstructed perfectly from a finite set of samples if the sampling rate is greater than twice the maximum frequency (the cutoff frequency or the Nyquist rate or the Nyquist frequency) of the wave.

A rigorous proof of this theorem is outside the scope of this book, but Subsection 5.2 attempts to give the reader an overall picture of such a proof in two steps, the first is mathematical and the second is intuitive. The claim of the sampling theorem is simple and it comes as a surprise. We intuitively feel that a perfect reconstruction of a continuous analog wavy (undulating) signal requires infinitely many samples, but the theorem says that a wavy signal featuring any frequencies can be fully reconstructed from a finite number of samples. This sounds like getting something for nothing. How can a finite number of samples represent the total information included in a continuous analog signal which consists of infinitely many points? The explanation is that the

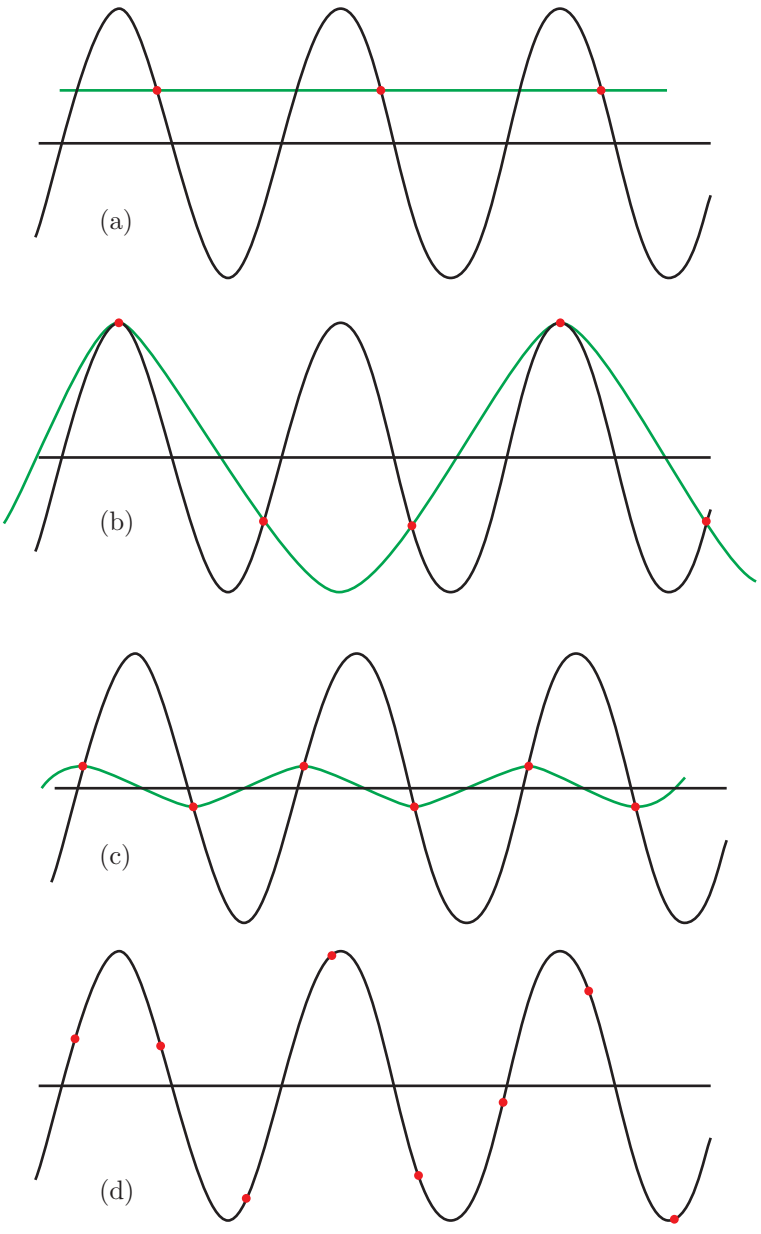


Figure 5.2: Sampling a Sine Wave.

Figure 5.3: The Fathers of the Sampling Theorem.

process of reconstructing and computing the infinite number of points of the original, continuous signal requires an infinite amount of computations, thereby cancelling out any advantage obtained when the signal was originally sampled (when its original, analog infinity was repackaged in a finite form). However, in practice we skip most of these computations and obtain a very close approximation of the original signal by performing a finite number of steps. The following discussion sheds more light on the meaning of the sampling theorem.

Chapter 4 discusses the Fourier transform and explains how a continuous, analog periodic signal can be represented as an infinite sum of pure sine waves of increasing frequencies. Each wave is fully described by its amplitude and phase. The frequencies do not have to be explicitly stated because they are implied in the sum. This is another example of throwing away an infinite amount of information without losing anything. The original, analog signal contains an uncountable ($\aleph_1$) amount of point information, while its transform consists of a finite amount, or at most a countable ($\aleph_0$) amount of numbers, namely two per sine wave. Thus, in much the same way that Newton stood on the shoulders of giants, Nyquist, Shannon, and Kotelnikov stand on the shoulders of

Fourier.

Imagine that we convert each audio sample to a two-dimensional point as follows. The value of the sample becomes the y coordinate of the point, while its x coordinate is the time the sample was taken. The original sound wave can be visualized as a two-dimensional curve with time as the x axis, and the problem of reconstructing the sound wave is now equivalent to finding all the curves that pass through the points and selecting the right one. The simplest way to generate a curve is to connect the points with straight segments, in the same way as the game of connect the dots. The result is rarely the right curve, and Subsection 5.2 shows how to interpolate the points in order to end up with the original curve.

Given a sound wave whose frequency is less than 1,000 Hz, we sample it at 2,100 Hz and end up with 2,100 samples for each second of sound. The theorem guarantees that this number of samples would be enough to fully reconstruct the wave. It is pointless to sample at a higher rate, because out of all the curves that pass through all the 2,100 sampled points, there would be only one curve whose frequency is less than 1,000 Hz.

Often, it is not known in advance what frequencies are included in a wave and what its maximum frequency is. In such a case, the best that we can do is to estimate, digitize the wave, and try to reconstruct it. Sound waves are all different and each may consist of a mixture of many frequencies. We don't know in advance what frequencies are included in a given sound wave, but we can use our knowledge of human physiology to solve this problem.

The range of human hearing is typically from 16–20 Hz to 20,000–22,000 Hz, depending on the person and on age. When sound is digitized at high fidelity, it should therefore be sampled at a little over the Nyquist rate of $2\times22{,}000 = 44{,}000$ Hz. This is why high-quality digital sound is based on (at least) a 44,100-Hz sampling rate. Anything lower than this rate may result in distortions, while higher sampling rates do not improve the reconstruction (playback) of the sound. We can consider the sampling rate of 44,100 Hz a lowpass filter, since it effectively removes all the frequencies above 22,000 Hz.

Applying the sampling theorem to audio is simple, because audio is one dimensional. Our interest, however, is in images, which are two dimensional. It turns out that images also feature frequencies and therefore obey the sampling theorem. The concept of pixel frequencies is easy to understand when we consider bi-level (black-and-white) images. Figure 5.4 shows a small, 5×8 bi-level image that illustrates pixel frequencies. The top row of this image is uniform, so we can assign it zero frequency. The following rows feature increasing pixel frequencies as measured by the number of color changes along a row. The four waves on the right roughly correspond to the frequencies of the four top rows of the image.

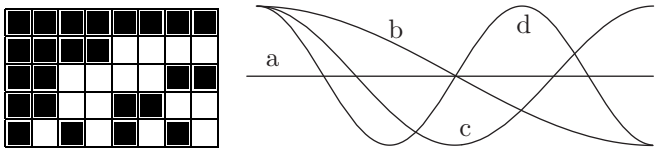


Figure 5.4: Image Frequencies.

A digression. Image frequencies are important because they make it possible to distinguish between the important features of an image and the features that are mere details. Low pixel frequencies correspond to the important image features, while high frequencies correspond to the details of the image, which are generally less important. This fact is the basis of image compression by means of transforms. (End of digression.)

Today (in 2022), bi-level images are fairly rare and most digital images are in color. A pixel in a color image consists of three parts, for the three color components. Given a color image, we start by separating the three parts and considering each a grayscale image. Thus, we need to consider how to measure pixel frequencies in a grayscale image. We assume that a grayscale image is given where a pixel can have values from 0 (black) to 255 (white). The frequencies of pixels in this image can be measured in several ways as follows:

- If each row of the image consists of the n pixels P_1 through P_n, we first convert them to the two-dimensional points $(1, P_1)$ through (n, P_n) and then compute the degree-$(n-1)$ Lagrange polynomial (Subsection 4.8.2) that passes through the points. Fourier analysis is then used to break down this curve into its constituent sine waves.

- Apply the discrete cosine transform (see Wikipedia) to all the 8×8 blocks of the image. In each block, the 63 AC coefficients of this transform indicate the strengths of the various pixel frequencies in the block. Specifically, if a certain pixel frequency does not appear in the block, its AC coefficient will be zero. This makes it easy to choose the maximum pixel frequency of each block.

- Measure the size (in millimeters) of the smallest discernible detail in the image. This can often be done manually, by looking through a magnifying glass and measuring distances on the image with a ruler. If the size of this detail (its smallest dimension) is d, we can define the highest frequency of the image as $1/d$ cycles-per-mm and scan it at a resolution of $\lceil(2/d)+\epsilon\rceil$ gray-level samples per mm, where ϵ is a number much smaller than $1/d$.

  In practice, however, we are restricted by the fixed resolution of our devices. When scanning an image, the resolution of the scanner is either fixed, or is limited to a few values such as 300, 600, and 1200 samples per inch. If we use a camera, the image resolution is simply the resolution of its image sensor, which is high, but fixed.

- Measure the smallest discernible change in the gray level. However, such measurements are highly subjective and are better done by machine, a scanner. We denote the gray levels from 0 (black) to 1 (white). If the smallest discernible change in the gray level is g, then we can assume that the highest image frequency is $1/g$ and we need to scan the image at a resolution of $\lceil(2/g)+\epsilon\rceil$ gray-levels. Thus, for example, if $g = 0.0085$, meaning that our scanner can discern gray levels as close as x and $x + 0.0085$, then $2/g \approx 235.29$ and we should scan at a resolution of 236 gray levels. In practice, this would mean $2^8 = 256$ levels or eight bits per sample, but again we are limited to the resolution of the scanner at hand.

- The approaches above are not very practical. In practice, both digital cameras and scanners capture images on image sensors, which consist of discrete photosites. Thus, instead of measuring pixel frequencies, we consider the smallest detail that we want to

be able to discern in an image. Similarly, instead of computing a sampling rate that is greater than twice the maximum pixel frequency, we compare the size d of the smallest detail to the size f of a photosite. Once we migrate from pixel frequencies to a size d and from sampling rates to the photosite size f, the sampling theorem becomes the simple relation $d > 2f$. The image must be scaled (or zoomed) such that the smallest detail we are interested in covers more than two photosites. Figure 5.5 illustrates various relations between d (the diameter of the white circle, our detail) and f (the size of the square photosites).

In part (a) of the figure, the smallest detail is the same size as a photosite. The detail is captured as a single square photosite, very different from a circle. This capturing of the small detail is also inaccurate on the right-hand side of part (a), where the circle falls on the intersection of photosites. The circle is captured as a large, dim square because its light energy is now split among four pixels.

In part (b) of the figure, the smallest detail (the white circle) is the size of two photosites. On the left-hand side, the circle is captured by four pixels, resulting in a large gray square, significantly different from the circle. On the right side, however, the circle is centered on a photosite and is captured by a group of 3×3 pixels that better approximates its round shape.

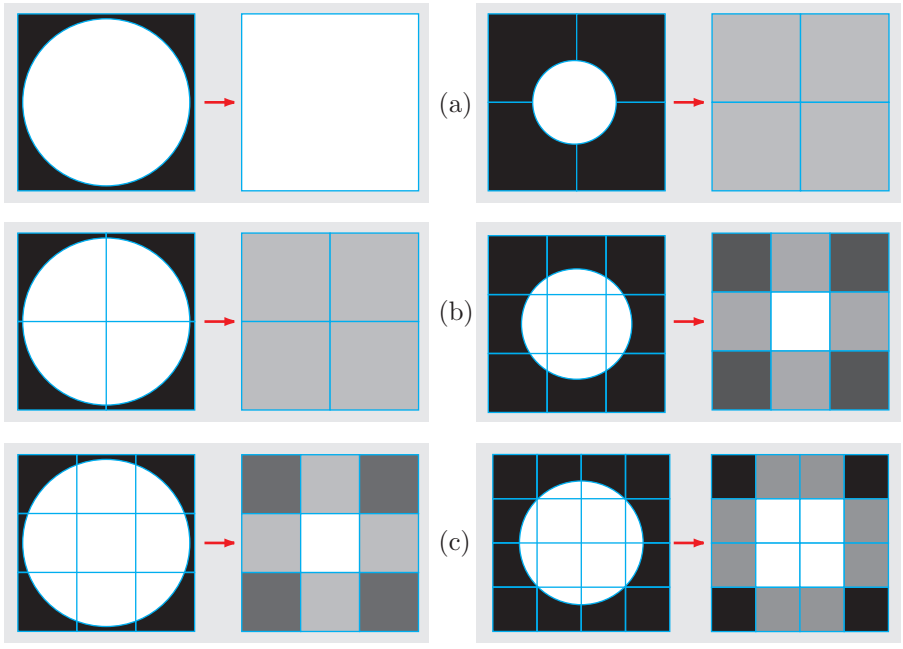


Figure 5.5: Three Scan Resolutions.

In part (c) of the figure, the smallest detail (the white circle) is the size of three photosites. Its round shape is captured on both sides of part (c). On the left, it is centered on a photosite and is captured by a group of 3×3 pixels. On the right, it is

centered on the intersection of four photosites and its shape is reflected even better by a group of 4×4 pixels.

The advantage of the sampling theorem is now obvious. This theorem makes it possible to enlarge a continuous image to any size and without any loss of detail by scanning (or sampling) it at a high rate. Imagine dragging a large camera with a 4000 Mpixel sensor up the Acropolis in Athens, and taking a picture of a large part of the city below. The resulting digital image would contain enough pixels to easily distinguish small details, such as people and cars, seen in the distance. In general, if a continuous image is photographed or scanned at high-enough resolution, it is possible to compute the image intensity and color at any point between adjacent pixels and in this way to create as many new pixels as desired. The end result of such a process is a large image that is as close to the real, continuous image as desired. A surprising, unexpected, and useful result.

