

In presenting this dissertation in partial fulfillment of the requirements for an advanced degree at Idaho State University, I agree that the Library may make it freely available for inspection. I further state that permission for extensive copying of this dissertation for scholarly purposes may be granted by the Dean of the Graduate School or by the dean of my academic division. It is understood that any copying or publication of this dissertation for financial gain shall not be allowed without my written permission.

Signature Carol J. V. Fisher

Date 4/1/94

**IDENTIFYING HIDDEN PERIODICITIES
IN DISCRETE-DOMAIN DATA**

by
Carol J.V. Fisher

A dissertation submitted in partial fulfillment of the
requirements for the degree of

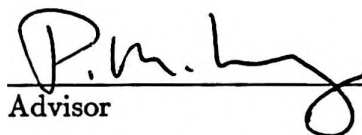
**DOCTOR OF ARTS
IN
MATHEMATICS**

**IDAHO STATE UNIVERSITY
1994**

Copyright © Carol J.V. Fisher 1994
All Rights Reserved

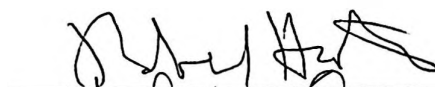
To the Graduate Faculty:

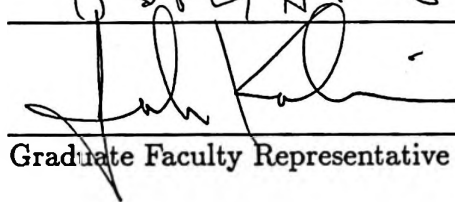
The members of the committee appointed to examine the dissertation of
CAROL J.V. FISHER find it satisfactory and recommend that it be accepted.



Advisor







Graduate Faculty Representative

ABSTRACT

This dissertation investigates the problem of identifying hidden periodicities in discrete-domain data, with emphasis on identification for the purpose of *prediction*. By definition, *discrete-domain data* is a collection of ordered pairs, $\{(t_i, y_i)\}_{i=1}^N$ or ∞ , with the property that the time values t_i can be arranged in strictly increasing order. The data values y_i are allowed to come from the set of real numbers, denoted by \mathbf{R} . Implementation of all the techniques discussed herein are incorporated throughout the dissertation, using the MATLAB software package.

The dissertation is written as the basis for a textbook, and assumes a mathematical background typical of an undergraduate degree in engineering: three semesters of calculus, introductory courses in linear algebra and statistics, and a moderate amount of mathematical maturity. Additional information that is essential for an understanding of the material is included in the appendices.

Periodicity is studied in Chapter 1. The usual definition of periodic functions (as functions from \mathbf{R} to \mathbf{R}) is generalized, to provide a viewpoint that favors investigation of periodicities in discrete-domain data. It is verified that functions obeying this generalized definition satisfy the properties commonly associated with periodicity. The set of *all* periods of a periodic function is studied. A reshaping method for identifying relatively prime periodic components is presented. Important logical considerations regarding the use of identified periodic components for prediction are discussed. The chapter closes with an overview of historical contributions in the search for hidden periodicities.

Chapter 2 develops techniques for fitting a data set with a function. A ‘turning point’ test for random behavior is developed. If the hypothesis of random behavior cannot be rejected, one may still be able to take advantage of the turning points in economics data, via an interesting application of a Martingale Algorithm. Linear and nonlinear least squares approximation techniques are presented for situations where there are specific conjectured components in the data. Condition numbers and discrete orthogonal functions are discussed in the context of overcoming numerical difficulties with computer applications. Gradient methods and a genetic algorithm provide a way to deal with nonlinear least squares approximation. Cubic spline interpolation is presented as a way to achieve a uniform time list, if necessary. Discrete Fourier theory and the periodogram are presented as useful tools, particularly when the data is thought to have unknown periodic components. Efficient computation of the periodogram via the discrete Fourier transform is discussed.

Chapter 3 discusses nonrecursive mathematical filters, and their corresponding transfer functions. Such filters can be used for removal of noise, and are useful identification tools when the data contains sinusoidal components. An overall approach to identifying hidden periodicities, together with examples, is given to conclude the dissertation.

ACKNOWLEDGMENTS

For thirteen years, my husband Bob has supported my efforts in Mathematics. His dedication to and respect for the subject have been an inspiration to me. My daughter Julia has had to put up with a 'distracted' Mom on many occasions. Thanks, Julia and Bob, for your patience and love.

My father, James Vreeland, aroused my interest in hidden periodicities. I continually strive to emulate his extremely thorough and organized approach to analysis. My mother, Pauline Vreeland, has graciously put up with many hours of 'technical talk'. Thanks, Mom and Dad, for a lifetime of love and support.

Many thanks go to my advisor, Professor Pat Lang, for his time and guidance throughout my graduate work at ISU. The exposition and results herein were greatly improved due to his careful scrutiny and constructive comments. Thanks go also to Ken Bosworth, Alan Egger, Robert Huotari, and John Kalivas, for serving on my graduate committee.

TABLE OF CONTENTS

ABSTRACT	iv
ACKNOWLEDGMENTS	v
TABLE OF SYMBOLS	viii
FLOWCHART	ix
CHAPTER 1. PERIODIC FUNCTIONS	
1.1 The Purpose of this Dissertation	1
1.2 Notation	10
1.3 Periodicity:	
Introduction to Periodic Functions	26
Some Basic Reshaping Techniques	33
1.4 More on Periodic Functions	41
1.5 Using Identified Periodic Components for Prediction	59
1.6 Sinusoids	72
1.7 Historical Contributions in the Search for Hidden Periodicities	94
CHAPTER 2. 'FITTING' A DATA SET WITH A FUNCTION	
2.1 Random Behavior:	
The Turning Point Test	108
Economics Application: Taking Advantage of Turning Points	124
2.2 Tests for Specific Conjectured Components:	
Linear Least-Squares Approximation	142
2.3 Computer Application Considerations for Linear Least-Squares:	
Condition Numbers	155
Discrete Orthogonal Functions	164
2.4 Nonlinear Least-Squares Approximation:	
A Linearizing Technique and Gradient Methods	173
A Genetic Algorithm	186
2.5 Cubic Spline Interpolation	203
2.6 Discrete Fourier Series and the Periodogram	215
2.7 The Periodogram, via the Discrete Fourier Transform	233
CHAPTER 3. FILTER THEORY	
3.1 Mathematical Filters	244
3.2 Transfer Functions	252
3.3 Designing and Improving Filters	267
3.4 Identifying Hidden Periodicities: Approach and Examples	278

APPENDIX 1. Mathematical Logic	291
APPENDIX 2. Vector Spaces, Norms, and Inner Products	300
APPENDIX 3. Derivation of the Least-Squares Approximation Formula .	313
REFERENCES	315
INDEX	318

MATLAB IMPLEMENTATIONS

All MATLAB software given in this dissertation was created using PC-MATLAB for MS-DOS Personal Computer 286 users, Version 3.5k .

Arranging Time Values in Increasing Order; Locating Repeated Time Values in a Data Set; Checking For a Uniform Time List; Supplying Alternate Indexing of a List	21
Reshaping Procedure to Test for Relatively Prime Cycles	39
Sums of Sinusoids with Close Periods, Amplitudes, Phase Shifts	90
Applying the Turning Point Test	122
Computing the Probability that the Martingale Algorithm will Stop in less than or equal to N steps, beginning with a series of length m . . .	139
Linear Least-Squares Approximation	150
Converting Functions to Discrete Orthogonal Functions	169
Nonlinear Least-Squares, Gradient Method	181
A Genetic Algorithm	188
Polynomial and Spline Interpolation	211
Discrete Fourier Series and the Periodogram	223
Finding the Periodogram, using a Fast Fourier Transform	241
Applying a Nonrecursive Filter	249
Finding the Transfer Function for a Symmetric Nonrecursive Filter . . .	264
Finding a Symmetric Nonrecursive Filter Corresponding to a Desired Transfer Function	274
Deleting 'Not A Number' Entries in a Matrix	279

the symbol
★

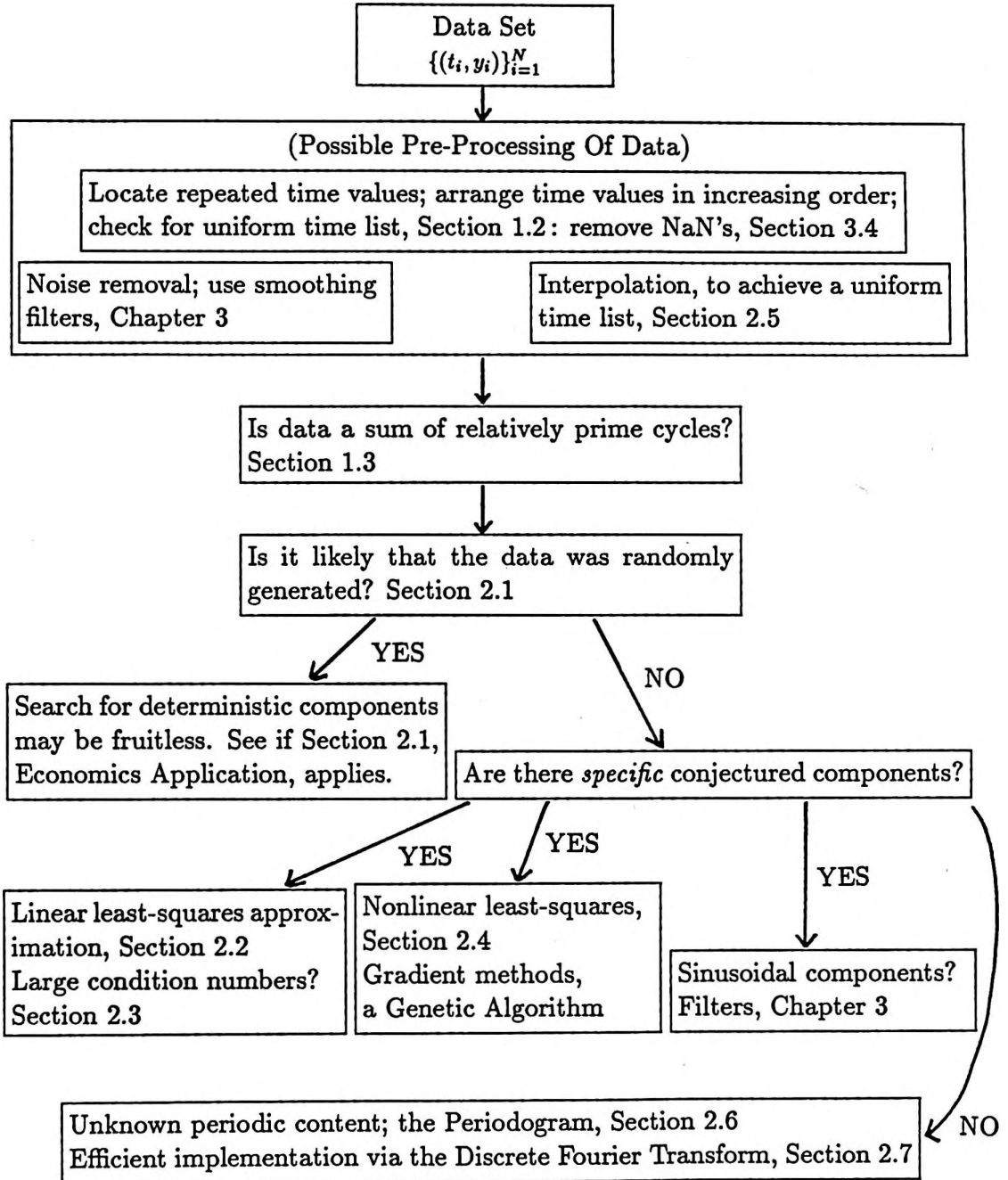
NOTE: The symbol ★ is used in this dissertation to mark occasional digressions that are important theoretically, but may go beyond the assumed prerequisites. Such material can be skipped over without any loss of continuity.