5.2 The Sampling Theorem: Reconstruction

Once an analog signal has been sampled, saved, and possibly also edited, a time may come when it has to be reconstructed. This is the magical process where a finite set of numbers is converted to a continuous, analog signal containing, in principle, an uncountable infinity of values. The discussion here consists of three parts. The first part looks at signal reconstruction as a process of interpolation. The second part attempts to be rigorous and uses mathematics. The third part is intuitive and relies on diagrams. First, some terms used in the field of signal processing.

- We can think of a time signal as a point moving in an ambient (i.e., boundless) space, leaving a trace behind it, which ends up as a graph. If we know the mathematical expression of this graph, we say that the point is a dynamic system.

- The term impulse response (or impulse response function), of a dynamic system is the reaction of the system in response to some external change. More accurately, the impulse response is the output resulting from multiplying the system by an impulse (a brief input signal).

- The unit impulse function, also referred to as the Dirac delta (δ) function, is an unusual mathematical object. We can think of it as a curve whose value is zero everywhere except at $x = 0$, where it is infinite, and whose integral over the entire real number line is equal to 1. It can be described naively as

$$\delta(x) = \begin{cases} +\infty, & x = 0 \\ 0, & x \neq 0 \end{cases}, \quad \text{and} \quad \int_{-\infty}^{\infty} \delta(x)dx = 1,$$

but it is wrong to say that a mathematical quantity equals infinity because infinity is not a number. It is therefore common to define the delta function as the limit

$$\delta(t) = \lim_{a\to\infty} a\, e^{-\pi(ax)^2},$$

```
(*Dirac delta definition*)
del[t_, a_] := a Exp[-Pi (a t)^2];
Plot[del[t, 12], {t, -1, 1}, Filling -> Axis, PlotRange -> All]
Assuming[a > 0, Integrate[del[t, a], {t, -Infinity, Infinity}]]
```

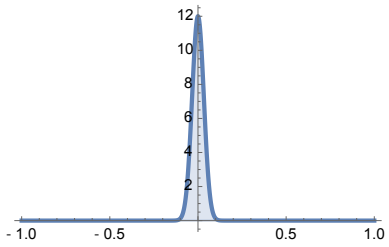


Figure 5.6: The Dirac delta as a Limit

which is illustrated by Figure 5.6. The figure also shows that the area under the delta curve equals 1 regardless of the value of a.

In engineering diagrams the Dirac delta is depicted as a vertical arrow of any length and can be shifted horizontally such that the arrow may be located at any point on the x axis (which usually serves as the time axis). The delta function is used by scientists to model abstractions such as a point charge, point mass, or electron point. Engineers use it to model a tall narrow spike function (an impulse), which is why they often refer to it as the unit impulse symbol. We normally refer to $\delta(x)$ as a function for historical reasons, even though it is not a function in the conventional sense. A mathematical function must have a definite value at each point in its domain, but $\delta(x)$ approaches infinity at $x = 0$ and therefore cannot be used in mathematical analysis like an ordinary function.

A note. The delta function and the familiar cosine are dual in the sense that each is the Fourier transform of the other, as is illustrated in Figure 5.7. To prove this little-known relation, we first point out that the famous Euler formula implies

$$\cos(\omega t) = \frac{1}{2}(e^{i\omega t} + e^{-i\omega t}).$$

From this, we show that the Fourier transform of $x(t) = \cos(\omega_0 t)$ is

$$X(\omega) = \pi\big(\delta(\omega - \omega_0) + \delta(\omega + \omega_0)\big),$$

and we do this by considering the inverse Fourier tranform of $X(\omega)$

$$\begin{aligned}\mathcal{F}^{-1}\{X(\omega)\} &= \frac{1}{2\pi}\int_{-\infty}^{\infty} X(\omega)e^{i\omega t}d\omega \\ &= \frac{1}{2\pi}\left[\int_{-\infty}^{\infty}\pi\delta(\omega-\omega_0)e^{i\omega t}d\omega + \int_{-\infty}^{\infty}\pi\delta(\omega+\omega_0)e^{i\omega t}d\omega\right] \\ &= \frac{1}{2}(e^{i\omega_0 t}+e^{-i\omega_0 t}) \\ &= \cos(\omega_0 t).\end{aligned}$$

Once this relation is clear, we start with $\cos(t)$ in the time domain and pass it through the Fourier transform to end up with the frequency-domain function $X(\omega)$ above, which is a delta.

$$X(\omega) = \int_{-\infty}^{\infty}\cos(t)e^{-i\omega t}dt = \int_{-\infty}^{\infty}\cos(t)\cos(\omega t)dt.$$

Figure 5.7 shows the resulting approximations of the delta function. In principle, the limits of the integration should be from $-\infty$ to ∞, but for practical reasons, the four parts of the figure have limits in the intervals $[-t,t]$ for $t = 5$, 10, 15, and 50. Note how the height of the delta equals parameter a. The cosine function is also shown, in yellow, for comparison.

Armed with these terms, we plunge into the first part of the discussion of reconstruction. We are given a one-dimensional continuous time-signal $x_c(t)$ and we already have a set $x[n]$ of samples that satisfy the sampling theorem and that are set T time units apart. We start with simple, intuitive attempts to estimate any point on $x_c(t)$ from the one or two samples nearest it in time.

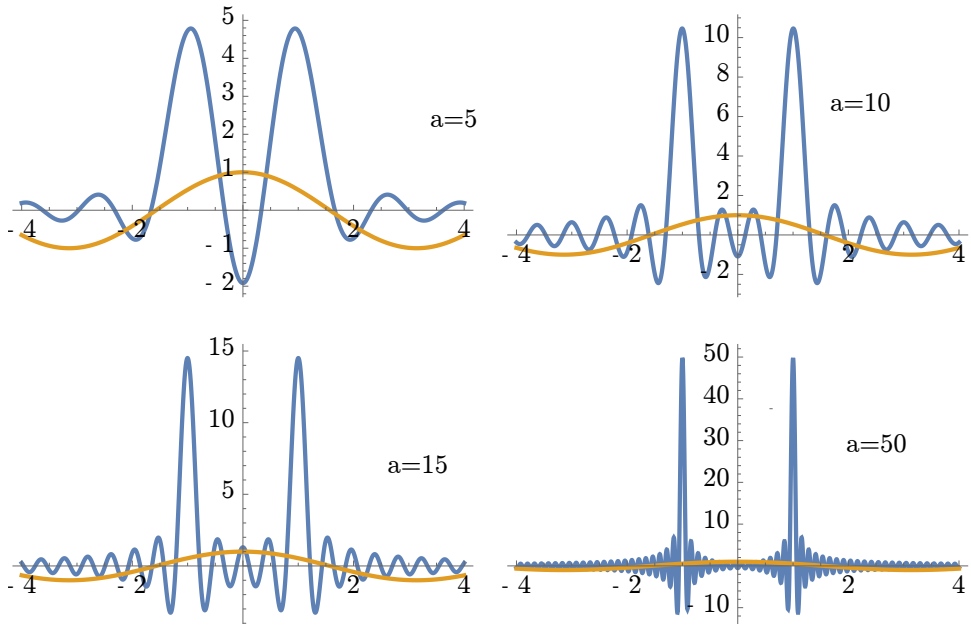
The left column of Figure 5.8 illustrates three such attempts. It is obvious that the nearest neighbor method is almost always wrong because it always relies on a single sample. The green point is estimated as the nearest sample to its right, whereas it is clear that its value should be determined by the two samples surrounding it.

The zero-order hold method is similar, with the only difference being that the thick polygon representing the estimates has been shifted to the right half the distance between samples.

The first-order hold method is much better. It estimates a point on the reconstructed curve by linearly interpolating its two nearest samples.

It is now obvious that a second-order hold method should use a cubic (or other form of) interpolation of three consecutive samples to yield a better estimate of a point on the reconstructed curve. Similarly, higher-order interpolations could use more and more adjacent samples to result in better and better estimates.

The other two columns of Figure 5.8 illustrate how the various hold methods can be expressed formally in terms of the delta function combined with the operation of convolution (Chapter 6). We replace each sample with a delta function of the same height and convolve each delta with an appropriate impulse response filter (shown in the rightmost column). For zero- and first-order holds, the impulse response is a rectangle




```
a = 50;
Plot[{ NIntegrate[Cos[t] Cos[ w t], {t, -a, a}], Cos[w]}, {w, -4, 4},
 BaseStyle -> {FontFamily -> "cmr10", FontSize -> 10},
 PlotRange -> All, AspectRatio -> 27/44, ImageSize -> 72*2.45]
```

Figure 5.7: The delta as the Fourier Transform of a cosine

of width T and height 1, while for the second-order hold, it is a triangle of width $2T$ and height 1.

Figure 5.9a shows several examples of how a series of such triangles, shifted and vertically scaled, can generate each point of the first-order linear interpolation as the sum of points from two triangle sides. Three examples are given, using green circles to indicate points from triangle sides, and how they are added to form a black circle that is located on the reconstructed curve.

It can be shown that perfect reconstruction is achieved when the special sinc function is used as the impulse response filter. This function (Figures 5.9b and 5.10) is defined as $\sin(x)/x$, except for $x = 0$, where it is indeterminate and is defined as 1. It is infinite in the horizontal direction, but its amplitude decays fast as we move away from $x = 0$.

Part (c) of Figure 5.9 illustrates two sinc functions (in red and green) convolved with delta functions. Once each sample is replaced with the convolution of a delta and a sinc, each point on the original, analog curve can be computed as the sum of points on many sinc functions. There is one sinc function for each sample, which is why each point on the reconstructed curve is the sum of points on N sinc functions, where N is the number of samples. The figure shows two such points, in red and green. In practice,

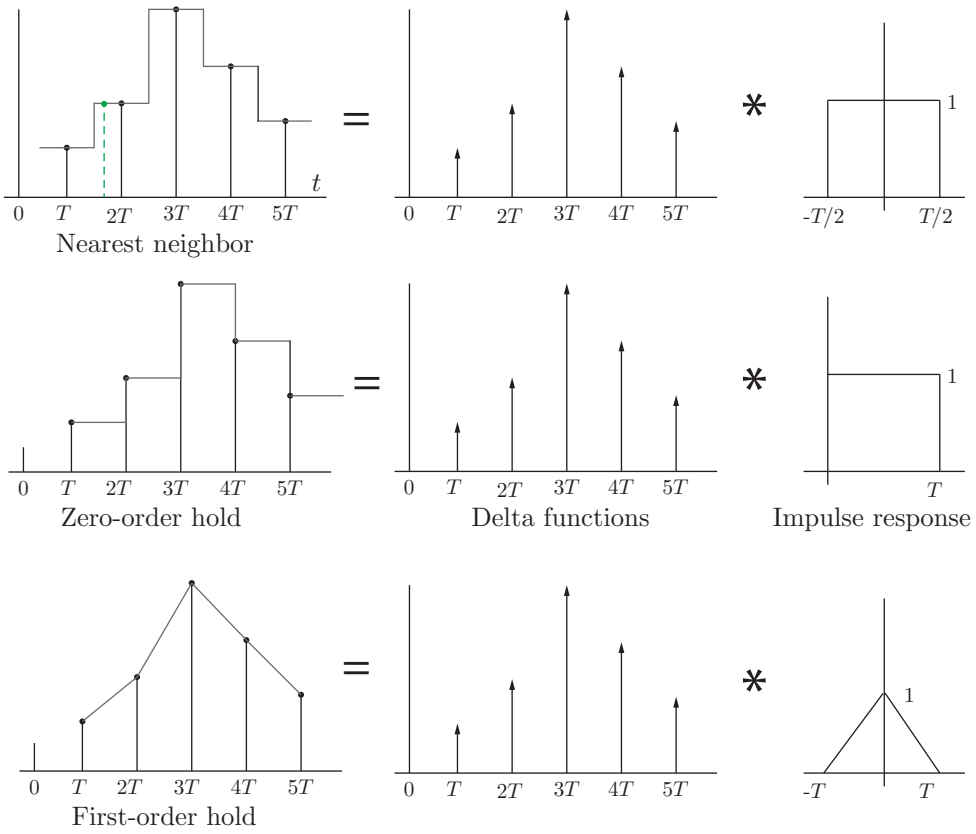


Figure 5.8: Intuitive Reconstruction.

N is normally very large, which is why a point on the reconstructed curve is computed as the sum of only a few functions, those that correspond to nearby samples.

Finally, Figure 5.9d shows the 2D sinc function, resembling a sombrero, which is used in reconstructing images from their samples. Its equation is $\text{sinc}(\sqrt{x^2+y^2})$ and it is used repeatedly on each point sample in an array of samples (pixels). Its effect is to spread each pixel based on all (in practice, only several of) the other pixels in the image.

Now, for the second part of the discussion of signal reconstruction, starting with the concept of impulse sampling of a continuous-time signal. This term refers to how we represent the sampling process of a signal as the product of the signal with an impulse train. Figure 5.11 shows a continuous-time signal $x(t)$ and a train $p(t)$ of unit impulses separated by a distance T_s (which will later be interpreted as the inverse of the Nyquist frequency f_s). We can consider this train an infinite, continuous, periodic function which is mostly zero and has a pulse of height 1 in each period. Another way of looking at $p(t)$ is as a sifting function whose job is to sift out the values of the original signal $x(t)$ at many points that become samples. The product $x(t) \cdot p(t)$ which is denoted by $x_\delta(t)$, is another train of impulses that become our samples.

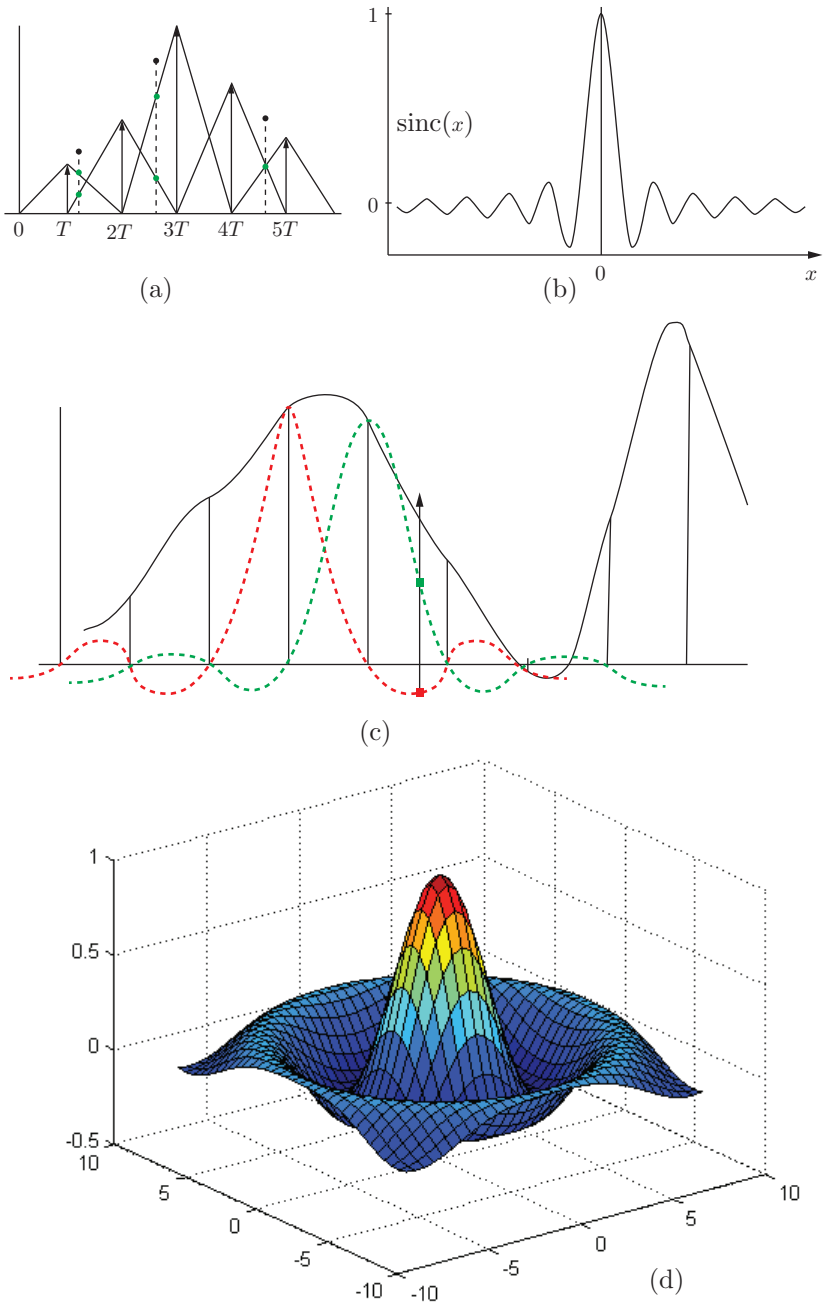


Figure 5.9: Better Reconstruction.

Figure 5.10: The Sinc Function in Real Life.

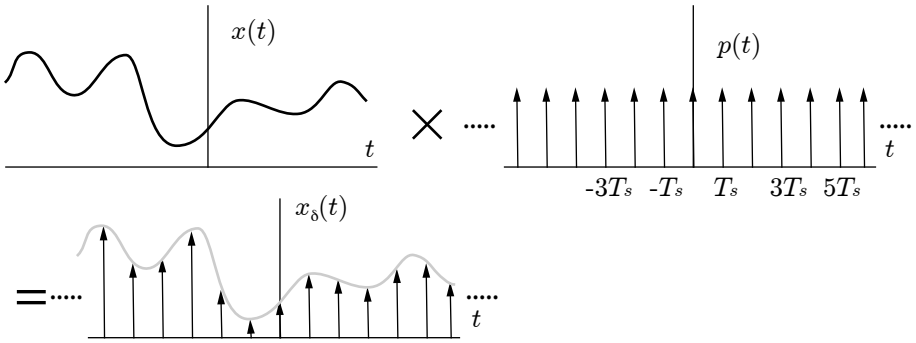


Figure 5.11: Impulse Sampling of a Signal.

The product $x_\delta(t) = x(t) \cdot p(t)$ can be written explicitly in the time domain because $p(t)$ is not simply a bunch of separate impulses but is a continuous, periodic function of unit impulses separated by distance T_s. Thus $p(t) = \sum_{n=-\infty}^{\infty} \delta(t - n\,T_s)$, and

$$x_\delta(t) = x(t) \cdot \sum_{n=-\infty}^{\infty} \delta(t - n\,T_s). \tag{5.1}$$

In order to convert this product to the frequency domain, we use the well-known convolution theorem (see Chapter 6 and especially Section 6.3, for a discussion of convolution) which states that convolution in one domain equals pointwise multiplication in the other domain. Equation (5.1) is a multiplication in the time domain, implying that its frequency domain equivalent is the convolution

$$X_\delta(\omega) = \frac{1}{2\pi} X(\omega) \star P(\omega),$$

where $X(\omega)$ is the Fourier transform (the spectrum) of signal $x(t)$ and $P(\omega)$ is the Fourier transform of $p(t)$.

Since $x(t)$ is a known signal, we can compute its spectrum $X(\omega)$, which leaves us with the task of figuring out the Fourier transform $P(\omega)$ of a pulse train $p(t)$. Since the period of $p(t)$ is $T_s = 1/f_s$, its fundamental frequency is $\omega_0 = 2\pi f_s = 2\pi/T_s$, and we start with the Fourier transform of a single, arbitrary period P_k of $p(t)$, centered on the pulse

$$P_k = \frac{1}{T_s}\int_{-T_s/2}^{T_s/2} e^{-ik\omega_0 t}dt = \frac{1}{T_s}e^{-ik\omega_0 t}\,|_{t=0} = \frac{1}{T_s}.$$

The unexpected result is that the Fourier transform of an arbitrary period P_k of $p(t)$ does not depend on k and is simply a pulse of height $1/T_s = f_s$. The entire transform is an infinite train of such pulses, separated by the fundamental frequency ω_0,

$$P(\omega) = 2\pi \sum_{k=-\infty}^{\infty} P_k\,\delta(\omega - k\,\omega_0) = \frac{2\pi}{T_s}\sum_{k=-\infty}^{\infty}\delta(\omega - k\,\omega_0).$$

Now that we have $P(\omega)$, we can compute the convolution

$$\begin{aligned} X_\delta(\omega) &= \frac{1}{2\pi}X(\omega)\star P(\omega) \\ &= \frac{1}{2\pi}X(\omega)\star\frac{2\pi}{T_s}\sum_{k=-\infty}^{\infty}\delta(\omega - k\,\omega_s) \\ &= \frac{1}{T_s}\sum_{k=-\infty}^{\infty}X(\omega - k\,\omega_s). \end{aligned} \tag{5.2}$$

(This is because convolving with pulses is simple; just place the spectrum $X(\omega)$ in the location of the impulse.) Thus, the sampled spectrum $X_\delta(\omega)$, the frequency domain equivalent of the product $x(t)\cdot p(t)$ consists of an infinite train of copies of the original spectrum $X(\omega)$ of $x(t)$. The copies are scaled down vertically by a factor T_s and are shifted by a multiple of $\omega_s = 2\pi f_s = 2\pi/T_s$. Ideal reconstruction is obtained if f_s is greater than or equals the Nyquist frequency of the original signal $x(t)$. Once $X(\omega)$ has been computed, the entire set of its copies is prepared and can be edited if needed. After editing, this set can be used to reconstruct the edited version of the original signal $x(t)$.

Figure 5.12 illustrates the relation between the single copy of $X(\omega)$ and the train of $X_\delta(\omega)$ of copies. We assume that a time-signal $x(t)$ is given, whose Fourier transform (spectrum) $X(\omega)$ is the triangle of height 1 shown in part (a) of the figure, where ω is the highest frequency contained in the spectrum. Part (b) shows two copies of $X(\omega)$ on both sides of the original triangle. These copies are scaled down vertically by a factor of $1/T_s$ and are separated by a distance of ω_s, which in part (b) happens to be large (larger than 2ω), thereby resulting in copies that are widely separated. Part (c) of the figure shows the train of copies for the case where $\omega_s = 2\omega$, the Nyquist frequency. Here, the individual copies of $X(\omega)$ butt each other perfectly.

When the user chooses a small value for ω_s, a value less than the Nyquist value of 2ω, the individual copies of the spectrum may overlap, as shown in part (d) of the

figure. In such a case of undersampling, the original spectrum is partly obscured by copies, making it impossible to recognize its original shape, which leads to errors in the reconstruction. Such a case is referred to as aliasing, and it has been mentioned elsewhere that in his excellent lectures, Marc Levoy defines aliasing as high frequencies masquerading as low frequencies due to insufficiently closely-spaced samples.

In order to understand the reconstruction, we assume no editing. The task of reconstruction is simply to convert the set $X_\delta(\omega)$ of copies (the sampled spectrum) to the original spectrum $X(\omega)$, which is a single copy. Once this is done, the resulting original spectrum is passed through the inverse Fourier transform to obtain the original temporal signal $x(t)$. Converting $X_\delta(\omega)$ to $X(\omega)$ requires getting rid of all the copies except the central one, and scaling this copy vertically by a factor of T_s. This is accomplished by a rectangular filter $H_r(\omega)$ with the frequency response: height T_s and width in the frequency interval $[-\omega_s/2, \omega_s/2]$. Mathematically, since we are still in the frequency domain, we express the filtering by the equation $X(\omega) = H_r(\omega) \cdot X_\delta(\omega)$. Applying again the convolution theorem, Section 6.3, we conclude that going back to the time domain and reconstructing the original signal is done by the convolution $x(t) = h_r(t) \star x_\delta(t)$ where $x_\delta(t)$ is given by Equation (5.1) and $h_r(t)$ is the time-domain equivalent of the rectangular filter. The example discussed on Page 52 shows that this equivalent is the sinc function, defined on Page 136. Our particular rectangular filter is a function of ω, has a height of T_s and width of ω_s. A search in a table of Fourier transforms indicates that it converts to the specific sinc function $T_s \cdot \frac{\omega_s}{2\pi} \cdot \text{sinc}(\frac{\omega_s}{2}t)$. Thus, we end up with

$$\begin{aligned} x(t) &= h_r(t) \star x_\delta(t) \\ &= \text{sinc}(\frac{\omega_s}{2}t) \star \sum_{n=-\infty}^{\infty} x(n\,T_s)\delta(t - n\,T_s) \\ &= \sum_{n=-\infty}^{\infty} x(n\,T_s)\text{sinc}[\omega_s(t - n\,T_s)/2]. \end{aligned} \tag{5.3}$$

(This is because $\omega_s = 2\pi f_s = 2\pi/T_s$.)

Equation (5.3) is an infinite weighted sum of time-shifted sinc functions. Technically, it is referred to as an ideal band-limited (or I.B.L) interpolation.

This is the mathematics of sample reconstruction. It shows that the sinc function is the ideal tool for perfect reconstruction of an analog signal, but it (the mathematics) may be more than the average reader cares to know. Earlier, I promised a three-part explanation of reconstruction, so here is the third part, where the process described by Equation (5.3) is explained intuitively, with the use of figures and "hand waiving."

The discussion is based on the excellent 2021 book *A Biography of the Pixel*, by Alvy Ray Smith, reference [Smith, Alvy Ray 21], where the reconstruction is explained with the help of the *spreader*, a simple, well-known, and very useful function. In mathematics, the term sinc is used for the spreader, while engineers call it a reconstruction filter.

The basic (unnormalized) sinc function is defined as $\text{sinc}(x) = \sin(x)/x$ and is illustrated in red in Figure 5.13. This expression is clearly undefined for $x = 0$, which is why the complete definition includes $\text{sinc}(0) = 1$, because 1 is the limit of $\text{sinc}(x)$ for x approaching zero. The spreader that is actually used in digital reconstruction is

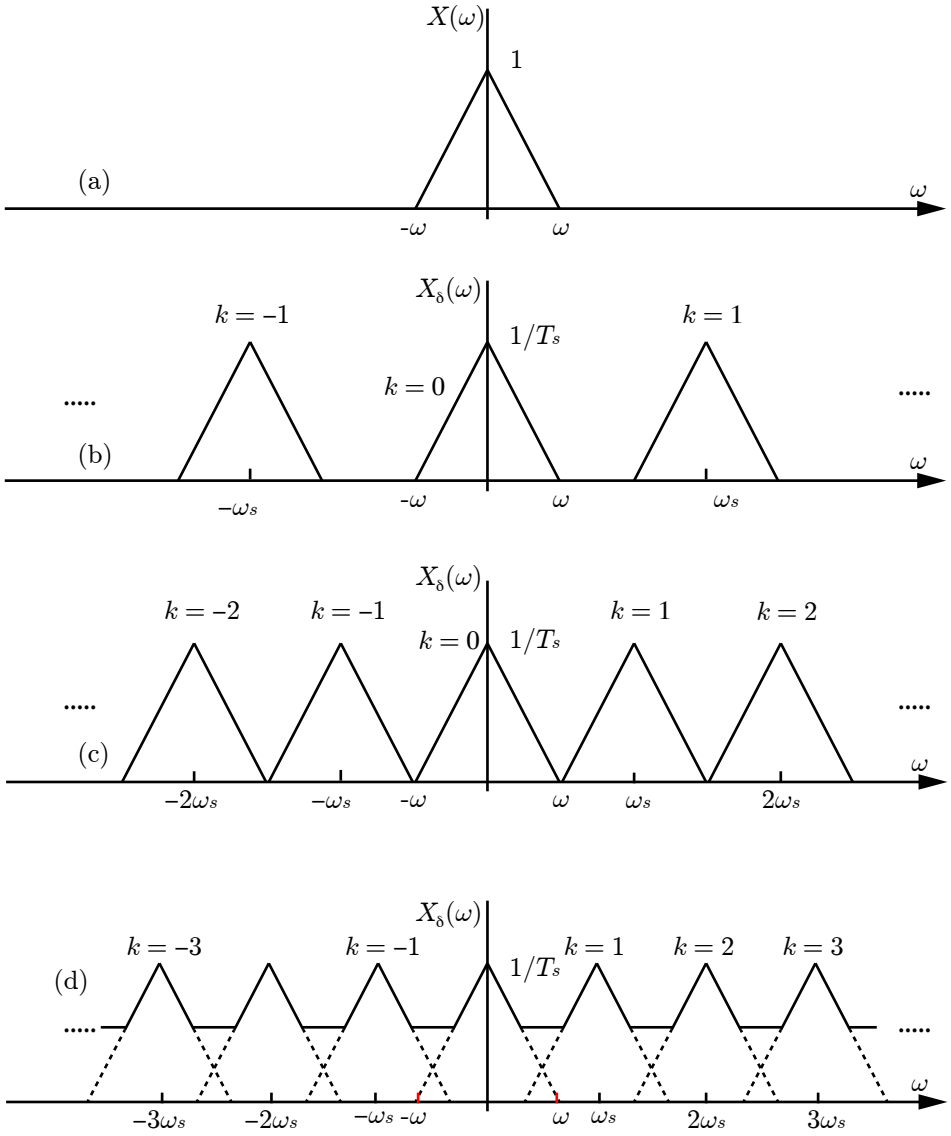


Figure 5.12: Impulse Sampling of a Signal.

the normalized sinc (blue in Figure 5.13), whose definition is $\sin(\pi x)/\pi x$, again with $\text{sinc}(0) = 1$. Notice how the zeros of this function are at integer (positive and negative) values of the x axis.