TABLE OF SYMBOLS

SYMBOL	MEANING OF SYMBOL	PAGE #
\exists, \forall	there exists, for all (for every)	
\subset, \subsetneq	set containment, proper set containment	
(t, y)	ordered pair (data point)	10, 13
$\{(t_i, y_i)\}_{i=1}^N \text{ or } \infty$	data set	10
$(t_i), (y_i)$	time list, list of data values	11, 12
i, j, k, m, n, M, N	integer variables	12
i	$i := \sqrt{-1}$, in context	12
$\mathbf{z} = (z_i)$	generic list	13
$:=, \mapsto$	equal, by definition; maps to	14, 15
$f: \mathcal{D}(f) \rightarrow B$	function notation	16
$\mathbf{R}, \mathbf{Z}, \mathbf{Z}^+, \mathbf{Q}, \mathbf{C}$	important sets of numbers	17, 18
$(a, b), [a, b], (a, b]$	interval notation	18, 19
f_e	extension of a function to \mathbf{R}	20
$\mu_{\mathbf{y}}$	mean of a finite list	33
$\tau_{\mathbf{x}}$	notation for a p -cycle	34
\mathbf{P}	set of all periods of a periodic function	41
glb	greatest lower bound	42
\star	material beyond assumed prerequisites	
lcm	least common multiple	54
$a b$	a divides b	54
$I_n, 0_{m,n}$	$n \times n$ identity, $m \times n$ zero matrices	64
$A \sin(\omega t + \phi)$	sinusoid	79
ω, f	radian and cyclic frequency	79
$g(t_i^-), g(t_i^+)$	one-sided limits	81
\int_P	integral over any interval of length P	81
e^{it}	complex exponential function	85
$\text{Re}(z), \text{Im}(z)$	real, imaginary parts of a complex number	86
$E(C), \mu_C$	expected value of C	111
$\text{var}(C), \sigma^2$	variance of C	111
$\ x\ $	norm of x	143
\mathbf{R}^n	Euclidean n -space	144
$\mathbf{y}^t, \mathbf{X}^t$	transpose of vector, matrix	145
\sup	supremum, least upper bound	157
$\ A\ _2$	2-norm for matrices	157
$\det(A)$	determinant of matrix A	162
\mathbf{A}_{ij}	matrix entry, i^{th} row, j^{th} column	164
$\langle \mathbf{u}, \mathbf{v} \rangle$	Euclidean inner product	165
$\mathbf{f}(\mathbf{T})$	vector-valued function of a vector variable	166
f_n	filter value	244
$(\dots, y_{-1}, \hat{y}_0, y_1, \dots)$	doubly infinite list with origin	252
$H(\omega), \tilde{H}(f)$	transfer functions	258, 259

FLOWCHART

The following flowchart suggests a strategy for analyzing a data set, using the tools presented in this dissertation. Section 3.4 contains examples illustrating the application of the procedure presented here.



Dedicated to
MOM and DAD

CHAPTER 1
PERIODIC FUNCTIONS

*The purpose of computing is insight,
not numbers.*

R.W. Hamming

1.1 The Purpose of This Dissertation

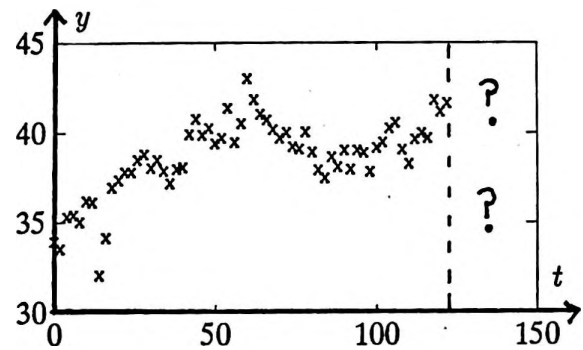
*the purpose of
this section*

This introductory section gives an informal discussion of the purpose of this dissertation. For this section only, the reader is assumed to have an intuitive understanding of the meaning of familiar words (like *data set* and *periodic*); precise meaning will be assigned to these concepts in later sections.

*the analyst
has a data set
in hand*

This dissertation is written for the person (hereafter called the *analyst*), who *has a data set in hand*, like

t	y
0	33.9300
2.0000	33.5100
4.0000	35.2900
6.0000	35.3900
8.0000	35.0100
10.0000	36.1700
12.0000	36.1100
14.0000	32.0300
16.0000	34.1200
18.0000	36.9200
20.0000	37.3400
22.0000	37.7600
\vdots	\vdots



and who seeks to *understand this data*, for the primary purpose of *predicting (forecasting) future behavior of the process that generated the data*.

Such a data set is completely described by a finite collection of ordered pairs $\{(t_i, y_i) \mid i = 1, \dots, N\}$. For most analysts, these ordered pairs will be stored in a computer in two columns (lists): a list of 'input' values $\{t_i\}_{i=1}^N$ and a list of corresponding 'output' values $\{y_i\}_{i=1}^N$:

t_1	y_1
t_2	y_2
t_3	y_3
\vdots	\vdots
t_N	y_N

The letter ' t ' is used merely because *time* is a common input; and the letter ' y ' is deeply entrenched in mathematical notation as denoting output values.

different hypotheses lead to different analyses

Different *hypotheses* (i.e., assumed truths) by the analyst regarding the nature of the data will lead to different methods for analyzing the data. For example, based on experience or initial knowledge about the mechanism(s) generating the data, a researcher might be led to the hypothesis that a particular data set is a random walk [Dgthy, 140–142]; or is a realization of a certain type of stochastic process [B&J]. Such hypotheses would dictate a particular investigative approach to be followed by the analyst.

the hypothesis to be investigated in this dissertation

The hypothesis to be investigated in this dissertation is that the output list is a finite sum of component functions. Particular emphasis is placed on the situation where at least one component is periodic. This type of hypothesis is investigated in the next example.

EXAMPLE

the output is a finite sum of component functions

Consider the schematic on the next page, which illustrates two periodic components (P_1 has period 3, P_2 has period 7) and a non-periodic component (a linear trend, T) that, together with some noise (N), sum to yield the data outputs denoted by Y : denote this by

$$P_1 + P_2 + T + N = Y .$$

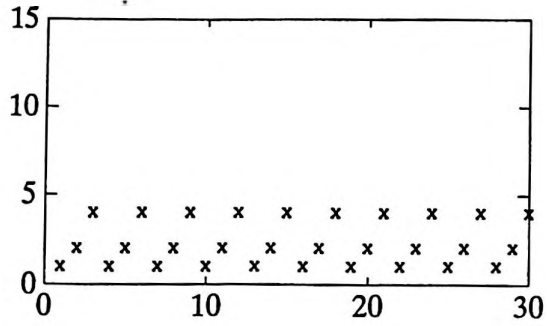
Although, in this example, the components of Y are known (they are P_1 , P_2 , T , and N), one must remember that in actuality *the analyst only has access to the data outputs Y* . Based on Y alone, the analyst's hope is to recover information about the components, and then use this information to predict future values of Y .

no hope of recovering the exact components

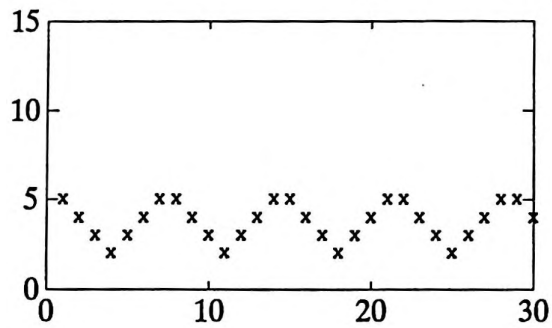
It should be noted immediately that the components P_1 , P_2 , T and N are not unique; for example, any real number K can be used to write Y as a sum of four components:

$$\overbrace{(P_1 - K)}^{\text{component 1}} + \overbrace{(P_2 + K)}^{\text{component 2}} + T + N = Y .$$

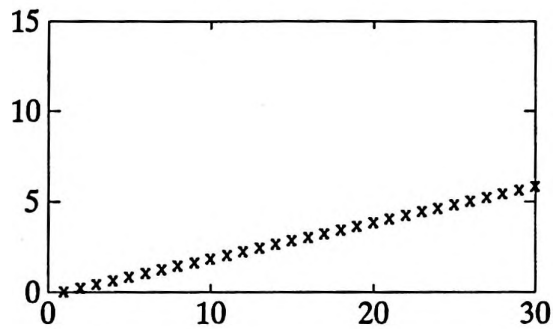
The 'new' component $P_1 - K$ still has period 3; the 'new' component $P_2 + K$ still has period 7; and these four components still sum to give Y . Many other combinations are possible. Thus, there is no hope of recovering the *exact* components that make up the output data.