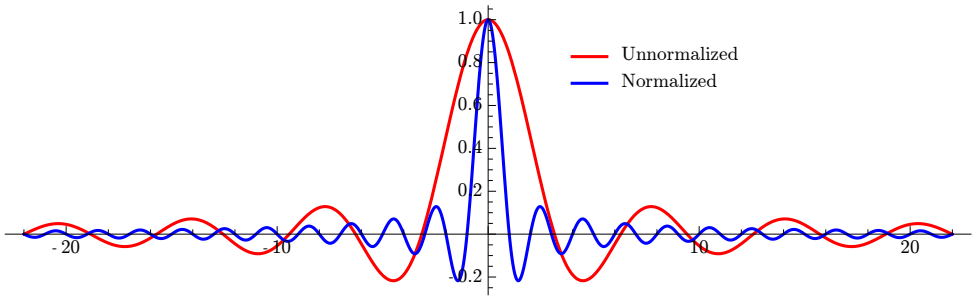


Figure 5.13: The Spreader (Sinc Function).

The principle of reconstructing an analog signal from discrete points by using the spreader is first explained for the one-dimensional case, where we start with an analog wave and digitize it into audio samples that (following [Smith, Alvy Ray 21]) we call soxels, a contraction of sonic element. Figure 5.14a shows two adjacent soxels chosen out of a set of many that came from a sound wave S. The height of a soxel indicates the amplitude of the original, analog wave at a point in time, and the location of the soxel indicates the time it was sampled. The first step in the reconstruction is to place the (normalized) spreader on top of the soxel, with its zeros on the x axis, and scale it vertically such that its central amplitude equals the soxel's amplitude (part (b) of the figure). Notice that the scaling does not change the horizontal dimensions (the frequency) of the spreader. We then do the same for the second soxel, place the spreader on top of it and scale it vertically (part (c) of the figure). Finally, we add the two scaled spreaders. The resulting curve (in green) is a continuous function (or an analog signal) that can in principle be computed at uncountably many points.

However, it is easy to see that the sum of the two spreaders, the green curve of Figure 5.14c, drops very low in the space between the soxels. This is because our soxels are spaced too far apart, as a result of a very low sampling rate. Figure 5.14d shows how moving the soxels closer improves the reconstruction. Notice also how the frequencies of the two spreaders are identical, but out of phase.

The Nyquist rate is achieved in this case when the distance between consecutive soxels is at least half the period of the spreader. In this case, each period of the green sum contains at least two soxels, as the sampling theorem demands. This can be seen in Figure 5.14d by noticing that the distance between the two soxels is exactly half the period of the spreaders. In other words, the soxels must be close enough such that the period of the spreader would equal that of the highest-frequency term of the discrete Fourier transform (DFT), Equation (4.2), of the original sound wave S. Phrased another way, the spreader would be that term, except for amplitude.

The part of the green curve between the soxels is a smooth, reasonable interpolation of the soxels, while the parts outside the soxels would be improved later, when more

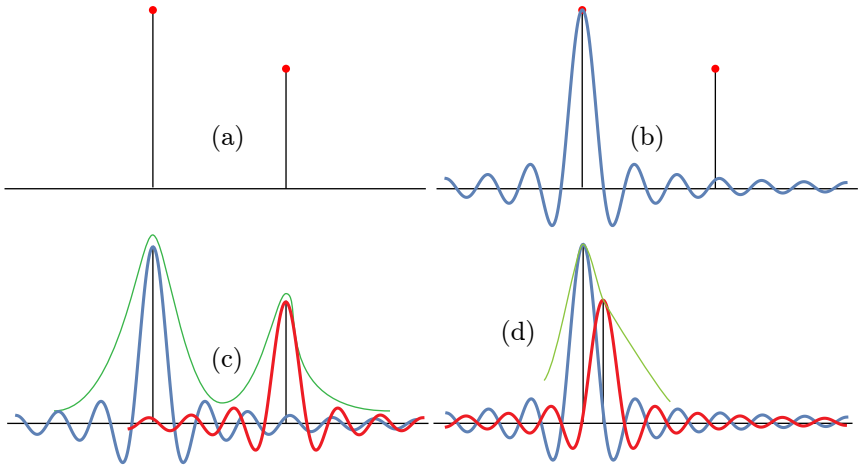


Figure 5.14: Reconstructing Soxels With a Spreader.

spreaders are placed on all the soxels and are vertically scaled and added to the sum of spreaders.

Figure 5.15 shows a pure sine wave and above it, part of an analog signal with some soxels. A manual check will verify that each of the seven periods of the sine wave corresponds to two soxels in the analog signal. Thus, this sine wave has the Nyquist frequency, and is therefore the highest-frequency wave in the discrete Fourier transform of the analog signal.

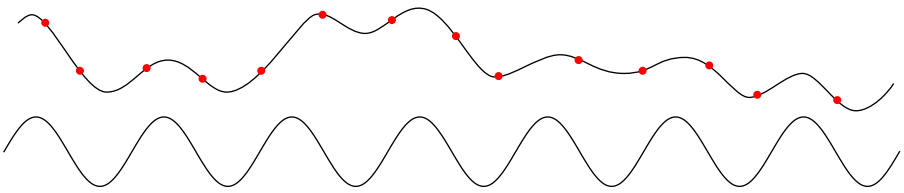


Figure 5.15: Reconstructing With the Nyquist Rate.

In practice, the spreaders are infinitely wide, which results in a vast number of computations for each soxel. Even one second of sound typically produces, when digitized, more than 44,000 soxels, a number that may tax the capabilities of our present digital devices (computers and smart phones). This is why practical reconstructions of one-dimensional time signals often use only the central part of the spreader, shown in Figure 5.16. This so-called cubic spreader includes just the central part and two small negative lobes of the original spreader. It reduces the computation time significantly, while only slightly affecting the quality of the reconstructed wave.

Once the reader has grasped this one-dimensional reconstruction process, its extension to two dimensions, i.e., image pixels, is in principle straightforward but is harder to

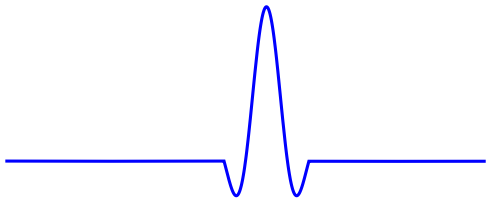
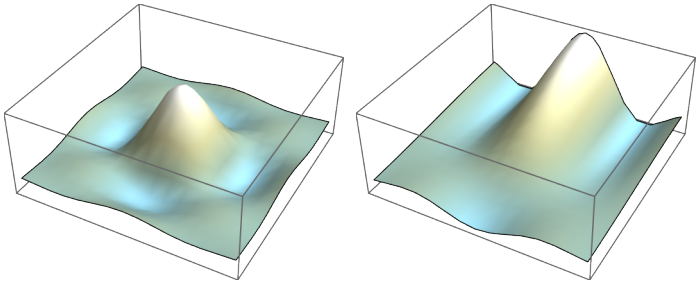


Figure 5.16: Reconstructing With a Truncated Spreader.

visualize and it involves many more computations. An image is generated on the sensor of a camera. Each photosite becomes a pixel, and the pixels are arranged in a matrix, which is the two-dimensional equivalent of a one-dimensional time-signal $x(t)$.

The main differences between the one- and two-dimensional cases are (1) a two-dimensional spreader is required, which is placed on every pixel value and then scaled vertically, and (2) a two-dimensional sum of all the spreaders, which produces a smooth, continuous surface `IMG` that can be visualized as floating above the tops of the pixels. Once the expression of `IMG` is ready, the intensity (grayscale) of the image can be obtained at any mathematical point on `IMG` simply be measuring the height (i.e., value) of `IMG` at the point. Figure 5.17 shows a two-dimensional truncated spreader, together with a cut in its middle, which proves that its profile is the familiar one-dimensional spreader. Such a spreader, which has circular symmetry, can also be referred to as a bicubic sinc function.



```
SetOptions[Plot3D, ImageSize -> 72*2.45,
  ColorFunction -> "LightTerrain", Mesh -> None, Axes -> None];
Plot3D[Sinc[x] Sinc[y], {x, -7, 7}, {y, -7, 7}, PlotRange -> All,
 AspectRatio -> 13/16, ImageSize -> 72*2.45]
Export["/Users/davidsalomon/Desktop/sinc1.pdf", %]
Plot3D[Sinc[x] Sinc[y], {x, -7, 7}, {y, -4, 0}, PlotRange -> All,
 AspectRatio -> 13/16, ImageSize -> 72*2.45]
Export["/Users/davidsalomon/Desktop/sinc2.pdf", %]
```

Figure 5.17: A Three-Dimensional Truncated Spreader.

Most current digital images are in color, with each pixel having three color components. The reconstruction process of such an image starts by collecting the first

component (often red) of all the pixels, to become a new, "red" set of pixels. The above one-dimensional reconstruction process is then applied to this set, with two-dimensional spreaders, to generate a smooth mathematical surface $\texttt{IMG}_r$ that will become the red component of the final image. Similar steps are then taken for the remaining two pixel components, producing surfaces $\texttt{IMG}_g$ and $\texttt{IMG}_b$. To determine the color of a mathematical point on the surface, measure the heights of the three surfaces $\texttt{IMG}_n$ at the point, and use them as the (R, G, B) components of the color of the reconstructed image at the point.

A Summary. This chapter is concerned with the sampling theorem and its effect on our daily lives. This summary attempts to shed more light on this important subject by following the "life" of a typical video. Imagine a video, perhaps a concert or a documentary, performed by musicians, singers, or actors. The video and associated audio are picked up by a camera and its microphone and are digitized. After being edited, the digital data becomes two streams, one of soxels (audio samples) and the other of pixels (triplets of color values).

The streams are then compressed, an error-control code is included for added reliability, and the result is recorded, as a large set of numbers, on a DVD. You buy the DVD, feeling that you have a long, complete video in it, whereas this plastic disc contains many soxels and pixels, edited, compressed, and protected by the error-control code. We can say that the sampling/digitizing process has converted the analog data, which in principle can have infinite capacity, to a finite set of numbers, thereby losing an infinite amount of information.

Eventually, you may decide to watch the DVD, and this is when the sampling theorem performs its magic, converting the finite stream of discrete numbers to analog signals for the audio and video. We can say that the reconstruction process fills up the gap between the discrete pixels with an interpolation that is controlled by the spreader. While this is done, you can already hear and see the performance, but it is fleeting. It disappears as soon as you sense and enjoy it. All that is materially left is the same DVD with the same numbers, ready to reconstruct the content and play it again for your enjoyment.

The sampling theorem is the most important theoretical result in the field of digital signal processing.

—Richard G. Lyons.

6
Convolution

The term convolution, as used in image processing, is the process of applying a filter to an image. Pixels, the basic elements of an image, are discrete, which is why the discrete version of convolution is used with images and is the one discussed here. Many signals are one dimensional, but images are two dimensional, which is why this chapter starts with the one-dimensional discrete convolution, and then extends it to two dimensions.

> Convolution
>
> 1. Something that is very complicated and difficult to understand: a twist or curve.
>
> 2. (In mathematics) a function derived from two given functions by integration that expresses how the shape of one is modified by the other.
>
> 3. An integral that expresses the amount of overlap of one function g as it is shifted over another function f. It therefore "blends" one function with another. (From MathWorld—A Wolfram Web Resource.)
>
> —Dictionary definitions of convolution.

6.1 Example

We start with a simple example. Given an array a of numbers, our task is to smooth the differences between consecutive numbers. This can easily be done by replacing each number with the average of itself and its two immediate neighbors, an operation that can be written as

$$b[i] \leftarrow (a[i-1]+a[i]+a[i+1])/3 = a[i-1]h[-1]+a[i]h[0]+a[i+1]h[1] = \sum_{j=-1}^{1} a[i+j]h[j], \tag{6.1}$$

where index i varies over all the elements of a. Figure 6.1 illustrates two steps of this process. Array h, the convolution kernel (or the filter kernel), contains 1/3 in each of its three elements. The kernel is also often referred to as the system's impulse response.

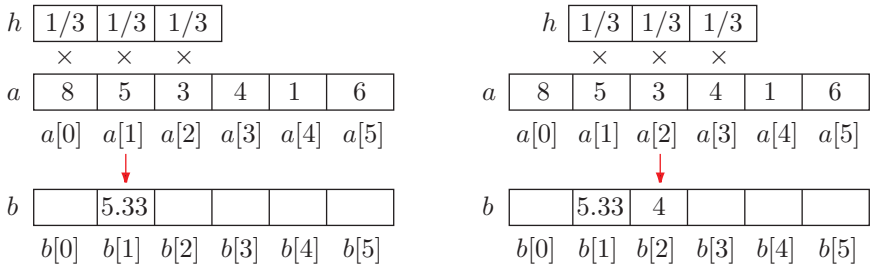


Figure 6.1: Smoothing as a Convolution.

For reasons that will be explained shortly, it is customary to assign descending index values to the convolution kernel h and label its elements $h[1]$, $h[0]$, and $h[-1]$, which changes Equation (6.1) to

$$b[i] \leftarrow a[i-1]h[1] + a[i]h[0] + a[i+1]h[-1] = \sum_{j=i-1}^{i+1} a[j]h[i-j]. \tag{6.2}$$

The star symbol $\star$ is used to indicate convolution, so Equation (6.2) is normally written as

$$b[i] = (a \star h)[i] = \sum_{j} a[j]h[i-j], \tag{6.3}$$

where the summation index j varies over a range that includes all the nonzero elements of the kernel h. In technical jargon we say that the output array (or output signal) b equals the input signal a convolved with the impulse response h.

The ends (or edges) of the signal array a always present a problem and may cause wrong results, because some signal values are missing from the sum of Equation (6.3). In our example, the computation of $b[0]$ requires the value of $a[-1]$ and that of $b[5]$ requires the value of $a[6]$. These values do not exist, which is why practical implementations of convolution employ solutions such as the following:

- Do not compute convolution values for $b[0]$, $b[5]$ or any other cases where input values are missing.
- Assume that any missing values are zeros.
- Copy the nearest value to cover any missing values. In our example, element $a[-1]$ would become 8 (a copy of $a[0]$) and element $a[6]$ would be 6 ($a[5]$).
- Assume that the signal array is periodic. The last element $a[5]$ of our array would be followed by $a[0]$ and the first element $a[0]$ would be preceded by $a[5]$.

Perhaps the best way to understand convolution is to work out a detailed example twice, first from the point of view of the input signal a and then from the viewpoint of the output signal b. The former view illustrates the relation between convolution and signal processing, while the latter view clarifies the mathematical operations of convolution and explains why it may be interpreted as a weighted sum. Our example consists of the input signal $a = (0.5, 0, -0.8, -1.2, -1.2, -2, 0.2, 1, 0)$ and the kernel $h = (0.1, 0.3, 0.5, -1)$.