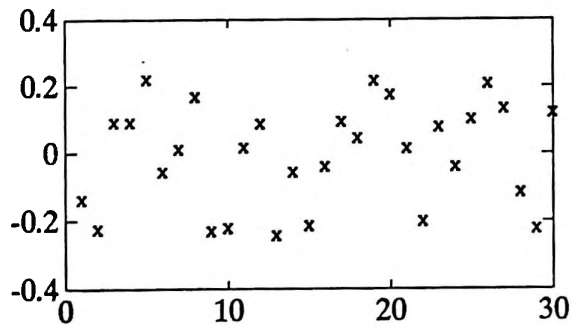
P_1 (PERIOD-3 COMPONENT)



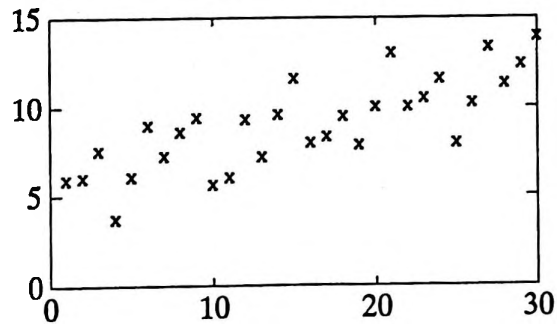
+ P_2 (PERIOD-7 COMPONENT)



+ T (LINEAR TREND)



+ N (NOISE)



= Y (OUTPUT DATA)

'known
unknown'

The output list Y just constructed is an example of what will be referred to, throughout this dissertation, as a *known unknown*. It is *known* because it was constructed by the analyst. However, the components that make up Y will be *assumed to be unknown* (to varying degrees) in order to test various techniques for identifying components.

CAUTION
the problem of
not knowing
the truth
of a hypothesis

A very simple example is presented next to illustrate some important logical considerations that must constantly be kept in mind. Suppose that an analyst is presented with the (unrealistically small) data set

$$(t_1, y_1) = (T, 1),$$

$$(t_2, y_2) = (2T, 2),$$

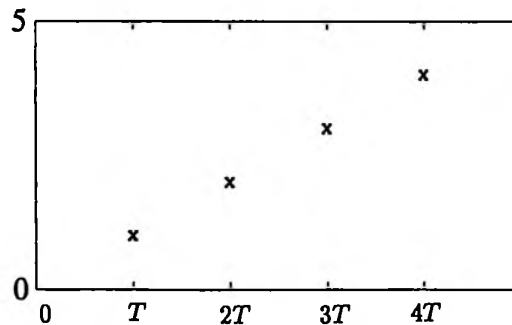
$$(t_3, y_3) = (3T, 3),$$

$$(t_4, y_4) = (4T, 4).$$

This data is stored in the computer as two columns of numbers:

T	1
$2T$	2
$3T$	3
$4T$	4

Here, T is any positive number; the particular value of T is unimportant to the current discussion, except for the consequence that the data is equally-spaced. This data set is graphed below.



Unknown to the analyst, this data was generated as the sum of a 2-cycle and a 3-cycle,

$$\begin{aligned} & (3, \quad 6, \quad 3, \quad 6, \quad 3, \quad 6, \quad 3, \quad 6, \quad 3, \quad \dots) \\ + & (-2, \quad -4, \quad 0, \quad -2, \quad -4, \quad 0, \quad -2, \quad -4, \quad 0, \quad \dots) \\ = & (1, \quad 2, \quad 3, \quad 4, \quad -1, \quad 6, \quad 1, \quad 2, \quad 3, \quad \dots) , \end{aligned}$$

producing a 6-cycle; but not even one cycle of the resulting 6-cycle has yet been observed by the analyst. Now that you are privy to this component information, *forget it*, and continue with the analysis.

first hypothesis

Based on the four pieces of observed data (1,2,3,4), you, as the analyst, hope to predict future behavior. Thus, you make an initial hypothesis:

HYPOTHESIS #1: The data is being generated by the linear function $y_n = n$.

Based on this hypothesis, you predict the next data point:

$$(y_5)_{\text{predicted}} = 5 .$$

When the next data value, -1 , becomes available, you have learned something: your hypothesis, at least in its 'purest' form, was incorrect.

*underlying
logic*

Let's look at the logic underlying this conclusion. Remember that the mathematical sentence

$$\text{'If } A, \text{ then } B\text{' } \tag{S1}$$

is logically equivalent to its contrapositive,

$$\text{'If not } B, \text{ then not } A\text{' } . \tag{S2}$$

For ease of notation, denote the sentence 'If A , then B ' by $S1$; and denote the sentence 'If not B , then not A ' by $S2$. The fact that $S1$ is logically equivalent to $S2$ means that $S1$ and $S2$ always have the same truth values: if $S1$ is true, so is $S2$; if $S1$ is false, so is $S2$; if $S2$ is true, so is $S1$; and if $S2$ is false, so is $S1$. (See Appendix 1 for a discussion of mathematical logic.)

The sentence

IF the data set is generated by $y_n = n$, THEN $y_5 = 5$

is a true mathematical sentence. Therefore, its contrapositive,

IF $y_5 \neq 5$, THEN the data set is not generated by $y_n = n$

is also true. Since $y_5 \neq 5$ (is true), it is concluded that $y_n = n$ is not the correct generator for the data.

second hypothesis Since hypothesis #1 is incorrect, it is discarded, and a new hypothesis is sought. The analyst is still struck by the linearity of the first four data values, and decides that this fifth data value is an anomaly—a mistake—an outlier—caused perhaps by some temporary outside influence. Thus, the hypothesis is only slightly modified:

HYPOTHESIS #2: The data set is being generated by the function $y_n = n$, for all $n \neq 5$.

Under this slightly modified hypothesis, the next data value is predicted: $(y_6)_{\text{predicted}} = 6$. When the next data value is observed, one indeed sees that $y_6 = 6$. This new information supports hypothesis #2. That is, at this point, there is no reason to reject hypothesis #2. *However, the analyst cannot conclude that hypothesis #2 is true.* It might be (as, indeed it is) that y_6 equals 6 for reasons other than hypothesis #2 being true.

Keeping hypothesis #2, the analyst predicts that $(y_7)_{\text{predicted}} = 7$. Upon observing $y_7 = 1$, however, hypothesis #2 must be discarded.

third hypothesis Suppose that the analyst is fortunate enough to make the following conjecture (educated guess):

HYPOTHESIS #3: The output list is a sum of a 2-cycle and a 3-cycle.

Such a sum must repeat itself every six entries. Consequently, if hypothesis #3 is made *after* having observed the first six outputs (1, 2, 3, 4, -1, 6), then no analysis needs to be done; the analyst immediately concludes that these six outputs must repeat themselves:

$$(1, 2, 3, 4, -1, 6, 1, 2, 3, 4, -1, 6, \dots) .$$

*basic reshaping
techniques to
gain further insight
into components*

Although future values have been correctly predicted in this case, the analyst may still desire more information about the components making up the output list. Some basic reshaping techniques (discussed in section 1.3) can be used on the first six pieces of data to find two components that can be used for prediction. The technique is illustrated next:

- Let

$$\mathbf{Y} = (1, 2, 3, 4, -1, 6) .$$

- Find the mean (average) of the entries in \mathbf{Y} , and denote it by μ_Y :

$$\mu_Y = \frac{1}{6}(1 + 2 + 3 + 4 + (-1) + 6) = 2.5 .$$

- Construct a 'mean-zero' output list \mathbf{Y}_0 by subtracting μ_Y from each entry in \mathbf{Y} :

$$\begin{aligned} \mathbf{Y}_0 &= \quad (\quad 1, \quad 2, \quad 3, \quad 4, \quad -1, \quad 6) \\ &\quad - (\quad 2.5, \quad 2.5, \quad 2.5, \quad 2.5, \quad 2.5, \quad 2.5) \\ &= \quad (-1.5, \quad -.5, \quad .5, \quad 1.5, \quad -3.5, \quad 3.5) . \end{aligned}$$

- 'Reshape' \mathbf{Y}_0 to test for a period-2 component, and take the mean of each column. This gives the mean-zero 2-component, called c_1 below.

$$\begin{array}{r|l} \begin{array}{r} -1.5000 \\ 0.5000 \\ -3.5000 \end{array} & \begin{array}{r} -0.5000 \\ 1.5000 \\ 3.5000 \end{array} \\ \hline c_1 = & \begin{array}{r} -1.5000 \\ 1.5000 \end{array} \end{array}$$

- 'Reshape' \mathbf{Y}_0 to test for a period-3 component, and take the mean of each column. This gives the mean-zero 3-component, called c_2 below.

$$\begin{array}{r|l|l} \begin{array}{r} -1.5000 \\ 1.5000 \end{array} & \begin{array}{r} -0.5000 \\ -3.5000 \end{array} & \begin{array}{r} 0.5000 \\ 3.5000 \end{array} \\ \hline c_2 = & \begin{array}{r} 0 \\ -2 \end{array} & \begin{array}{r} 2 \end{array} \end{array}$$

- If Y is indeed the sum of a 2-cycle and a 3-cycle, then adding μ_Y to c_1 and c_2 must give back Y , and it does:

$$\begin{aligned}
 & (-1.5, 1.5, -1.5, 1.5, -1.5, 1.5) \\
 + & (0, -2, 2, 0, -2, 2) \\
 + & (2.5, 2.5, 2.5, 2.5, 2.5, 2.5) \\
 = & (1, 2, 3, 4, -1, 6) .
 \end{aligned}$$

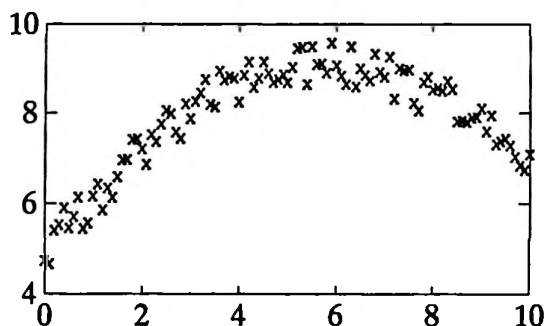
*only 4 outputs
need be observed
for predictive
purposes*

It is interesting to note that if the output list is truly a sum of a 2-cycle and a 3-cycle, then once only 4 outputs have been observed, future behavior can be predicted. See section 1.5 for details.

EXAMPLE
*a beautiful fit
does NOT mean
that the
components
are correct!*

Here is a second example, more realistic in terms of data set size than the first, which also emphasizes the problem of *not knowing if a hypothesis is true*.

Suppose that an analyst is presented with the data set shown below:



*the
known unknown*

Unknown to the analyst, this data set was generated by the continuous function

$$y(t) = 3 \sin\left(\frac{2\pi t}{20}\right) + .2t + 5 + \langle \text{noise} \rangle ,$$

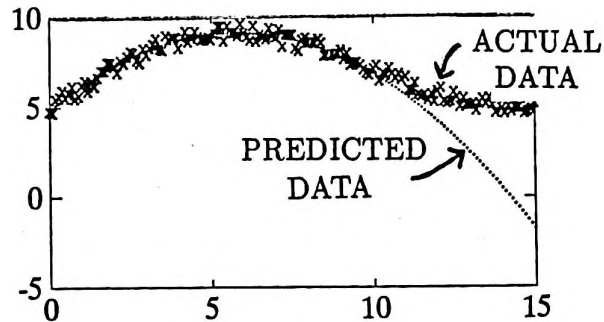
sampled at increments of .1 on the interval $[0, 10]$.

first hypothesis

The data certainly looks like a piece of a parabola, and the analyst may make the initial hypothesis:

HYPOTHESIS #1: $h_1(t) = b_0 + b_1t + b_2t^2$.

Motivated by this hypothesis, values of b_0 , b_1 , and b_2 are sought so that $h_1(t)$ 'fits' the data in the best least-squares sense (see section 2.2). Having done so, the analyst finds that the agreement between $h_1(t)$ (the hypothesized data generator) and $y(t)$ (the actual data) looks beautiful on the original data interval $[0, 10]$. However, when $h_1(t)$ is used to forecast behavior on the interval $[10, 15]$, a comparison with $y(t)$ shows that the fit is *terrible*.



*A beautiful fit
does NOT assure
that the hypothesis
is correct!*

The hypothesized components led to a 'fit' that was beautiful on the data currently available to the analyst. However, using this 'fit' to predict future values led to huge errors. What happened? It is indeed true that *IF* the hypothesis is correct, *THEN* the fit will be beautiful. However, *the 'fit' may be beautiful for reasons other than the hypothesis being correct.*

In this case, the Weierstrass Approximation Theorem [Bar, p. 172] can be cited as the reason that the 'fit' is beautiful. This theorem states that *any* continuous function defined on a closed interval can be uniformly approximated by polynomials; consequently, a good 'fit' can *always* be obtained, if a sufficiently high degree polynomial is used.

the moral

The moral is:

A 'beautiful fit' never assures the analyst that the conjectured components are correct.

The fit may be beautiful for reasons *other* than the hypothesis being correct, potentially rendering the hypothesized components useless for predictive purposes.

R.W. Hamming, author of several books in numerical analysis and digital filters, emphasizes that *the purpose of computing is insight, not numbers*. The analyst must always be attentive to the possibility that the hypotheses are not correct.

1.2 Notation

Introduction

In this section, notation is developed that will be used throughout the dissertation.

data sets

Many data sets consist of a collection of ordered pairs of real numbers: each ordered pair (t, y) in the collection contains an *input* to some data-generation process, denoted by t , and its corresponding *output* from the data-generation process, denoted by y .

In this dissertation, the word 'time' is used to refer to inputs.

The data sets to be studied in this dissertation have the additional property that the times can be *listed*; and, they can be listed in *increasing order*. This leads to the definition of *discrete-domain data*, stated next:

DEFINITION *discrete-domain data set*

A set of ordered pairs with real-number entries is called a *discrete-domain data set*, if the following conditions hold:

- The total number of ordered pairs in the set is either finite, or countably infinite (i.e., can be put in a one-to-one correspondence with the positive integers). Thus, the set can be notated as $\{(t_i, y_i)\}_{i=1}^N$, if it is finite; or $\{(t_i, y_i)\}_{i=1}^{\infty}$, if it is countably infinite.
- The time values, $\{t_i\}_{i=1}^{(N \text{ or } \infty)}$, can be arranged in strictly increasing order. In particular, the time values are all distinct.

MATLAB implementation

MATLAB commands for checking that the time values in a finite data set are distinct, and re-ordering the ordered pairs so that the times increase, are provided at the end of this section.

a discrete-domain data set can be described as a pair of lists

Every discrete-domain data set can be described as a pair of lists. These lists may be vertical in orientation:

$$\begin{array}{cc} t_1 & y_1 \\ t_2 & y_2 \\ t_3 & y_3 \\ \vdots & \vdots \end{array}$$

Or, the lists may be horizontal in orientation:

$$(t_1, t_2, t_3, \dots) \text{ and } (y_1, y_2, y_3, \dots).$$

In both orientations, time t_k always corresponds to output y_k ; that is, the pair (t_k, y_k) is an element of the data set.

Both horizontal and vertical orientations will be used in this dissertation, the choice depending on which orientation is best suited to a particular situation.

In what follows, the idea of a ‘list of numbers’ is formalized, and properties of lists are discussed. For ease of notation, a horizontal orientation for the lists is used.

lists

This dissertation will investigate *lists of real numbers*, like

$$(t_1, t_2, t_3, \dots, t_N) \quad (1)$$

or

$$(y_1, y_2, y_3, \dots) \quad (2)$$

or

$$(\dots, z_{-2}, z_{-1}, z_0, z_1, z_2, \dots) . \quad (3)$$

*entry;
element;
member*

The list may be finite or infinite. In a horizontal orientation, the numbers in the list are separated by commas, and enclosed in parentheses. A number in a list is called an *entry*, an *element*, or a *member* of the list.

time lists

The letter used in a list will influence the interpretation of the members in the list.

When the letter ‘*t*’ is used, the list is assumed to consist of *inputs* (times), and is called a *time list*. Time lists have an additional requirement: the times must (strictly) increase as one moves through the list, from left to right in a horizontal orientation, or from top to bottom in a vertical orientation.

The time values $\{t_i\}_{i=1}^{(N \text{ or } \infty)}$ in a discrete-domain data set can be made into a time list.

The list in (1) is often abbreviated as $(t_i)_{i=1}^N$.

*examples and
non-examples of
time lists*

Here are some examples of valid time lists:

$$\begin{aligned} &(1, 2, 3, \dots) \\ &(\dots, -1, -\frac{1}{2}, 0, \frac{1}{3}, \frac{2}{3}, 1, \dots) \\ &(1, 5, 7.1, 100) \end{aligned}$$

The following lists do *not* meet the special requirement of a time list:

$$(1, 1, 2, 2, 3, 3, \dots)$$

$$(0, 1, 3, 2, 4)$$

★

The rational numbers are countable, so they can be listed. Note, however, that they *cannot* be listed in increasing order. Thus, the rational numbers cannot form a time list.

output lists

When the letter 'y' is used in a list, the list is assumed to consist of *outputs*, and is called an *output list*.

Letters other than *t* and *y* are used when the interpretation of the list is unimportant.

i, j, k, m, n,
M, N
denote
integer values

When subscripts are used in a list, they must be integers, and must increase as one moves through the list from left to right (or from top to bottom). When the variables *i, j, k, m, n, M* and *N* are used in this dissertation, it is assumed that they are *integer* variables, unless otherwise specified. (In the proper context, *i* is used to denote $\sqrt{-1}$.)

length of
a list

The total number of entries in a finite list is referred to as the *length of the list*. Thus, for example, the list (2, 4, 6, 8, 10) has length 5, not, say, length 10 – 2.

The letters '*N*' and '*M*' are often used for the last entry in a finite list. Note that the list $(z_1, z_2, z_3, \dots, z_N)$ has length *N*, whereas the list $(z_0, z_1, z_2, z_3, \dots, z_N)$ (where the subscript begins at 0) has length *N* + 1.

The subscripts in a list need not begin at 1. In the list

$$(z_k, z_{k+1}, z_{k+2}, \dots, z_M), \quad (4)$$

where the increment in the subscripts is 1, the length of the list is

$$M - (k - 1) = M - k + 1.$$

The list in (4) is abbreviated as $(z_i)_{i=k}^M$.

If the increment in the subscripts is $j > 1$, the list becomes

$$(z_k, z_{k+j}, z_{k+2j}, z_{k+3j}, \dots, z_M)$$

where M is necessarily of the form $k + Nj$ for some positive integer N . The length of this list is

$$\frac{M - (k - j)}{j} = \frac{M - k}{j} + 1.$$

$(y_i)_{i=1}^{\infty}$

When a list is infinite, as in (2) and (3), the ellipsis ‘...’ indicates that the entries continue ad infinitum. The lists in (2) and (3) are abbreviated as $(y_i)_{i=1}^{\infty}$ and $(z_i)_{i=-\infty}^{\infty}$, respectively.

*simpler notation
for lists*

For any list, the notation (z_i) (without any indexing) is used in the following situations:

- if it is *unimportant* whether the list is finite or infinite; or,
- if the particular nature of the list is understood from context.

*distinguish
the list (z_i)
from
an element z_i*

In all cases, it is important to distinguish the list, (z_i) , from a particular element in the list, z_i . Lists are often denoted by boldface letters, for example, $\mathbf{z} = (z_i)$.

*data point,
 (t_k, y_k) ;
data value, y_k*

Sometimes it is necessary to focus attention on a particular input/output pair, (t_k, y_k) . Such a pair is called a *data point*. The output y_k is referred to as the *data value* (at time t_k).

*the phrase:
‘ i^{th} entry
in a list’*

In a list that has a first entry (like (1) and (2), but not (3)), the phrase ‘*the i^{th} entry in a list*’ always refers to the number that occupies the i^{th} slot from the left (or top) in the list, as the next example illustrates:

The fifth entry in the list $(y_3, y_4, y_5, y_6, y_7, \dots)$ is y_7 .

*operations
on lists*

All operations on lists are done component-wise. For example, given two lists \mathbf{z} and \mathbf{w} , each of length N , the sum $\mathbf{z} + \mathbf{w}$ is also a list of length N , and the i^{th} entry in $\mathbf{z} + \mathbf{w}$ is the sum of the i^{th} entry in \mathbf{z} and the i^{th} entry in \mathbf{w} .

For example,

$$(1, 2, 3, 4, 5) + (5, 4, 3, 2, 1) = (6, 6, 6, 6, 6)$$

and

$$(z_2, z_4, z_6) + (w_1, w_2, w_3) = (z_2 + w_1, z_4 + w_2, z_6 + w_3).$$

Whenever lists that are ‘infinite in both directions’ are summed, the alignment of the lists will determine the entries that are to be added: for example,

$$\begin{aligned} & (\dots, 1, 2, 1, 2, 1, 2, 1, 2, \dots) \\ + & (\dots, 0, 2, 3, 4, 0, 2, 3, 4, \dots) \\ = & (\dots, 1, 4, 4, 6, 1, 4, 4, 6, \dots) \end{aligned}$$

*scaling of
a list*

The scaling of a list by a constant k is defined by

$$k(z_i) := (kz_i) .$$

The symbol ‘ $:=$ ’ just used emphasizes that the equality is *by definition*.

equality of lists

The order in a list is important: therefore, two lists z and w are equal if and only if the i^{th} entry in z equals the i^{th} entry in w , for all i .

No attempt is made to define equality of lists that are infinite in both directions, as in (3).

*uniform
time list*

It is very common for time lists to have the property that their entries are *equally-spaced*. This means that there is a constant $T > 0$ such that successive entries t_k and t_{k+1} in the list always have difference T , that is,

$$t_{k+1} - t_k = T .$$

A time list with this property is referred to as a *uniform time list*.