The input side process illustrates how each input value (or input sample) contributes to the final output. It consists of two steps as follows:

- Multiply each of the nine input values by the four elements of the kernel, creating four numbers, shifted as illustrated in Figure 6.2. Each of the first nine parts of the figure displays the impulse response contributed by one of the elements of the input signal a. These impulse responses are shifted and are also scaled by the kernel h. Notice that the last three steps create numbers with indexes 9, 10, and 11. Thus, the length of the output in our example is 12. In general, the length of the output created by this variant of convolution is one less than the sum of the lengths of the input and the kernel.

- Add the nine results. Notice that each consists of 12 numbers, of which eight are zeros and four are the result of a multiplication in step 1. The final result is also shown graphically in Figure 6.2 bottom-right.

The nine results are

$$\begin{gathered}
(0.05, 0.15, 0.25, -0.5, 0, 0, 0, 0, 0, 0, 0, 0)\\
(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)\\
(0, 0, -0.08, -0.24, -0.4, 0.8, 0, 0, 0, 0, 0, 0)\\
(0, 0, 0, -0.12, -0.36, -0.6, 1.2, 0, 0, 0, 0, 0)\\
(0, 0, 0, 0, -0.12, -0.36, -0.6, 1.2, 0, 0, 0, 0)\\
(0, 0, 0, 0, 0, -0.2, -0.6, -1., 2, 0, 0, 0)\\
(0, 0, 0, 0, 0, 0, 0.02, 0.06, 0.1, -0.2, 0, 0)\\
(0, 0, 0, 0, 0, 0, 0, 0.1, 0.3, 0.5, -1, 0)\\
(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0),
\end{gathered}$$

and their 12-point sum, the final output, is

$$b = (0.05, 0.15, 0.17, -0.86, -0.88, -0.36, 0.02, 0.36, 2.4, 0.3, -1, 0).$$

The two steps involve only multiplications and additions, which are commutative. Those who carefully follow this example will agree that convolution is commutative. Exchanging the input and kernel would result in the same 12-point final output. However, even though the mathematics is simple and is commutative, there is a physical difference between the input and the kernel. The former is given by the physical problem at hand, while the latter is specified by the user, based on his experience and intuition.

Now, for the output-side viewpoint. This process examines the output values $b[]$ to figure out what input samples $a[i]$ contribute to each output value $b[j]$. This knowledge is useful because any practical software for convolution should consist of a loop where

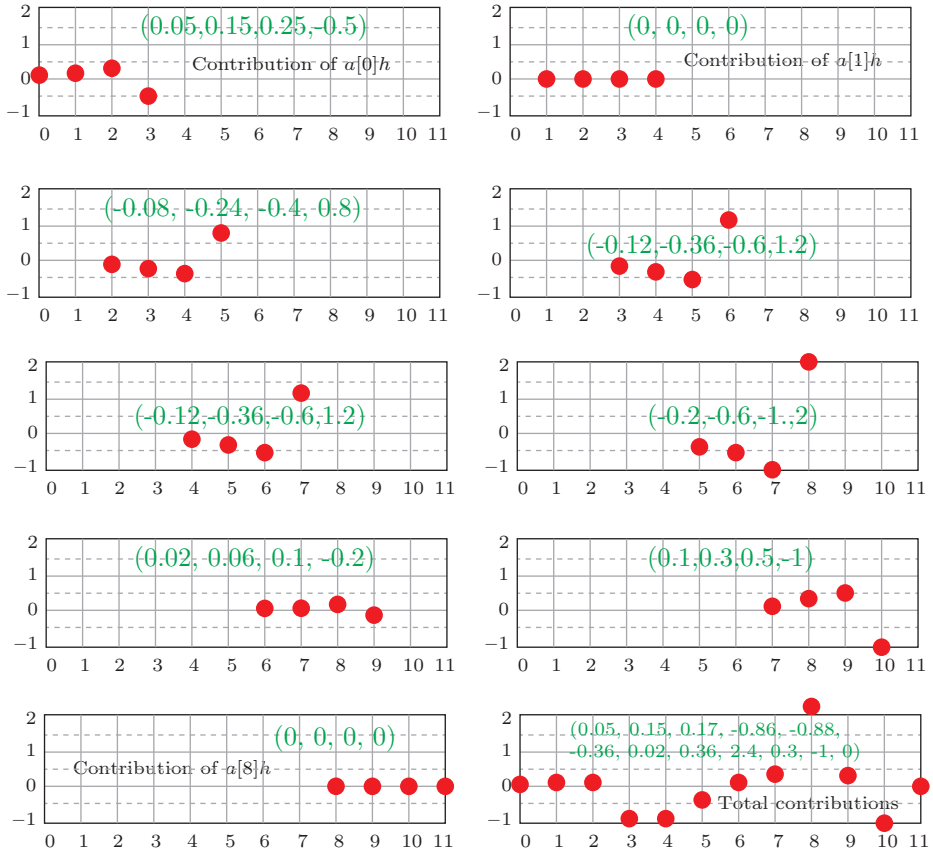


Figure 6.2: Input-Side Convolution Example.

each iteration computes one output value $b[j]$ from several input samples and from all the values of the kernel. Each value $b[j]$ should be computed independently of the other output values. We use the above example to figure out how the sixth output value $b[5] = -0.36 = 0.8 - 0.6 - 0.36 - 0.2$ was obtained. This value is the sum of the columns labeled 5 in Figure 6.2 (only four of these columns are nonzero).

$$\begin{aligned} &a[2]\cdot h[3] + a[3]\cdot h[2] + a[4]\cdot h[1] + a[5]\cdot h[0] \\ &= -0.8\cdot(-1) - 1.2\cdot 0.5 - 1.2\cdot 0.3 - 2\cdot 0.1 \\ &= \sum_{i=2}^{5} a[i]h[5-i]. \end{aligned}$$

From this result we conclude that the general output value $b[j]$ is computed as the

sum

$$b[j] = \sum_{i=j-3}^{j} a[i]h[j-i]. \tag{6.4}$$

(The value 3 in the subscript is one less than the length of the kernel.) The important point here is that each output value $b[j]$ is computed as the dot product of some elements of a with the elements of the kernel, where the latter are taken in reverse order. This explains the unusual choice of index in Equation (6.2).

Figure 6.3 is a graphic representation of this process. An element of the output b is computed as the dot product of part of the input a and a left-to-right flipped version of the kernel h. The flipped kernel (in the dashed box) is then slid to the right and the process is repeated for all the elements of the input a.

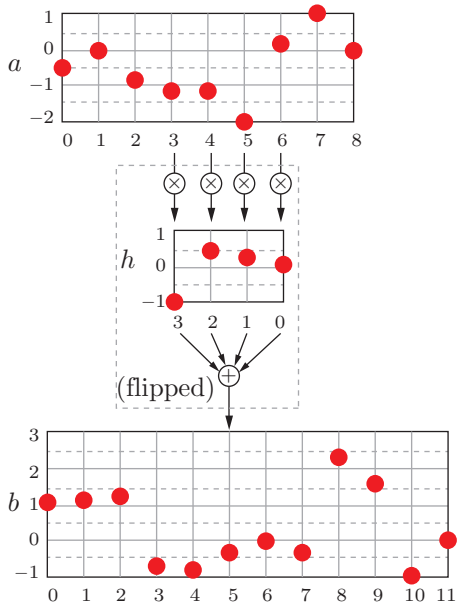


Figure 6.3: Output-Side Convolution As a Dot Product.

Introducing discrete convolution by means of the example above is simple, easy to follow, and it produces Equation (6.4), the main convolution equation, in a natural way. This should be compared to the way discrete convolution is typically introduced in textbooks on image processing. Such a book may simply say "Discrete time convolution is an operation on two discrete time signals defined by the integral...," a definition which does not explain why the elements of the kernel h are used in reverse order.

Edge effects. It is easy to implement convolution from Equation (6.4), but notice that for extreme values of the output index j, the results are wrong. The sum of Equation (6.4) goes from $i = j - 3$ to j, so for $j = 0$, index i should vary from -3 to 0, but input samples $a[-3]$, $a[-2]$, and $a[-1]$ do not exist, so the sum reduces to the

single product $a[0]h[0]$. (The technical term for this is that the impulse response is not fully immersed in the input signal.) Similarly, for $j = 1$, index i should vary from -2 to 1, but the actual result is only $a[0]h[1] + a[1]h[0]$. A similar situation exists for the largest values of j, where some input samples do not exist and the sum can be only partly computed. Several methods to handle edge effects have been discussed earlier.

The mathematical aspect of convolution can now be summarized as follows. Each input sample contributes a scaled and shifted version of the impulse response (the kernel) to the final output. Conversely, each output value is a contribution of several input samples, each multiplied by a value from the flipped impulse response.

While true, this summary does not tell us why convolution is so useful in image processing or in the more general field of digital signal processing (DSP). We can begin to grasp its importance when we again examine Equation (6.4) and Figure 6.3. They both illustrate how each output value is a weighted sum of several input samples, with the elements of the kernel as the weights. The very first example in this section employs a kernel, each of whose three elements equals 1/3. This kernel generates output values each of which is the average of three input samples. Viewing convolution as a weighted sum is the key to understanding its operation and how to use it in practice.

Figure 6.4 illustrates the effects of various kernels on a one-dimensional signal. Blurring, sharpening, and shifting kernels are shown.

6.2 2D Convolution

Image processing is concerned with images, and a digital image is a two-dimensional array of pixels. The method of convolution can be directly extended to such arrays. All that is needed is a two-dimensional kernel. The kernel is flipped twice, vertically and horizontally, and then it is slid over the entire image row by row to create the output image. This can be summarized by the equation

$$b[m,n] = a[m,n] \star h[m,n] = \sum_j \sum_i a[i,j]h[m-i,n-j]. \tag{6.5}$$

As an example, given a 3×3 kernel whose rows and columns are labeled from -1 to 1, this equation results in the sum

$$\begin{aligned} b[1,1] =& \sum_j \sum_i a[i,j]h[1-j,1-i] \\ =& a[0,0]h[1,1] + a[1,0]h[0,1] + a[2,0]h[-1,1] \\ &+ a[0,1]h[1,0] + a[1,1]h[0,0] + a[2,1]h[-1,0] \\ &+ a[0,2]h[1,-1] + a[1,2]h[0,-1] + a[2,2]h[-1,-1]. \end{aligned}$$

Figure 6.5 is an example of five typical filters obtained by a two-dimensional convolution. The original image shows the Wat Phra Kaew, Grand Palace in Bangkok, and the filters were achieved by the following kernels:

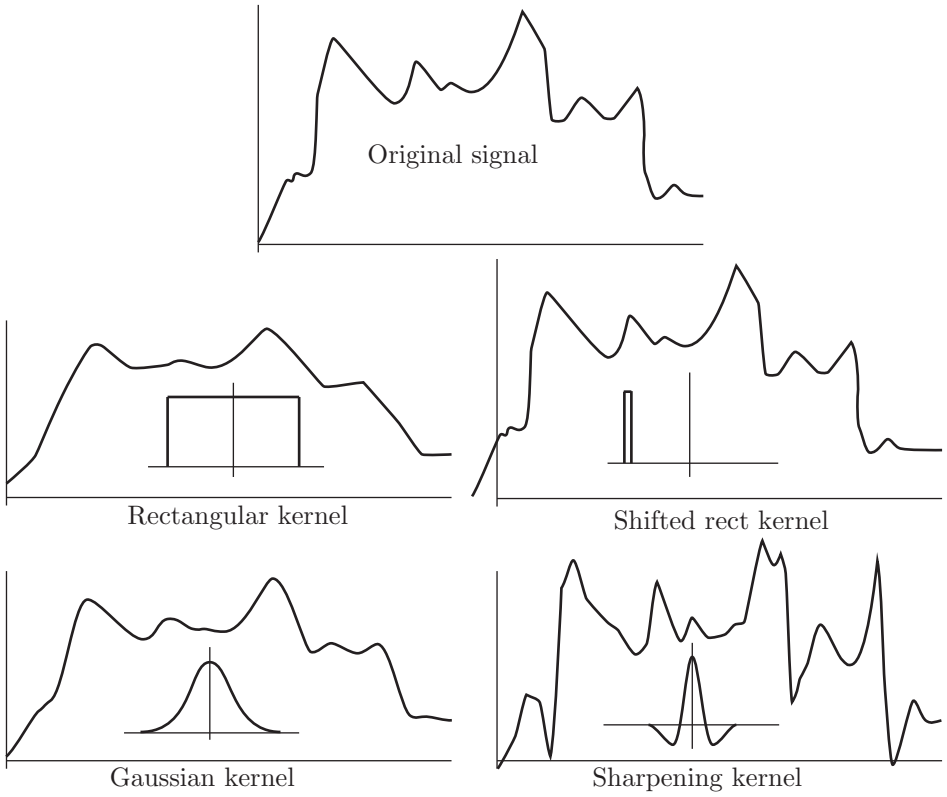


Figure 6.4: Various Convolution Kernels.