*MATLAB
implementation*

MATLAB commands to check for a uniform time list are provided at the end of this section.

*changing
time scales*

Let s denote a starting time and let $T > 0$. The uniform time list

$$(s, s+T, s+2T, \dots, \underbrace{s+(i-1)T}_{i^{\text{th}} \text{ entry}}, \dots, \underbrace{s+(N-1)T}_{N^{\text{th}} \text{ entry}}, \dots)$$

can be easily converted to a uniform time list of the form

$$(1, 2, 3, \dots)$$

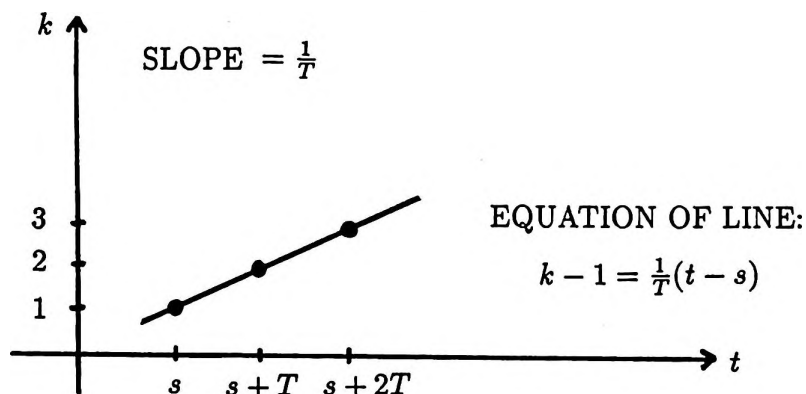
by use of the transformation

$$t \mapsto \frac{1}{T}(t-s) + 1 .$$

That is, the i^{th} member of the initial time list maps to the positive integer i .

The symbol ' \mapsto ' is read as '*maps to*', and means that the element t is sent to the element $\frac{1}{T}(t-s)+1$.

This transformation was found by writing down the equation of the line shown below:



*transforming
back*

It is equally easy to transform the list $(1, 2, 3, \dots)$ back into a list with starting time s , and uniform spacing T . Indeed, the transformation

$$k \mapsto T(k-1) + s$$

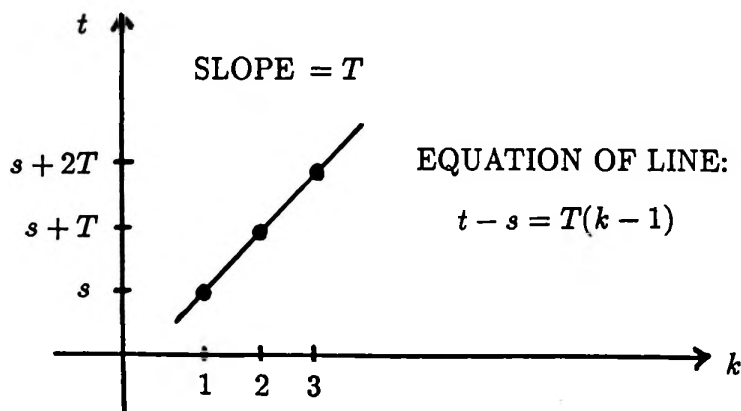
maps the list

$$(1, 2, 3, \dots, i, \dots, N, \dots)$$

to the list

$$(s, s+T, s+2T, \dots, s+(i-1)T, \dots, s+(N-1)T, \dots).$$

This transformation was found by writing down the equation of the line shown below:



**MATLAB
implementation**

MATLAB commands for converting any finite uniform time list to the list $(1, 2, 3, \dots, N)$; and for converting $(1, 2, 3, \dots, N)$ back to a time list with starting value s and spacing T , are provided at the end of this section.

If an output list (y_1, y_2, \dots, y_N) corresponds to a uniform time list, then it has been shown that this associated time list can *always* be labeled $(1, 2, 3, \dots, N)$. Consequently, there is often no need to 'carry around' the associated time list. In such cases, the time list is often suppressed, and only the output list is given.

functions

Functions are an extremely useful mathematical tool for working with input-output relationships that are characterized by each input having exactly one associated output. A convenient function notation that will be used throughout this dissertation is introduced next.

DEFINITION

function;

function notation;

$f: \mathcal{D}(f) \rightarrow B;$

domain $\mathcal{D}(f);$

codomain $B;$

range $\mathcal{R}(f)$

A *function* f is a rule that associates to each input x a unique output denoted by $f(x)$. The set of inputs to f is called the *domain* of f and is denoted by $\mathcal{D}(f)$. The notation

$$f: \mathcal{D}(f) \rightarrow B$$

symbolically indicates the rule f , the domain $\mathcal{D}(f)$, and a set B that contains the outputs of f . The set B is called the *codomain* of f .

The actual output set of f is called the *range of f* and is denoted by $\mathcal{R}(f)$. The range is found by letting f act on all domain elements, and forming a set from the resulting function values; this action is sometimes indicated by

$$\mathcal{R}(f) = \{f(x) \mid x \in \mathcal{D}(f)\} .$$

In the notation $f: \mathcal{D}(f) \rightarrow B$, it must be that $\mathcal{R}(f) \subset B$.

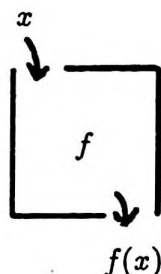
**the difference
between
 B and $\mathcal{R}(f)$**

The set B (the *codomain* of f) gives information about the *type* of outputs that the rule f generates. The set $\mathcal{R}(f)$ gives information about the *actual* outputs that the rule f generates. For example, consider the function described by

$$f: \{x \mid 1 \leq x \leq 3\} \rightarrow \mathbf{R}, \quad f(x) = x^2 .$$

Here, the fact that the codomain is \mathbf{R} (where \mathbf{R} denotes the set of real numbers), implies that the outputs from f are real numbers. To find the range of f , $\mathcal{R}(f)$, one must see precisely what outputs are generated as the squaring function acts on all elements in the domain of f :

$$\mathcal{R}(f) = \{f(x) \mid 1 \leq x \leq 3\} = \{x^2 \mid 1 \leq x \leq 3\} = \{y \mid 1 \leq y \leq 9\}.$$



Note that the function notation $f: \mathcal{D}(f) \rightarrow B$ does not include information about *how* the outputs are related to the inputs. Such information must be provided separately.

It is important to emphasize that f is the *name of the function*; and $f(x)$ is the output of f corresponding to the input x .

DEFINITION
discrete-domain
function

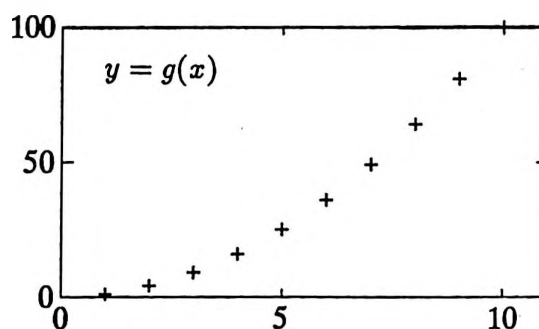
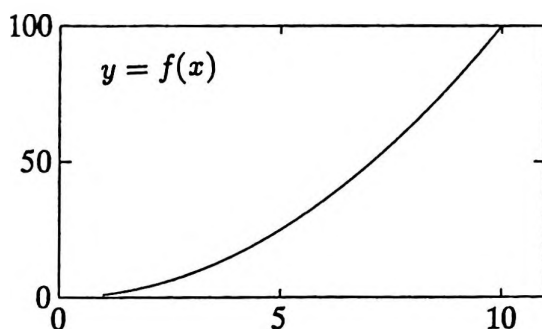
A function f is called a *discrete-domain function* if its domain consists of entries from a time list. Thus, the domain of a discrete-domain function is a set of time values (either finite or countably infinite), that can be listed in strictly increasing order.

In particular, a discrete-domain function has the property that its domain cannot contain any interval of real numbers.

EXAMPLE
using
function notation

The function $f: [1, 10] \rightarrow \mathbf{R}$ defined by $f(x) = x^2$ has the graph shown below. A related discrete-domain function $g: \{1, 2, \dots, 10\} \rightarrow \mathbf{R}$ defined by $g(x) = x^2$ is also graphed.

Note that $\mathcal{R}(f) = [1, 100]$, and $\mathcal{R}(g) = \{1^2, 2^2, 3^2, \dots, 10^2\}$.



real-valued
function

In this dissertation, the codomain set will always be the real or complex numbers. If the codomain is \mathbf{R} , then the function is called a *real-valued function*. If, in addition, the domain is a subset of \mathbf{R} , then the function is called a *real-valued function of a real variable*.

NOTATION <i>used with functions</i>	The following notation is commonly used in connection with functions:
Z, integers	The symbol Z denotes the set of integers: $\mathbf{Z} := \{\dots, -3, -2, -1, 0, 1, 2, 3, \dots\} .$
Z⁺, <i>positive integers</i>	The symbol Z⁺ denotes the set of positive integers: $\mathbf{Z}^+ := \{1, 2, 3, \dots\} .$ More generally, the superscript '+' always denotes the positive (strictly greater than zero) elements from a specified set.
Q, <i>rational numbers</i>	The symbol Q denotes the set of rational numbers; i.e., the numbers that are expressible as a Quotient of integers (with nonzero denominator).
C, <i>complex numbers</i>	The symbol C denotes the set of complex numbers, i.e., those numbers that can be written in the form $a + bi$, where a and b are real numbers, and $i := \sqrt{-1}$. (Many engineers use j to denote $\sqrt{-1}$, since i is reserved for electrical current.)
<i>interval notation</i>	There is a convenient notation for intervals of real numbers. Let a and b be real numbers, with $a < b$.
(a, b) , <i>open interval</i>	An interval that does not include either endpoint, like $(a, b) := \{x \mid a < x < b\} ,$ is called an <i>open interval</i> .
$[a, b]$, <i>closed interval</i>	An interval that includes both endpoints, like $[a, b] := \{x \mid a \leq x \leq b\} ,$ is called a <i>closed interval</i> .
$(a, b], [a, b)$, <i>half-open intervals</i>	An interval that includes exactly one endpoint, like $(a, b] := \{x \mid a < x \leq b\} \text{ or } [a, b) := \{x \mid a \leq x < b\} ,$ is called a <i>half-open interval</i> .

infinite intervals Infinite intervals can be accommodated by using the symbol ‘ ∞ ’ (‘infinity’), such as

$$[a, \infty) := \{x \mid x \geq a\} .$$

every discrete-domain data set has an associated discrete-domain function Observe that every discrete-domain data set $\{(t_i, y_i)\}_{i=1}^{(N \text{ or } \infty)}$ is naturally associated with a discrete-domain function f that maps each input t_i to the value y_i , i.e., $f(t_i) = y_i$. Thus, all discussions concerning *discrete-domain data sets* can be rephrased in terms of *discrete-domain functions*, if it is convenient to do so.

discrete signal There are various phrases used in the existing literature that one should be aware of. In communications and digital systems engineering, the phrase *discrete signal* usually refers to a data set associated with a function that has, as its domain, the entries from a time list. The codomain may be any subset of \mathbb{R} .

digital signal A *digital signal* is a special type of discrete signal, where not only the domain, but also the *codomain*, must consist of entries from a time list. Thus, both discrete and digital signals are ‘sampled in time’. However, a discrete signal has the *potential* of taking on values in an interval; whereas a digital signal may only take its values from a time list. The difference is further clarified in the next example.

EXAMPLE
discrete signal versus digital signal The data set associated with a function $f: \{1, 2, 3, \dots\} \rightarrow [0, \infty)$ is a discrete signal, but not a digital signal, because f has the *potential* of taking on values in an interval. The data set associated with a function $g: \{1, 2, 3, \dots\} \rightarrow \{0, 0.01, 0.02, 0.03, \dots\}$ is a digital signal, because g only has the potential of taking on values from a time list.

time series In time series analysis and statistical literature, a *time series* is any list of observations generated sequentially in time, where there is thought to be a *dependence between the observations* and where the nature of this dependence is of interest.

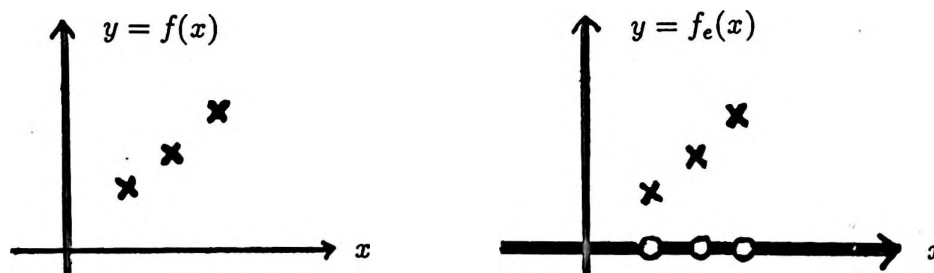
extending to \mathbf{R}
 f_e

To conclude this section on notation, observe that any real-valued function f defined on a proper subset of \mathbf{R} can always be *extended* to a function f_e defined on \mathbf{R} , by defining f_e to be zero for any input not in $\mathcal{D}(f)$. That is, define

$$f_e(x) := \begin{cases} f(x) & \text{for } x \in \mathcal{D}(f) \\ 0 & \text{for } x \notin \mathcal{D}(f) \end{cases}.$$

Whenever the notation f_e is used in this dissertation, it will denote this particular extension of a function f to \mathbf{R} .

A function and its extension to \mathbf{R} are illustrated below.



MATLAB
implementation

MATLAB ('MATrix LABoratory') is a software package for numeric computation that has become quite popular in academia because of its powerful capabilities and ease of use. MATLAB commands will be used throughout this dissertation to implement the techniques discussed herein.

All MATLAB software given in this dissertation was created using PC-MATLAB for MS-DOS Personal Computer 286 users, Version 3.5k.

The reader is assumed to be a competent MATLAB user. However, as a convenience to the reader, the MATLAB commands used are reviewed on their first appearance. Examples are provided to illustrate the command sequences.

MATLAB IMPLEMENTATION

PURPOSE

- To check that the time values in a data set are arranged in increasing order; if not, an appropriate rearrangement is made.
- To locate any repeated time values in a data set.
- To check for a *uniform* time list.
- To supply an alternate indexing of a uniform list: if the uniform list is of length N , it can alternately be indexed by $(1, \dots, N)$.
- To supply an alternate indexing of the list $(1, \dots, N)$: if s denotes a desired starting time and T denotes a desired positive uniform spacing, then the alternate list

$$(s, s + T, \dots, s + (N - 1)T)$$

is produced.

REQUIRED INPUTS

The data set being investigated must be contained in an $N \times 2$ matrix, denoted here by z . The first column of z contains the time values; the second column contains the corresponding outputs.

$z =$

$$\begin{array}{cc} t_1 & y_1 \\ t_2 & y_2 \\ t_3 & y_3 \\ \vdots & \vdots \\ t_N & y_N \end{array}$$

The positive integer N denotes the number of ordered pairs in the data set.

MATLAB COMMANDS

*re-ordering
the data set
and checking for
repeat time values*

The commands listed on the following page are used to re-order the ordered pairs, so that the resulting time values are in increasing order. The resulting (re-ordered) matrix *replaces* the original matrix.

Then, locations of any repeated time values in the re-ordered matrix are given.

The lines are numbered for easy reference in the discussion that follows.

```

1)  t = z(:,1);
2)  y = z(:,2);
3)  [t,i] = sort(t);
4)  y = y(i);
5)  z = [t y];
6)  for j = 1:(length(t)-1),
7)      if t(j) == t(j+1),
8)          j
9)      end
10) end

```

line 1 line 1: The first column of **z** is named **t**. The colon operator ':' is used here to denote *all* the rows of the matrix **z**. The semicolon ';' at the end of the line suppresses MATLAB echoing.

line 2 line 2: The second column of **z** is named **y**.

line 3 line 3: The entries of **t** are sorted in ascending order; the resulting column vector is again denoted by **t**. The list **i** contains the re-ordering information, and is used next to correspondingly re-order the entries in **y**.

line 4 line 4: The entries in **y** are re-arranged to coincide with the new arrangement in **t**. This re-arrangement of **y** is again denoted by **y**.

line 5 line 5: The original matrix **z** is replaced by the re-ordered matrix.

lines 6-10 lines 6-10: If entries **j** and **j+1** of the (re-ordered) list **t** are identical, then the value **j** is returned. The analyst can then make appropriate adjustments to the data set.

If desired, lines 6-10 could be stored in an m-file, say, **chkfdup.m** (check for duplicates). Then, lines 6-10 would be replaced by the single command, **chkfdup**.

checking for a uniform time list Continuing the commands above, the adjusted list **t** is checked, to see if it is a *uniform* time list. If not, the positions of 'errant' time values are returned in the list **err**.

```

11) d = diff(t);
12) p = ( ones(d)*d(1) ~= d );
13) err = find(p);

```

line 11

line 11: The vector \mathbf{d} contains the successive *differences* of the entries in \mathbf{t} . That is,

$$\mathbf{d}(1) = \mathbf{t}_2 - \mathbf{t}_1$$

$$\mathbf{d}(2) = \mathbf{t}_3 - \mathbf{t}_2$$

$$\vdots$$

If \mathbf{t} is uniform, then all the entries in \mathbf{d} are identical; and if all the entries in \mathbf{d} are identical, then \mathbf{t} is uniform.

line 12

line 12: The list $\mathbf{ones}(\mathbf{d}) * \mathbf{d}(1)$ has the same size as \mathbf{d} , with all entries equal to $\mathbf{d}(1)$. Instead of $\mathbf{d}(1)$, the analyst may choose to use $\mathbf{d}(j)$ for values of j greater than 1.

Via the command ' $\mathbf{ones}(\mathbf{d}) * \mathbf{d}(1) \sim= \mathbf{d}$ ', the list $\mathbf{ones}(\mathbf{d}) * \mathbf{d}(1)$ is compared to the list \mathbf{d} .

If the relation ' $\mathbf{ones}(\mathbf{d}) * \mathbf{d}(1) \sim= \mathbf{d}$ ' is TRUE, then a value of '1' is returned in the list \mathbf{p} . At these positions, a spacing different than the spacing between \mathbf{t}_2 and \mathbf{t}_1 is encountered.

If the relation ' $\mathbf{ones}(\mathbf{d}) * \mathbf{d}(1) \sim= \mathbf{d}$ ' is FALSE, then a value of '0' is returned in the list \mathbf{p} .

line 13

line 13: The MATLAB command ' \mathbf{find} ' locates the nonzero entries in a matrix. Each entry in \mathbf{p} is either '1' or '0'; therefore, the locations of the '1' entries in \mathbf{p} are recorded in \mathbf{err} .

alternate
indexing

The command

$(1:\mathbf{length}(\mathbf{t}))'$;

produces a vertical list of the positive integers $1, 2, 3, \dots, \mathbf{N}$, where $\mathbf{N} = \mathbf{length}(\mathbf{t})$.

Given a list $\mathbf{t} = (1, 2, 3, \dots, \mathbf{N})$, the command

$\mathbf{resp_t} = (\mathbf{s}:\mathbf{T}:\mathbf{s}+(\mathbf{length}(\mathbf{t})-1)*\mathbf{T})'$; % 'resp' is for 'respace'

produces a vertical list of length \mathbf{N} , starting at \mathbf{s} and with uniform spacing \mathbf{T} .

EXAMPLE The following diary of an actual MATLAB session illustrates the command sequences just discussed.

```

z = [.5 11; 2 14; 1 12.1; 1.5 12.9; 0 10; 1 12; 2.4 14.9; 3 16]

z =
    0.5000    11.0000
    2.0000    14.0000
    1.0000    12.1000
    1.5000    12.9000
         0    10.0000
    1.0000    12.0000
    2.4000    14.9000
    3.0000    16.0000

t = z(:,1);
y = z(:,2);
[t,i] = sort(t)

t =
         0
    0.5000
    1.0000
    1.0000
    1.5000
    2.0000
    2.4000
    3.0000

i =
     5
     1
     3
     6
     4
     2
     7
     8

y = y(i);
z = [t y]

z =
         0    10.0000
    0.5000    11.0000
    1.0000    12.1000
    1.0000    12.0000
    1.5000    12.9000
    2.0000    14.0000
    2.4000    14.9000
    3.0000    16.0000

chkfdup

j =
     3

```



```

% the analyst decides to delete row 3 of z
z(3,:) = [];
% let t be the first column of the adjusted data set
t = z(:,1);
d = diff(t)

d =

    0.5000
    0.5000
    0.5000
    0.5000
    0.4000
    0.6000

p = ( ones(d)*d(1) ~= d)

p =

     0
     0
     0
     0
     1
     1

err = find(p)

err =

     5
     6

% the analyst observes non-uniform spacing in lines 5-6, and 6-7 of z
% re-index t with positive integers
tnew = (1:length(t))'

tnew =

     1
     2
     3
     4
     5
     6
     7

% reindex tnew with s = 0, T = .5
tnew = (0:.5:0+(length(tnew)-1)*(.5))'

tnew =

     0
    0.5000
    1.0000
    1.5000
    2.0000
    2.5000
    3.0000

```

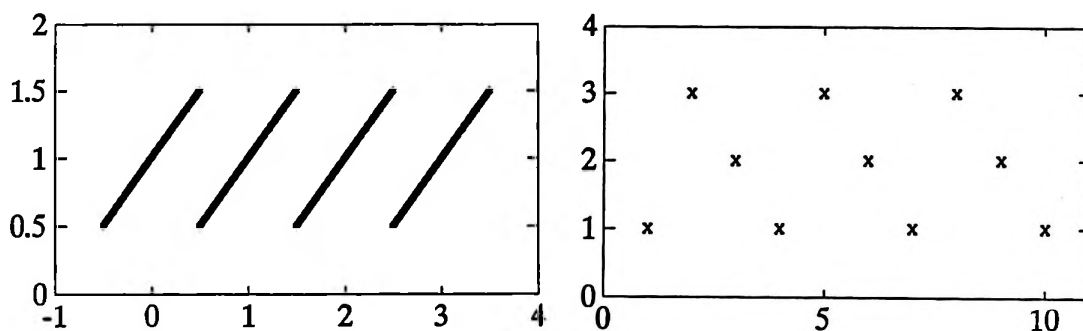
1.3 Periodicity

Introduction to Periodic Functions

Introduction

In this section, some basic properties of periodic functions are carefully established, from a viewpoint that favors investigation of periodicities in discrete-domain data. The next section addresses more delicate properties of periodic functions.