	1	1	1	
	1	1	1	
	1	1	1	

Blur

	0	0	0	
	-1	1	0	
	0	0	0	

Edge enhance

	0	1	0	
	1	-4	0	
	0	1	0	

Edge detect

	0	-1	0	
	-1	5	-1	
	0	-1	0	

Sharp

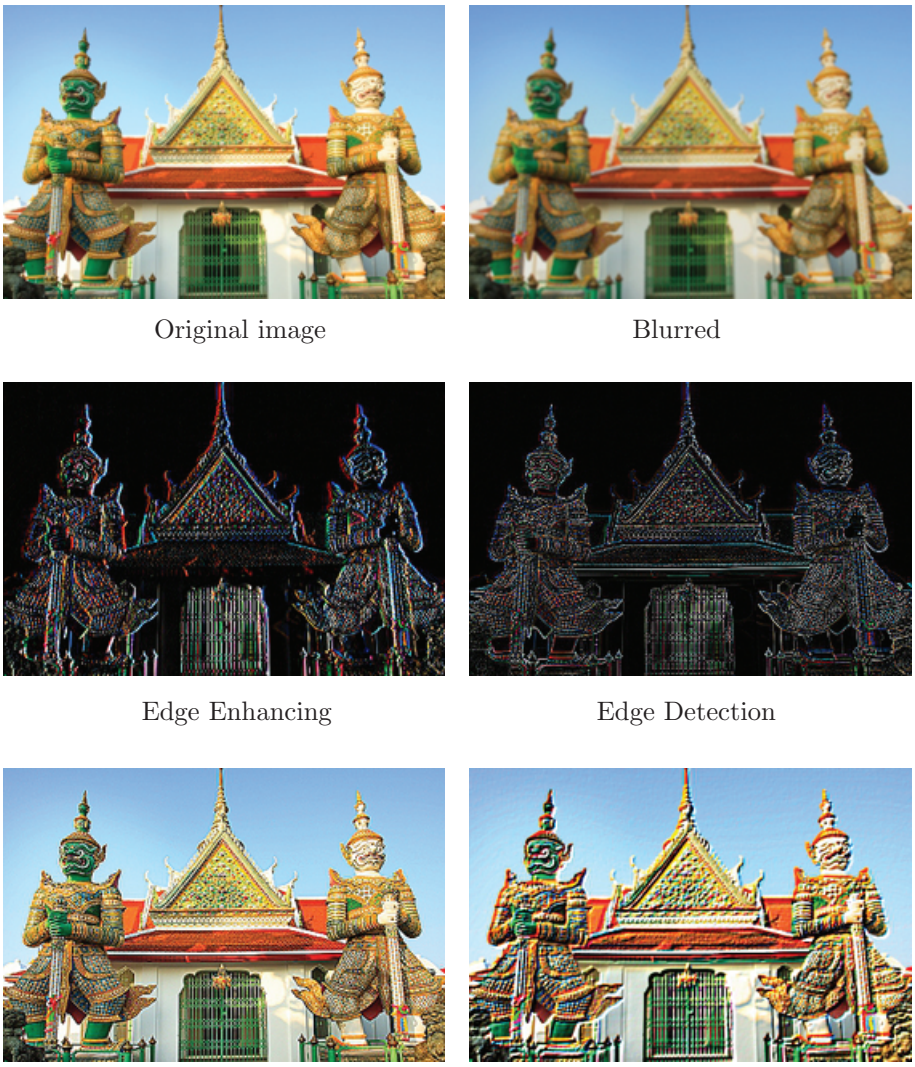
	-2	-1	0	
	-1	1	1	
	0	1	2	

Emboss

The kernel of a two-dimensional convolution does not have to be symmetric. Asymmetric kernels can be used to, for example, blur the image in the horizontal direction and sharpen it in the vertical direction.

The Sobel edge detection operator, an application of convolution

The Sobel operator for edge detection [Sobel 14] is a simple linear filter that has been used since 1968 to detect a wide variety of edges in images. It is based on a 3×3 kernel which features four main directions. An edge will be detected by this kernel if it is vertical, horizontal, or is oriented in one of the two main diagonals of the kernel. When used to identify edges in an image, this kernel is placed over adjacent 3×3 regions



Original image | Blurred

Edge Enhancing | Edge Detection

Sharpened | Embossed

Figure 6.5: Five Filters as 2D Convolutions.

of nine pixels each, and is convolved with each region. The two original Sobel kernels, for horizontal and vertical edges, are

$$H_x^S = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}, \quad \text{and} \quad H_y^S = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}.$$

```
(*Detect edges with Sobel Kernel*)
img = Import["/Users/davidsalomon/Desktop/Pema.tif"]
kernely = {{-1, 0, 1}, {-2, 0, 2}, {-1, 0, 1}};
kernelx = {{1, 2, 1}, {0, 0, 0}, {-1, -2, -1}};
a = ImageConvolve[img, kernelx];
ImageAdjust[a, {0, 6}]
a = ImageConvolve[img, kernely];
ImageAdjust[a, {0, 6}]
```

Figure 6.6: Horizontal and Vertical Edges with the Sobel Operator.

They are illustrated in Figure 6.6 which also lists the Mathematica code.

6.3 The Convolution Theorem

The following is not strictly a proof of the convolution theorem, because it uses the infinite in a loose way, as if it were a number. However, many readers may find the derivation here both readable and illuminating.

We look at the DTFT, the discrete time version of the Fourier transform (Page 58), the one that applies when the time signal input is both discrete and infinite, and show that $x(t)y(t) = 1/(2\pi)[X(\omega) \star Y(\omega)$, where $x(t)$ is a time signal and $X(\omega)$ is its DTFT transform, or its frequency domain. Thus, if

$$X(\omega) = \sum_{n=-\infty}^{\infty} x(n)e^{-i\omega n} \quad \text{and} \quad Y(\omega) = \sum_{n=-\infty}^{\infty} y(n)e^{-i\omega n},$$

then the DTFT of the convolution $x(n) \star y(n)$ equals the product $X(\omega)Y(\omega)$.

$$
\begin{aligned}
\mathrm{DTFT}\left[x(n) \star y(n)\right] &= \sum_{n=-\infty}^{\infty} \left[x(n) \star y(n)\right] e^{-i\omega n} \\
&= \sum_{n=-\infty}^{\infty} \left[\sum_{k=-\infty}^{\infty} x(n)\, y(n-k) \right] e^{-i\omega n} \\
&= \sum_{k=-\infty}^{\infty} x(k) \sum_{n=-\infty}^{\infty} y(n-k)\, e^{-i\omega n} \\
&\quad \text{(substituting } n = m + k\text{)} \\
&= \sum_{k=-\infty}^{\infty} x(k) \sum_{m+k=-\infty}^{\infty} y(m) e^{-i\omega(m+k)} \\
&\quad \text{(naively assuming that } m+k \text{ starting at } -\infty \\
&\qquad \text{is the same as } m \text{ starting at } -\infty - k\text{)} \\
&= \sum_{k=-\infty}^{\infty} x(k) \sum_{m=-\infty}^{\infty} y(m) e^{-i\omega m} e^{-i\omega k} \\
&= \sum_{k=-\infty}^{\infty} x(k) e^{-i\omega k} \sum_{m=-\infty}^{\infty} y(m) e^{-i\omega m} \\
&= X(\omega) \cdot Y(\omega).
\end{aligned}
$$

Again, this is a "hand waving" sort of proof because it implicitly assumes that we can treat the infinite naively and either vary m from $-\infty - k$ or from $-\infty$.

> I am, somehow, less interested in the weight and convolutions of Einstein's brain than in the near certainty that people of equal talent have lived and died in cotton fields and sweatshops.
>
> —Stephen Jay Gould, *The Panda's Thumb, p. 151.*, (1980)

References

> Some books leave us free and some books make us free.
> —Ralph Waldo Emerson.

2D series [2022a] is `tinyurl.com/5n7zdfnz`

2D series [2022b] is `https://tinyurl.com/5ftdfrf4`

Brigham, E. Oran, *The Fast Fourier Transform*, New York: Prentice-Hall, 2002.

DFT demo (2018) is `youtube.com/watch?v=7AddavtPWqM`

fingerprints (2020) is `youtube.com/watch?v=fRjFwTbJfes`

Gibbs (2022) is `https://www.wikiwand.com/en/Gibbs_phenomenon`

Goodman, Joseph W. (2020) *Fourier Transforms Using Mathematica*, SPIE.

Lagrange, J. L. (1877) "Leçons Élémentaires Sur Les Mathématiques, Donées à l'Ecole Normale en 1795," in *Ouvres*, VII, Paris, Gauthier-Villars, 183–287.

leastSQ (2017a) is `youtube.com/watch?v=EnNH3SxyZEI`

leastSQ (2017b) is `youtube.com/watch?v=kcHe0z2RNhM`

leastSQ (2017c) is `youtube.com/watch?v=u8ccubUfhKY`

leastSQ (2017d) is `tinyurl.com/m2nwk9t4`

leastSQ (2017e) is `tinyurl.com/n46pap9w`

librow (2022) is `librow.com/articles/article-10`

Live FT (2014) is `youtube.com/watch?v=aiKrrGR57aI`

Nahin, P. (1995), *The Science of Radio*, Woodbury, NY: American Institute of Physics.

reducible (2020a) is `https://www.youtube.com/watch?v=h7apO7q16V0`

reducible (2020b) is `https://www.youtube.com/watch?v=Ty0JcR6Dvis`

ScuolaTech (2012) is www.youtube.com/watch?v=SZxWVnIlXb0

Session 3 (2015) is youtube.com/watch?v=a7TUIkn3qjY&t=1394s

Smith, Alvy Ray (2021), *A Biography of the Pixel*, The MIT Press: Leonardo series.

Sobel, Irwin (2014) "An Isotropic 3x3 Image Gradient Operator," available from
http://www.researchgate.net/publication
as file 239398674_An_Isotropic_3_3_Image_Gradient_Operator.

wiki FFT (2022) is en.wikipedia.org/wiki/Fast_Fourier_transform

A bibliography is a list of books that you have not read.
—Anonymous

Answers to Exercises

> A good goal is like a strenuous exercise, it makes you stretch.
>
> —Mary Kay Ash.

Intro.1: Figure Ans.1 shows how the inner product of $\sin(8t)$ with $\sin(7t)$ produces 13 red points, all lying on the x axis.

Intro.2: We start with the four roots for $n = 4$. If α is such a root, then $\alpha^4 = 1$, implying that $\alpha^8 = 1^2 = 1$, making α one of the eight roots for $n = 8$. Four more roots are needed, and we guess that $\sqrt{i}$ is one of them. To prove this, we observe that if $\alpha = \sqrt{i}$ then $\alpha^2 = i$ and $\alpha^8 = i^4 = 1$, but what number is $\sqrt{i}$? If we write $\sqrt{i} = a + bi$ and square both sides, we get $i = a^2 + 2abi - b^2$, which implies that $a^2 - b^2 = 0$ and $2ab = 1$. Thus

$$\sqrt{i} = \frac{\sqrt{2}}{2} + i\frac{\sqrt{2}}{2}. \quad \text{The other roots are } \frac{\sqrt{2}}{2} - i\frac{\sqrt{2}}{2}, \ -\frac{\sqrt{2}}{2} + i\frac{\sqrt{2}}{2}, \text{ and } -\frac{\sqrt{2}}{2} - i\frac{\sqrt{2}}{2}.$$

1.1: Here are several reasons.

- Any physical system that oscillates or resonates produces motion (of air molecules or atoms in solids) that is sinusoidal or very close to it.
- The human ear starts analyzing incoming sound by splitting it into its sinusoidal components.
- Sinusoids of the same frequency constitute a set that is closed with respect to addition and subtraction. If we select several sinusoids with various amplitudes and phases but with the same frequency, and add and subtract them in various ways, the result is a sinusoid at the same frequency. Moreover, the sinusoid is the only periodic waveform that has this property.
- The sine and cosine are orthogonal. Their inner product $\int_0^{2\pi} \cos(mt)\sin(nt)dt$ is zero for $m \neq n$. See also Equation (3.2).

```
Clear["Global`*"]
signal[t_] := 0.3 Sin[8t + 60 Degree];
tb = Table[Integrate[signal[t] Sin[8t + p], {t, 0, 2 Pi}],
  {p, 0, 2 Pi, Pi/6}]
tickMarks = Table[i, {i, 0, 2 Pi, Pi/6}];
g1 = ListLinePlot[tb, InterpolationOrder -> 3, DataRange -> {0, 2 Pi},
   AspectRatio -> 1/Pi, Ticks -> {tickMarks, Automatic}];
g2 = ListPlot[tb, PlotStyle -> {Red, PointSize[0.015]},
   DataRange -> {0, 2 Pi}, AspectRatio -> 1/Pi];
Show[g1, g2]

{0.471, 0.8162, 0.942, 0.8162, 0.471, 0., -0.471,
 -0.816, -0.942, -0.816, -0.471, 0., 0.471}
```

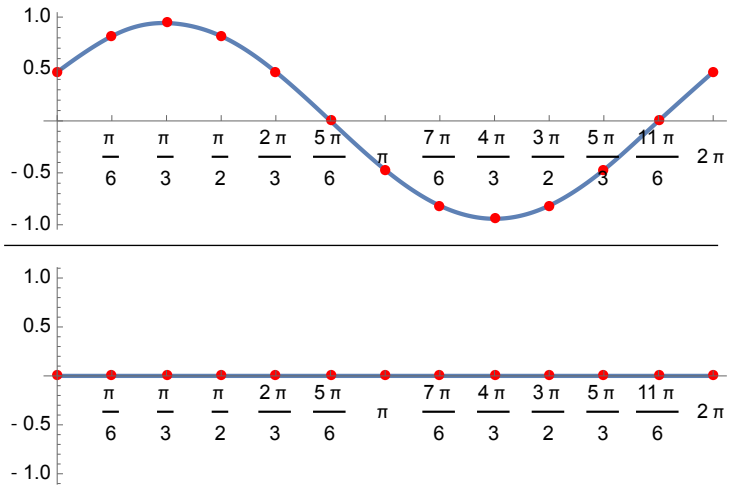


Figure Ans.1: The Inner Product as a Measure of Closeness.

■ The derivative of sine is a cosine and the derivative of a cosine is a negative sine. This associates the two functions with their derivatives in a simple way and simplifies the mathematics of waves. In contrast, the derivatives of a circle are very different from a circle and tend to be complicated functions. Here is why.

The Cartesian equation of a circle is $(x-a)^2+(y-b)^2=r^2$ where r is the radius and (a,b) is the center. The polar equation of a circle of radius a and a center at (r_0,ϕ) is

$$r^2-2r\,r_0\cos(\theta-\phi)+r_0^2=a^2,$$

a complex expression which, when solved for r, becomes the even more elaborate

$$r=r_0\cos(\theta-\phi)\pm\sqrt{a^2-r_0^2\sin^2(\theta-\phi)}.$$

Such expressions are difficult to work with and manipulate mathematically.

On the other hand, the parametric equation of a circle with a center at (a,b) is the

much simpler

$$x = a + r\cos t, \quad y = b + r\sin t,$$

but since it depends on the sine and cosine functions, it is better to use the latter as the basis for representing waves.

For completeness, we mention the alternative parametric representation

$$x = a + r\frac{1-t^2}{1+t^2}, \quad y = b + r\frac{2t}{1+t^2}.$$

But it is obvious that this equation provides no advantages over the basic sine and cosine.

Because of these reasons, we also refer to the sine waves as harmonic. They are simple, pleasing, elegant, in short, perfect.

1.2: An Internet search produces the following information: A person's respiratory rate is the number of breaths you take per minute. The normal respiration rate for an adult at rest is 12 to 20 breaths per minute. The former value translates to $12/60 = 1/5$ breaths per second or 0.2 Hz, and the former is $20/60 = 1/3$ or 0.33 Hz.

3.1: As usual, we start with deriving the three Fourier coefficients

$$\begin{aligned}
a_0 =& \frac{1}{2\pi}\int_{-\pi}^{\pi} f(t)dt = \frac{2}{2\pi}\left[\int_{-\pi}^{0}(-t)dt + 0\right] = \frac{1}{2\pi}\,\frac{t^2}{2}\,\Big|_{-\pi}^{0} = \pi/4,\\
a_n =& \frac{1}{\pi}\int_{-\pi}^{\pi} f(t)\cos(nt)dt = \frac{1}{\pi}\int_{-\pi}^{0}(-t)\cos(nt)dt\\
=& \frac{1}{\pi}\left[\frac{-t}{n}\sin(nt)\,\Big|_{-\pi}^{0} + \frac{1}{n}\int_{-\pi}^{0}\sin(nt)dt\right]\\
=& \frac{1}{n\pi}\left[0 + \frac{-1}{n}\cos(nt)\,\Big|_{-\pi}^{0}\right] \frac{-1}{n^2\pi}\big(\cos(0) - \cos(-nt)\big) = \frac{-1}{n^2\pi}(1-(-1)^n),\\
& \text{where the last pair of parentheses is 0 for even } n \text{ and } -2 \text{ for odd } n.\\
b_n =& \frac{1}{\pi}\int_{-\pi}^{\pi} f(t)\sin(nt)dt = \frac{1}{\pi}\int_{-\pi}^{0}(-t)\sin(nt)dt = \frac{1}{\pi}\left[\frac{t}{n}\cos(nt)\,\Big|_{-\pi}^{0} - \frac{1}{n}\int_{-\pi}^{0}\cos(nt)dt\right]\\
=& \frac{1}{\pi}\left[\frac{t}{n}\cos(nt) - \frac{1}{n^2}\sin(nt)\right] - \pi^0 = \frac{-1}{n\pi}\big(0\cdot\cos(0) - (-\pi)\cos(-nt)\big)\\
=& \frac{-1}{n}\cos(-n\pi) = \begin{cases} \frac{1}{n}, & n \text{ even}\\ -\frac{1}{n}, & n \text{ odd.}\end{cases}
\end{aligned}$$

Once this is done, the final Fourier sum is

$$f(t) = \pi/4 + \sum_{n=1}^{\infty}\frac{-2}{(2n-1)^2\pi}\cos\big((2n-1)t\big) + \sum_{n=1}^{\infty}\frac{(-1)^n}{n}\sin(nt).$$

The Mathematica code

```
Plot[(Pi/4) + Sum[(-2/(n^2 Pi))Cos[n t], {n, {1, 3, 5, 7}}] +
  Sum[((-1)^n/n)Sin[n t], {n, 1, 7}], {t, -5, 5}]
```

generates the wave, although convergence is slow.

3.2: Figure Ans.2 lists the Mathematica code for this series, the original function (a surface) and the first five terms of its Fourier series. A list of the expressions (messy) of each term is also included.

3.3: The period of this signal is $N = 10$ and the derivation is similar to the example in the text.

$$X[n] = \sin\left(\frac{2\pi}{10}n\right) = \frac{1}{2i}\left[e^{2\pi i\frac{n}{10}} - e^{-2\pi i\frac{n}{10}}\right]$$
$$= \frac{1}{2i}e^{2\pi i(1)\frac{n}{10}} - \frac{1}{2i}e^{2\pi i(-1)\frac{n}{10}} = C_1 e^{2\pi i(1)\frac{n}{10}} + C_{-1}e^{2\pi i(-1)\frac{n}{10}}.$$

Thus, $C_1 = 1/2i$ and $C_{-1} = C_9 = -1/2i$. All other coefficients are zero.

4.1: The following identity comes from page 346 of reference [Nahin 95]. It shows how the simple sum $a\cos\theta + b\sin\theta$ can be written in the form $c\cos(\theta + \delta)$ and can therefore also be expressed as a sine. We start by applying one of the trigonometric addition formulas, which gives us the expansion

$$c\cos(\theta + \delta) = c\cos(\theta)\cos(\delta) - c\sin(\theta)\sin(\delta).$$

Equating the coefficients of the above expansion with those of the original sum, we get $a = c\cos\delta$ and $b = -c\sin\delta$. This implies

$$\tan\delta = \frac{\sin\delta}{\cos\delta} \quad \text{and} \quad a^2 + b^2 = c^2(\cos^2\delta + \sin^2\delta) = c^2,$$

which implies

$$\delta = \tan^{-1}\left(-\frac{b}{a}\right) \quad \text{and} \quad c = \pm\sqrt{a^2 + b^2}.$$

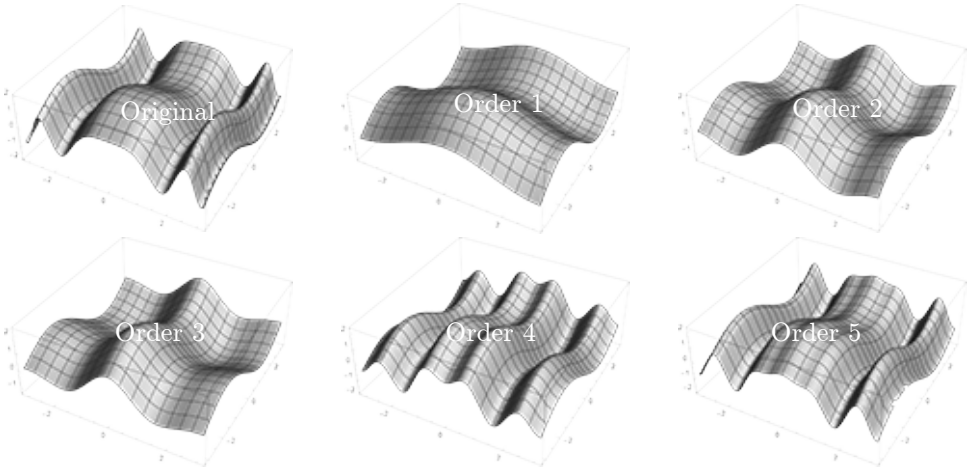
(In order for the original sum to exist, a must be nonzero, but either or both a and b can be negative.)

The final result is

$$a\cos\theta + b\sin\theta = \text{sgn}(a)\sqrt{a^2 + b^2}\cos\left[\theta + \tan^{-1}\left(-\frac{b}{a}\right)\right].$$

Thus, given quantities a, b, and θ, the sum $a\cos\theta + b\sin\theta$ can be written as a phase-shifted cosine, which also equals a phase-shifted sine, but with a different phase.

```
(*2D Fourier series from AneesAbdulMFSslides.pdf*)
Clear["Global`*"]
f[x_, y_] := Cos[x^2] Sin[y];
Plot3D[f[x, y], {x, -Pi, Pi}, {y, -Pi, Pi}, PlotRange -> All]
an[y_] := (1/Pi) Integrate[f[x, y] Cos[n x], {x, -Pi, Pi}];
bn[y_] := (1/Pi) Integrate[f[x, y] Sin[n x], {x, -Pi, Pi}];
Do[p = (1/(2 Pi)) Integrate[f[x, y], {x, -Pi, Pi}]
   + Sum[an[y] Cos[n x] + bn[y] Sin[n x], {n, 1, k}];
 Print[{Plot3D[p, {x, -Pi, Pi}, {y, -Pi, Pi}], p}], {k, 1, 4}]
```



$$\left\{, \frac{\text{FresnelC}\left[\sqrt{2\pi}\right]\text{Sin}[y]}{\sqrt{2\pi}} - \frac{\text{Cos}[x]\left(\text{Cos}\left[\frac{1}{4}\right]\left(\text{FresnelC}\left[\frac{1-2\pi}{\sqrt{2\pi}}\right] - \text{FresnelC}\left[\frac{1+2\pi}{\sqrt{2\pi}}\right]\right) + \left(\text{FresnelS}\left[\frac{1-2\pi}{\sqrt{2\pi}}\right] - \text{FresnelS}\left[\frac{1+2\pi}{\sqrt{2\pi}}\right]\right)\text{Sin}\left[\frac{1}{4}\right]\right)\text{Sin}[y]}{\sqrt{2\pi}}\right\}$$

$$\left\{, \frac{\text{FresnelC}\left[\sqrt{2\pi}\right]\text{Sin}[y]}{\sqrt{2\pi}} - \frac{\text{Cos}[x]\left(\text{Cos}\left[\frac{1}{4}\right]\left(\text{FresnelC}\left[\frac{1-2\pi}{\sqrt{2\pi}}\right] - \text{FresnelC}\left[\frac{1+2\pi}{\sqrt{2\pi}}\right]\right) + \left(\text{FresnelS}\left[\frac{1-2\pi}{\sqrt{2\pi}}\right] - \text{FresnelS}\left[\frac{1+2\pi}{\sqrt{2\pi}}\right]\right)\text{Sin}\left[\frac{1}{4}\right]\right)\text{Sin}[y]}{\sqrt{2\pi}} + \frac{\text{Cos}[2x]\left(\text{Cos}[1]\left(\text{FresnelC}\left[\sqrt{\frac{2}{\pi}}(1+\pi)\right] - \text{FresnelC}\left[\sqrt{\frac{2}{\pi}} - \sqrt{2\pi}\right]\right) + \left(\text{FresnelS}\left[\sqrt{\frac{2}{\pi}}(1+\pi)\right] - \text{FresnelS}\left[\sqrt{\frac{2}{\pi}} - \sqrt{2\pi}\right]\right)\text{Sin}[1]\right)\text{Sin}[y]}{\sqrt{2\pi}}\right\}$$

$$\left\{, \frac{\text{FresnelC}\left[\sqrt{2\pi}\right]\text{Sin}[y]}{\sqrt{2\pi}} - \frac{\text{Cos}[x]\left(\text{Cos}\left[\frac{1}{4}\right]\left(\text{FresnelC}\left[\frac{1-2\pi}{\sqrt{2\pi}}\right] - \text{FresnelC}\left[\frac{1+2\pi}{\sqrt{2\pi}}\right]\right) + \left(\text{FresnelS}\left[\frac{1-2\pi}{\sqrt{2\pi}}\right] - \text{FresnelS}\left[\frac{1+2\pi}{\sqrt{2\pi}}\right]\right)\text{Sin}\left[\frac{1}{4}\right]\right)\text{Sin}[y]}{\sqrt{2\pi}} + \frac{\text{Cos}[2x]\left(\text{Cos}[1]\left(\text{FresnelC}\left[\sqrt{\frac{2}{\pi}}(1+\pi)\right] - \text{FresnelC}\left[\sqrt{\frac{2}{\pi}} - \sqrt{2\pi}\right]\right) + \left(\text{FresnelS}\left[\sqrt{\frac{2}{\pi}}(1+\pi)\right] - \text{FresnelS}\left[\sqrt{\frac{2}{\pi}} - \sqrt{2\pi}\right]\right)\text{Sin}[1]\right)\text{Sin}[y]}{\sqrt{2\pi}} - \frac{\text{Cos}[3x]\left(\text{Cos}\left[\frac{9}{4}\right]\left(\text{FresnelC}\left[\frac{3-2\pi}{\sqrt{2\pi}}\right] - \text{FresnelC}\left[\frac{3+2\pi}{\sqrt{2\pi}}\right]\right) + \left(\text{FresnelS}\left[\frac{3-2\pi}{\sqrt{2\pi}}\right] - \text{FresnelS}\left[\frac{3+2\pi}{\sqrt{2\pi}}\right]\right)\text{Sin}\left[\frac{9}{4}\right]\right)\text{Sin}[y]}{\sqrt{2\pi}}\right\}$$

$$\left\{, \frac{\text{FresnelC}\left[\sqrt{2\pi}\right]\text{Sin}[y]}{\sqrt{2\pi}} - \frac{\text{Cos}[x]\left(\text{Cos}\left[\frac{1}{4}\right]\left(\text{FresnelC}\left[\frac{1-2\pi}{\sqrt{2\pi}}\right] - \text{FresnelC}\left[\frac{1+2\pi}{\sqrt{2\pi}}\right]\right) + \left(\text{FresnelS}\left[\frac{1-2\pi}{\sqrt{2\pi}}\right] - \text{FresnelS}\left[\frac{1+2\pi}{\sqrt{2\pi}}\right]\right)\text{Sin}\left[\frac{1}{4}\right]\right)\text{Sin}[y]}{\sqrt{2\pi}} + \frac{\text{Cos}[2x]\left(\text{Cos}[1]\left(\text{FresnelC}\left[\sqrt{\frac{2}{\pi}}(1+\pi)\right] - \text{FresnelC}\left[\sqrt{\frac{2}{\pi}} - \sqrt{2\pi}\right]\right) + \left(\text{FresnelS}\left[\sqrt{\frac{2}{\pi}}(1+\pi)\right] - \text{FresnelS}\left[\sqrt{\frac{2}{\pi}} - \sqrt{2\pi}\right]\right)\text{Sin}[1]\right)\text{Sin}[y]}{\sqrt{2\pi}} - \frac{\text{Cos}[3x]\left(\text{Cos}\left[\frac{9}{4}\right]\left(\text{FresnelC}\left[\frac{3-2\pi}{\sqrt{2\pi}}\right] - \text{FresnelC}\left[\frac{3+2\pi}{\sqrt{2\pi}}\right]\right) + \left(\text{FresnelS}\left[\frac{3-2\pi}{\sqrt{2\pi}}\right] - \text{FresnelS}\left[\frac{3+2\pi}{\sqrt{2\pi}}\right]\right)\text{Sin}\left[\frac{9}{4}\right]\right)\text{Sin}[y]}{\sqrt{2\pi}} + \frac{\text{Cos}[4x]\left(\text{Cos}[4]\left(\text{FresnelC}\left[\sqrt{\frac{2}{\pi}}(2+\pi)\right] - \text{FresnelC}\left[2\sqrt{\frac{2}{\pi}} - \sqrt{2\pi}\right]\right) + \left(\text{FresnelS}\left[\sqrt{\frac{2}{\pi}}(2+\pi)\right] - \text{FresnelS}\left[2\sqrt{\frac{2}{\pi}} - \sqrt{2\pi}\right]\right)\text{Sin}[4]\right)\text{Sin}[y]}{\sqrt{2\pi}}\right\}$$

Figure Ans.2: A 2D Fourier Series of $\cos(x^2) \times \sin(y)$.

4.2: When the number m of data samples (and also the matrix size) is even, the top row of the matrix is still full of 1's and produces the sum of the data samples (the DC component). An odd number of rows, from row 2 to row m, remain. Each pair of rows k and $m+2-k$ produces, when multiplied by the data array, a complex number and its conjugate. Row $m+2-\lceil m/2\rceil$, which is midway between rows 2 and m, contains only $+1$'s and -1's, so when multiplied by the data samples, it produces a frequency coefficient which is a real number and so equals its conjugate.

4.3: The pixels on both sides of a sharp edge feature very different colors and therefore resemble tiny details.

4.4: Figure Ans.3 illustrates the principle with a simple example. It is obvious that the same swaps would bring the four quadrants back to their original locations.

4.5: Figure Ans.4 lists the commands. They are based on the (obsolete but correct) information at `tinyurl.com/92u67mbm`. The original and final matrices are also displayed graphically, with arrows indicating the movements of several cells.

4.6: Figure Ans.5 shows the simple function `kr` and some of its outputs.