Roughly, a *periodic function* is one with values that repeat some basic pattern. The graphs of two periodic functions are shown below.



The usual definition of a periodic function is adapted, as follows, to better suit the investigation of periodicities in discrete-domain data:

DEFINITION

period p ;

periodic function

Let f be a real-valued function, with $\mathcal{D}(f) \subset \mathbf{R}$.

The function f has a *period* p , where p is a real number, if and only if the domain of f contains both $x + p$ and $x - p$ whenever it contains x , and if

$$f(x + p) = f(x - p) = f(x)$$

for all $x \in \mathcal{D}(f)$.

A function f is *periodic* if and only if it has a nonzero period.

$x \pm p$

The symbol ' \pm ' is read as 'plus or minus', and the abbreviation ' $x \pm p$ ' is used for ' $x + p$ or $x - p$ '. Similarly, ' $f(x \pm p)$ ' is shorthand for ' $f(x + p)$ or $f(x - p)$ '.

The requirement that *both* $x+p$ and $x-p$ be in the domain of f is essential. It assures that one can always move *both* to the right and to the left in the domain of f . Without this requirement, it is possible to have 'functions with a nonzero period' that no reasonable person would want to call periodic, as illustrated in a later example.

*some immediate
consequences of
the definition*

Here are some immediate consequences of the definition just given:

*every function
has period 0*

Every function has period 0, since $f(x \pm 0) = f(x)$ for all $x \in \mathcal{D}(f)$.

*symmetry in
the definition;
if f has a
nonzero period,
then it has
a positive period*

Whenever a function f has a period p , it also has a period $-p$. This is a consequence of the symmetry in the definition. Thus, if a function f has *any* nonzero period, then it necessarily has a *positive* period.

*a constant
function has
all periods*

A constant function $f: \mathbf{R} \rightarrow \mathbf{R}$ has *all* periods, since for all real numbers x and p , $f(x \pm p) = f(x)$.

EXAMPLE
*a function
with rational
number periods*

Define $g: \mathbf{R} \rightarrow \mathbf{R}$ by

$$g(x) = \begin{cases} 1 & \text{for } x \in \mathbf{Q} \\ 0 & \text{for } x \notin \mathbf{Q} \end{cases}.$$

Every rational number is a period of this function, and these are the only periods. To see this, argue as follows:

Let p be any rational number. Then,

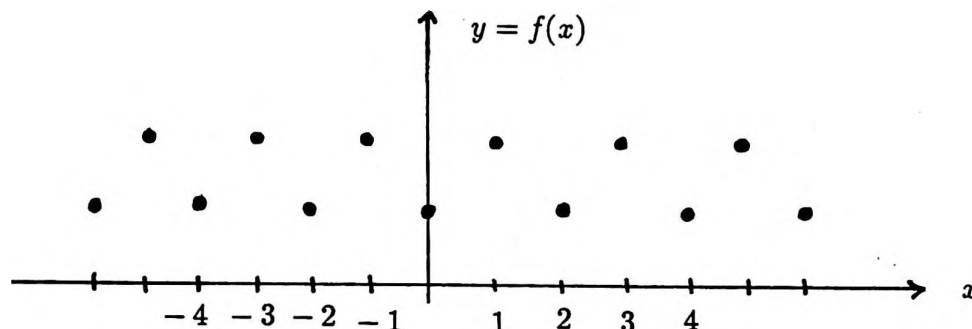
$$\begin{aligned} x \in \mathbf{Q} &\implies x \pm p \in \mathbf{Q} \implies 1 = g(x) = g(x \pm p); \text{ and} \\ x \notin \mathbf{Q} &\implies x \pm p \notin \mathbf{Q} \implies 0 = g(x) = g(x \pm p), \end{aligned}$$

which shows that every rational number is a period of g . The argument used the fact that a sum of rational numbers is rational; and the sum of a rational and irrational number is irrational.

To see that no irrational number is a period, let p be any irrational number, and choose $x := -p$. Thus, x is irrational, so $g(x) = 0$. However, $x + p = -p + p = 0$ is rational, so $g(x + p) = 1$. Thus, $g(x) = g(x + p)$ does *not* hold for all x , so the irrational number p is not a period of g .

EXAMPLE
a function with
period 2

The discrete-domain function f graphed below has period 2. Observe that it also has periods $2k$ for all integers k .



building new
periodic functions

The next result gives a way to build 'new' periodic functions from functions with a common domain and a common period.

LEMMA 1

sums,
scalar multiples,
and products of
functions with
a period p

Let f and g be functions with common domain \mathcal{D} and a common period p . Then the functions $f \pm g$, fg , and kf (for all real numbers k) also have a period p . If $g(x) \neq 0$ for all $x \in \mathcal{D}$, then $\frac{f}{g}$ has a period p .

PROOF

Recall that the functions $f \pm g$, fg , kf and $\frac{f}{g}$ are defined by:

$$(f \pm g)(x) := f(x) \pm g(x)$$

$$(fg)(x) := f(x) \cdot g(x)$$

$$(kf)(x) := k \cdot f(x)$$

$$\left(\frac{f}{g}\right)(x) := \frac{f(x)}{g(x)}, \quad g(x) \neq 0.$$

Now, for every $x \in \mathcal{D}$,

$$\begin{aligned} (f + g)(x \pm p) &:= f(x \pm p) + g(x \pm p) \\ &= f(x) + g(x) \\ &:= (f + g)(x). \end{aligned}$$

This shows that $f + g$ also has a period p . The remaining cases are similar. ■

*Lemma 1 holds
for all finite
sums and
products*

By induction, it follows easily that Lemma 1 also holds for all finite sums and products.

For example, suppose that functions f , g and h have common domain \mathcal{D} and a common period p . Then, one application of the lemma shows that $f+g$ has a period p ; a second application shows that $(f+g)+h$ has a period p .

If a pattern repeats itself on an interval of length p , then it necessarily repeats itself on intervals of length $2p, 3p, 4p, \dots$. This idea is formalized next.

LEMMA 2

*any multiple
of a period
is also a period*

If a function f has a period p , then it also has periods kp for $k \in \mathbb{Z}$. In particular, whenever $x \in \mathcal{D}(f)$, then so are $x + kp$, $k \in \mathbb{Z}$.

PROOF

*a typical
induction
argument*

The proof proceeds by induction. Let $S(k)$ be the statement

‘ f has a period kp ’.

The function f has period $0 \cdot p = 0$, since $f(x \pm 0) = f(x)$ for all $x \in \mathcal{D}(f)$. Thus, $S(0)$ is true. In what follows, it is shown that whenever $S(k)$ is true (for $k \geq 0$), then $S(k+1)$ is also true. Since $-p$ is a period of f whenever p is, this will complete the proof.

For induction, suppose that f has period kp (i.e., $S(k)$ is true). Then,

$$\begin{aligned} x \in \mathcal{D}(f) &\implies x \pm kp \in \mathcal{D}(f) && (f \text{ has a period } kp) \\ &\implies (x \pm kp) \pm p \in \mathcal{D}(f) && (f \text{ has a period } p) \end{aligned}$$

In particular, both $x + (k+1)p$ and $x - (k+1)p$ are in $\mathcal{D}(f)$. Furthermore,

$$\begin{aligned} f(x) &= f(x + kp) = f(x - kp) && (f \text{ has period } kp) \\ &= f((x + kp) + p) = f((x - kp) - p) && (f \text{ has a period } p) \\ &= f(x + (k+1)p) = f(x - (k+1)p) . \end{aligned}$$

It has been shown that whenever $x \in \mathcal{D}(f)$, so are $x \pm (k+1)p$, and $f(x) = f(x \pm (k+1)p)$. Thus, f has a period $(k+1)p$. Therefore, $S(k+1)$ is true. ■

LEMMA 3 Let f have a period p . Any two domain elements whose distance apart is $|p|$ must have identical function values.

if f has a period p ,
then domain
elements
 $|p|$ apart
have identical
function values

Whenever two domain elements are any multiple of $|p|$ apart, then they must have identical function values.

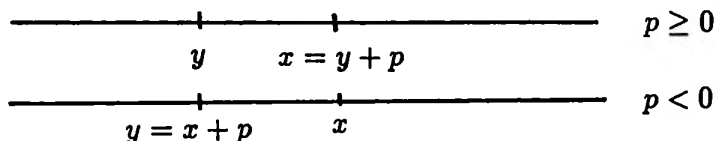
PROOF

Suppose that f has a period p . Let x and y be in the domain of f , with $|x - y| = |p|$. Switching names, if necessary, suppose that $x \geq y$.

If $p \geq 0$, then $x = y + p$, and so $f(x) = f(y + p) = f(y)$.

If $p < 0$, then $y = x + p$, and so $f(y) = f(x + p) = f(x)$.

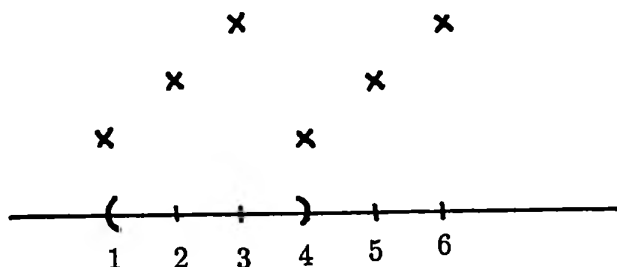
In either case, $f(x) = f(y)$.