4.7: Referring to Equation (4.2) and assuming that the complex exponentials have been computed in advance and are stored in a table, the sum from 0 to $N-1$ in this equation requires $N-1$ complex additions and N complex multiplications for each frequency coefficient X_k.

A complex addition requires two real additions, so the contribution of $(N-1)$ complex additions to the sum is $2(N-1)$. A complex multiplication requires one real addition, one real subtraction, and four real multiplications, a total of six real arithmetic operations. Its contribution to the sum is therefore $6N$ real operations.

The total number of real operations for the N frequency coefficients X_k is therefore $N[2(N-1)+6N]=8N^2-2N$.

4.8: The details are listed here.

$$\begin{aligned}X[4]&=\left[\left(x_0+x_4e^{-i\pi 4}\right)+e^{-i\frac{\pi}{2}4}\left(x_2+x_6e^{-i\pi 4}\right)\right]\\&\quad+e^{-i\frac{\pi}{4}4}\left[\left(x_1+x_5e^{-i\pi 4}\right)+e^{-i\frac{\pi}{2}4}\left(x_3+x_7e^{-i\pi 4}\right)\right]\\&=[(1+(-3))+1\,(5+4)]\\&\quad-1\,[(-2+8)+1\,(7+6)]=-12.\end{aligned}$$

$$\begin{aligned}X[5]&=\left[\left(x_0+x_4e^{-i\pi 5}\right)+e^{-i\frac{\pi}{2}5}\left(x_2+x_6e^{-i\pi 5}\right)\right]\\&\quad+e^{-i\frac{\pi}{4}5}\left[\left(x_1+x_5e^{-i\pi 5}\right)+e^{-i\frac{\pi}{2}5}\left(x_3+x_7e^{-i\pi 5}\right)\right]\\&=[(1-(-3))+(-i)\,(5-4)]\\&\quad(-0.7+0.7i)\,[(-2-8)+(-i)\,(7-6)]=11.7781-7.3639i.\end{aligned}$$


```
(*Shifting a DFT matrix by swapping quadrants 1<->4, 2<->3*)

ClearAll[matShiftQuad, mm, i, j, c1, c2, r1, r2];
matShiftQuad[mm_] := Module[{dim, c, r, i, j},
  dim = Dimensions[mm];
  b = Table[0, {i, dim[[1]]}, {j, dim[[2]]}];
  i = Floor[dim[[1]]/2];
  If[2 i < dim[[1]], r1 = i + 1; r2 = i, r1 = r2 = i];
  j = Floor[dim[[2]]/2];
  If[2 j < dim[[2]], c1 = j + 1; c2 = j, c1 = c2 = j];
  b[[dim[[1]] - r1 + 1 ;; dim[[1]], dim[[2]] - c1 + 1 ;; dim[[2]]]] =
   mm[[1 ;; r1, 1 ;; c1]];
  b[[dim[[1]] - r1 + 1 ;; dim[[1]], 1 ;; c2]] =
   mm[[1 ;; r1, c1 + 1 ;; dim[[2]]]];
  b[[1 ;; r2, c2 + 1 ;; dim[[2]]]] = mm[[r1 + 1 ;; dim[[1]], 1 ;; c1]];
  b[[1 ;; r2, 1 ;; c2]] =
   mm[[r1 + 1 ;; dim[[1]], c1 + 1 ;; dim[[2]]]];
]

(*Test*)
a = Table[RandomInteger[{0, 99}], {i, 3}, {j, 4}];
a // MatrixForm
matShiftQuad[a]
b // MatrixForm
```

Figure Ans.3: Shifting By Moving Quadrants.

$$
\begin{aligned}
X[6] = &\left[\left(x_0 + x_4 e^{-i\pi 6}\right) + e^{-i\frac{\pi}{2}6}\left(x_2 + x_6 e^{-i\pi 6}\right)\right] \\
&+ e^{-i\frac{\pi}{4}6}\left[\left(x_1 + x_5 e^{-i\pi 6}\right) + e^{-i\frac{\pi}{2}6}\left(x_3 + x_7 e^{-i\pi 6}\right)\right] \\
&= \left[(1-3) + (-1)(5+4)\right] \\
&\quad i\left[(-2+8) + (-1)(7+6)\right] = -11 - 7i.
\end{aligned}
$$

```
ml={{1961.00,-388.23,-0.26,118.49,118.49,-0.26,-388.23},
{-56.89,-119.26,70.29,7.54,-196.00,-91.88,-4.72},
{-96.32,47.30,48.45,74.98,-43.60,-85.47,-35.35},
{-123.80,-29.10,37.81,133.31,-92.31,39.70,98.81},
{-123.80,98.81,39.70,-92.31,133.31,37.81,-29.10},
{-96.32,-35.35,-85.47,-43.60,74.98,48.45,47.30},
{-56.89,-4.72,-91.88,-196.00,7.54,70.29,-119.26}};
MatrixPlot[ml]
dim=Dimensions[ml]
If[OddQ[dim[[2]]],c=Ceiling[dim[[2]]/2],c=dim[[2]]/2]
If[OddQ[dim[[1]]],r=Ceiling[dim[[1]]/2],r=dim[[1]]/2]
ml=RotateLeft[ml,{0,c}];
ml=RotateLeft[ml,r];
MatrixPlot[ml]
```

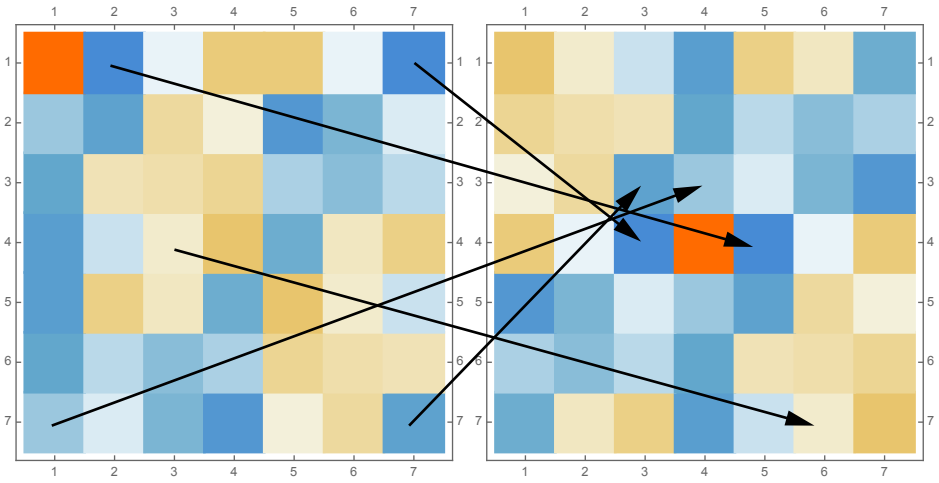
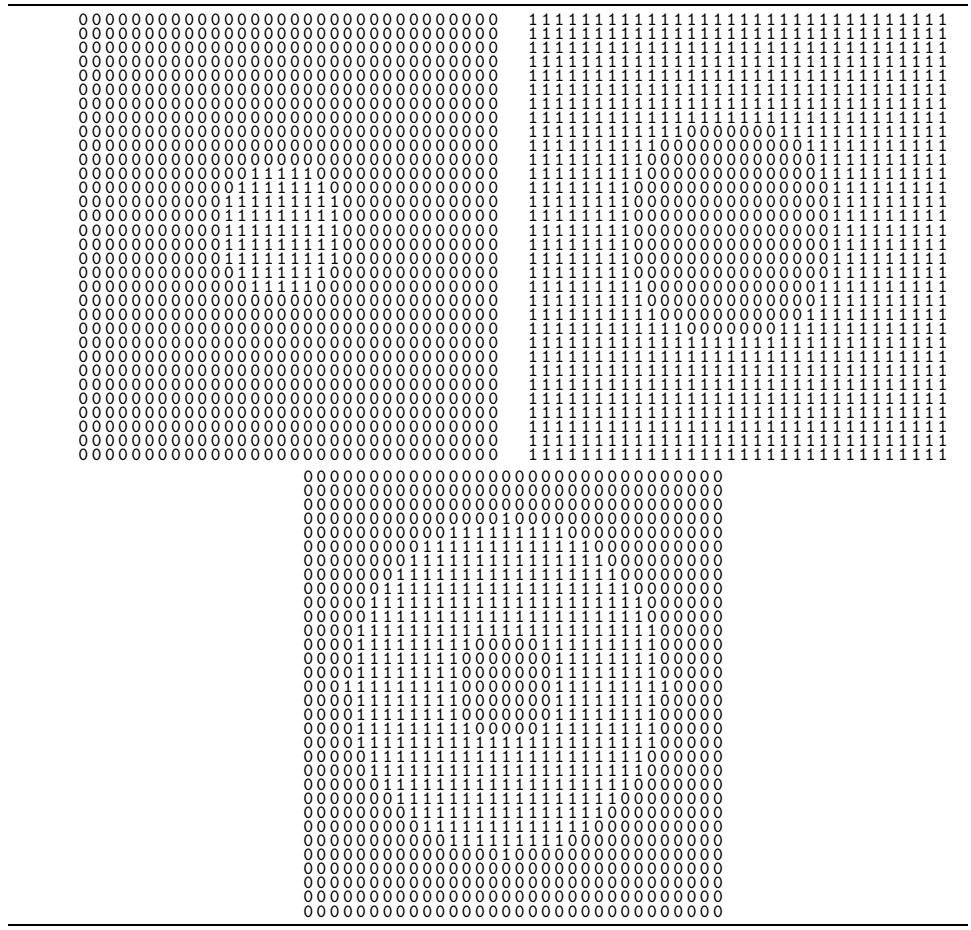


Figure Ans.4: Shifting Rows and Columns in 7×7 Image.

$$\begin{aligned} X[7] &= \left[\left(x_0 + x_4 e^{-i\pi 7}\right) + e^{-i\frac{\pi}{2}7}\left(x_2 + x_6 e^{-i\pi 7}\right)\right] \\ &\quad + e^{-i\frac{\pi}{4}7}\left[\left(x_1 + x_5 e^{-i\pi 7}\right) + e^{-i\frac{\pi}{2}7}\left(x_3 + x_7 e^{-i\pi 7}\right)\right] \\ &= \left[(1-(-3)) + i\,(5-4)\right] \\ &\quad (0.7+0.7i)\left[(-2-8) + i\,(7-6)\right] = -3.7781 - 5.3639. \end{aligned}$$

4.9: A root of the degree-n polynomial $P_n(x)$ is a value x_r such that $P_n(x_r) = 0$. A polynomial $P_n(x)$ can have at most n distinct roots (unless it is the zero polynomial). Suppose that there is another polynomial $Q_n(x)$ that passes through the same $n+1$ data points. At the points, we would have $P_n(x_i) = Q_n(x_i) = y_i$ or $(P_n - Q_n)(x_i) = 0$. The difference $(P_n - Q_n)$ is a polynomial whose degree must be $\leq n$, so it cannot have more than n distinct roots. On the other hand, this difference is 0 at the $n+1$ data points, so it has $n+1$ roots. We conclude that it must be the zero polynomial, which implies that $P_n(x)$ and $Q_n(x)$ are identical.



```
n = 32;
kr[w_] := Table[UnitBox[1 - Sqrt[x^2 + y^2]/w],
  {x, -n/2 + 1, n/2}, {y, -n/2 + 1, n/2}];
 (* or use [UnitBox[Sqrt[x^2 + y^2]/w] *)
kr[8]
```

Figure Ans.5: Using the `UnitBox` Command to Prepare Circular Fourier Filters.

This uniqueness theorem can also be employed to show that the Lagrange weights $L_i^n(x)$ are barycentric; their sum is always 1. Given a function $f(x)$, select $n+1$ distinct values x_0 through x_n, and consider the $n+1$ *support* points $\big(x_0, f(x_0)\big)$ through $\big(x_n, f(x_n)\big)$. The uniqueness theorem states that there is a unique polynomial $p(x)$ of degree n or less that passes through the points, i.e., $p(x_k) = f(x_k)$ for $k = 0, 1, \ldots, n$. We say that this polynomial interpolates the points. Now consider the constant function

$f(x) \equiv 1$. The Lagrange polynomial that interpolates its points is

$$\mathrm{LP}(x) = \sum_{i=0}^{n} y_i L_i^n(x) = \sum_{i=0}^{n} 1 \times L_i^n(x) = \sum_{i=0}^{n} L_i^n(x).$$

On the other hand, $\mathrm{LP}(x)$ must be identical to 1, because $\mathrm{LP}(x_k) = f(x_k)$ and $f(x_k) = 1$ for any point x_k. Thus, we conclude that $\sum_{i=0}^{n} L_i^n(x) = 1$ for any x.

4.10: This is straightforward

$$\begin{aligned} P(x) &= (9x^6 - 6x^4 + 4x^2 + 1) + (8x^5 + x^3 - 5x) \\ &= (9x^6 - 6x^4 + 4x^2 + 1) + x(8x^4 + x^2 - 5) \\ &= P_e(x) + xP_o(x). \end{aligned}$$

The only difference is that when we write $P_e(x)$ and $P_o(x)$ as functions of x^2, they no longer have identical degrees. Thus $P_e(x^2) = 9x^3 - 6x^2 + 4x + 1$ and $P_o(x^2) = 8x^2 + x - 5$.

5.1: The top of Figure Ans.6 shows two sets of parallel horizontal lines that form a low-frequency Moiré pattern after one of them has been rotated 5°. The bottom part of the figure shows several positions of a wheel that rotates clockwise. If we look only at positions 1, 4, and 7 (in red), the wheel seems to slowly rotate counter-clockwise.

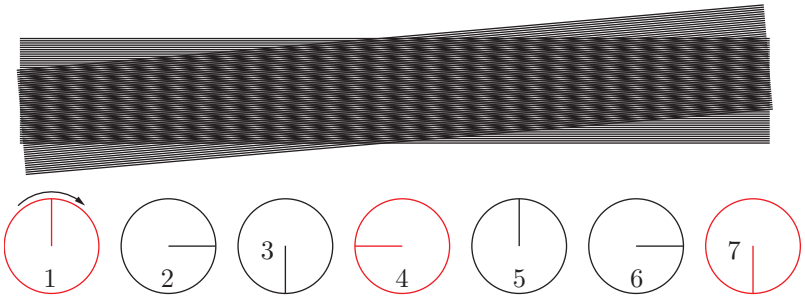


Figure Ans.6: Temporal Aliasing.

An answer is only as good as the question it addresses.
—Anonymous.

Index

I tried to make the index items as complete as possible, including middle names and dates. Any errors and omissions brought to my attention are welcome. They will be added to an errata list and will be included in any future editions, if any.

An index is a map of the world. It is a way of understanding the world. And it is a way of communicating the world to others.

—Michael Wilkerson, *The Index Revolution, (2016)*.