By the previous lemma, f also has periods kp , for all $k \in \mathbb{Z}$. So if $|x - y| = |kp|$, then an application of the result just proved shows that $f(x) = f(y)$. ■

The next lemma shows that if a function has a positive period p , then on *any* half-open interval of length p , the function must take on *all* of its function values. Thus, there is no 'natural starting place' for the repeating pattern that the function exhibits.

The sketch below illustrates that a function with a positive period p may *not* take on all its function values on an *open* interval of length p . It is interesting to note that this is a 'fault' of the definition of the length of an interval; the intervals (a, b) , $[a, b]$, $[a, b)$ and $(a, b]$ all have the same length.



PERIOD 3; TAKE $I := (1, 4)$

LEMMA 4

Let f have a positive period p . On any half-open interval of length p , f must take on all the values in its range.

Let f have a positive period p . On any half-open interval of length p , f must take on all the values in its range.

On any interval that contains a half-open interval of length p , f must take on all its output values.

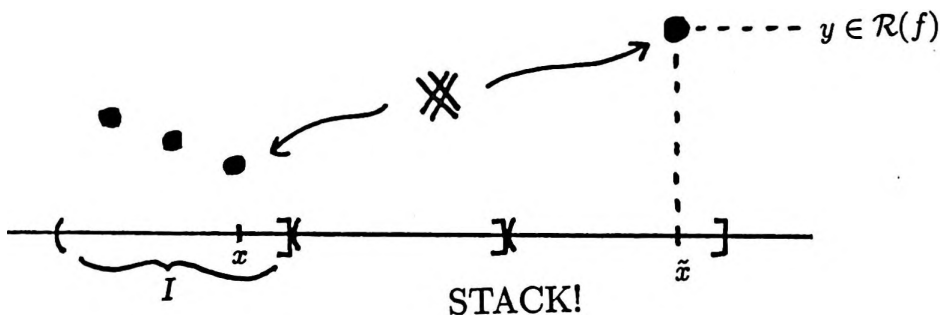
PROOF

Let f have a positive period p , and let I be any half-open interval of length p . Thus, I is of the form $(a, b]$ or $[a, b)$, where $b - a = p$.

Suppose, for contradiction, that there is an output of f which is *not* assumed in I ; that is, suppose there exists $y \in \mathcal{R}(f)$ for which there is *no* $x \in (\mathcal{D}(f) \cap I)$ with $f(x) = y$. Since $y \in \mathcal{R}(f)$, there must exist $\tilde{x} \in \mathcal{D}(f)$ with $y = f(\tilde{x})$. Write $\tilde{x} = x + np$ for some $x \in I$, $n \in \mathbb{Z}$. (To see that this is always possible, merely place enough intervals of length I end-to-end so that they eventually cover \tilde{x} . See the sketch below.) Since \tilde{x} is in the domain of f , and the distance from \tilde{x} to x is a multiple of p , x is also in $\mathcal{D}(f)$. Furthermore,

$$\begin{aligned} y &= f(\tilde{x}) = f(x + nP) \\ &= f(x), \end{aligned}$$

yielding the desired contradiction. ■



investigating a different definition for a periodic function

It is tempting to simplify the definition of a periodic function as follows:

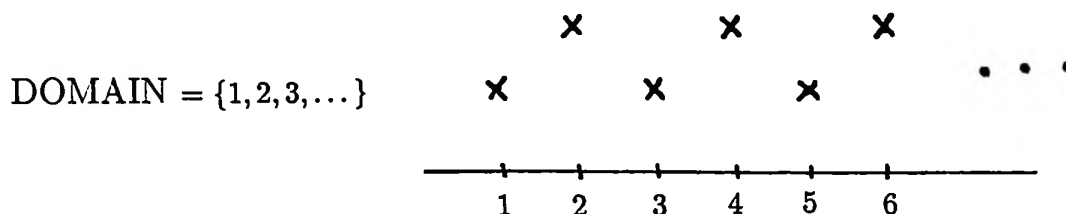
Proposed Definition: The function f has a period p if the domain of f contains $x + p$ whenever it contains x , and if

$$f(x + p) = f(x)$$

for all $x \in \mathcal{D}(f)$.

With this 'proposed definition', a function need not have a period $-p$ whenever it has a period p .

For example, the function graphed below has a period 2, but not a period -2 .



The problem with this 'proposed definition' is that it is possible to define functions with nonzero periods that no reasonable person would want to call periodic. This is illustrated in the next example.

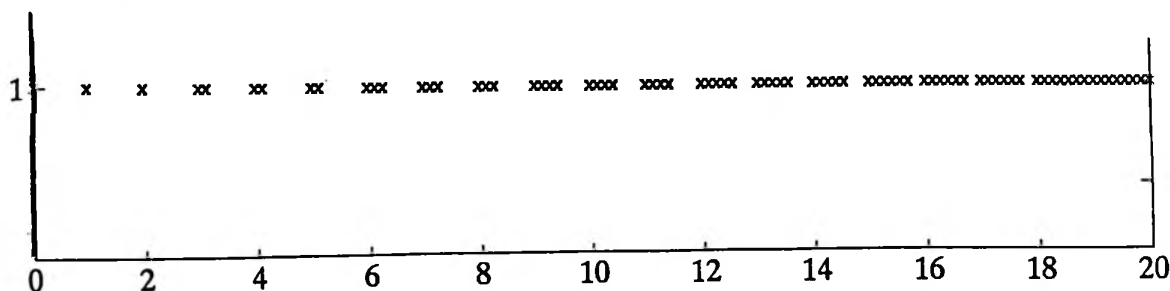
EXAMPLE

Define a function f as follows: let $0 \in \mathcal{D}(f)$, and require f to have periods 1 and π (as per the 'proposed definition'). Then, the numbers $1, 2, 3, 4, \dots$ must all be in $\mathcal{D}(f)$, as must $\pi, 2\pi, 3\pi, 4\pi, \dots$. Then, all elements of the form $k + n\pi$ must be in $\mathcal{D}(f)$, where k and n are nonnegative integers. Now, the set $\{k + n\pi \mid k \geq 0, n \geq 0\}$ is closed under addition, and forms $\mathcal{D}(f)$.

Define f to be 1 everywhere on its domain. A moment's reflection confirms that f does indeed have periods 1 and π .

The problem, however, is this: the graph of the function does not periodically repeat itself as one moves from left to right, since new domain elements are constantly being added! In particular, the graph of f does NOT repeat itself on intervals of length 1 or π , so most people would feel uncomfortable calling f 'periodic'. A partial graph of f is shown next.

A 'periodic'
function?



contents of
the next section

Thus far, this section has carefully established basic properties of functions with a period p . Many interesting questions arise when one considers the set of *all* periods of a function. Does every periodic function have a least positive period? (No.) If a periodic function does not have a least positive period, can anything be said about it? (Yes. The periods must be dense in \mathbb{R} , and the function must be either constant, or everywhere discontinuous.) Is the sum of two periodic functions (with different periods) necessarily periodic? (No.) If two functions have the same least positive period, must their sum have this same least positive period? (No.) These are some of the questions addressed in Section 1.4.

Some Basic Reshaping Techniques

This section closes with some interesting 'reshaping' results that can be used, in certain cases, to identify periodic components in a finite list of real numbers. The next few definitions and notation will considerably simplify the statement and proof of the 'reshaping' theorem.

DEFINITION <i>mean of a list</i>	Let $y = (y_1, \dots, y_N)$ be a finite list. The <i>mean of the list y</i> is the number μ_y defined by $\mu_y := \frac{1}{N}(y_1 + y_2 + \dots + y_N)$.
--	--

LEMMA 5 <i>the mean of a sum of lists is the sum of the means</i>	<p>Let x_1, x_2, \dots, x_M be lists, each of the same (finite) length, and let</p> $S = x_1 + x_2 + \dots + x_M .$ <p>Then,</p> $\mu_S = \mu_{x_1} + \mu_{x_2} + \dots + \mu_{x_M} .$
---	---

Thus, the mean of a sum of lists is the sum of the means of the component lists.

PROOF
of Lemma 5

First, it is shown that the result holds when S is generated by only two lists. Thus, suppose that

$$S = (x_1, \dots, x_N) + (y_1, \dots, y_N) = (x_1 + y_1, \dots, x_N + y_N) .$$

Then,

$$\begin{aligned} \mu_S &= \frac{1}{N}((x_1 + y_1) + \dots + (x_N + y_N)) \\ &= \frac{1}{N}(x_1 + \dots + x_N) + \frac{1}{N}(y_1 + \dots + y_N) = \mu_x + \mu_y . \end{aligned}$$

The remainder of the proof follows by induction. Suppose that the result holds for K lists, where $K \geq 2$ is a fixed integer, and suppose that S is a sum of $K + 1$ lists, denoted by x_1, \dots, x_{K+1} . Write

$$S = \overbrace{x_1 + \dots + x_K} + \overbrace{x_{K+1}},$$

so that S is viewed as a sum of two lists. Then,

$$\begin{aligned}\mu_S &= \mu_{x_1 + \dots + x_K} + \mu_{x_{K+1}} \\ &= (\mu_{x_1} + \dots + \mu_{x_K}) + \mu_{x_{K+1}},\end{aligned}$$

where the inductive hypothesis was used in the second step. ■

DEFINITION
cycle

Let p be a positive integer. The phrase ' p -cycle' is used to denote any list of the form

$$\overbrace{(x_1, x_2, \dots, x_p)}^{1^{\text{st}} \text{ cycle}}, \overbrace{(x_1, x_2, \dots, x_p)}^{2^{\text{nd}} \text{ cycle}}, \dots, \overbrace{(x_1, x_2, \dots, x_p)}^{r^{\text{th}} \text{ cycle}};$$

that is, a p -cycle is any list composed of p numbers in a specified order, that are repeated r times, where r is a positive integer.

The length of a p -cycle is necessarily a multiple of p .

For positive integers p and q , a p -cycle and a q -cycle are called *relatively prime* if p and q are relatively prime; that is, if p and q have no common factors other than 1.

NOTATION
for cycles

Let r be a positive integer, and let $x = (x_1, \dots, x_p)$. The notation $r x$ is used for the p -cycle formed from r repetitions of x :

$$r x := \overbrace{(x_1, \dots, x_p)}^{1^{\text{st}} \text{ cycle}}, \overbrace{(x_1, \dots, x_p)}^{2^{\text{nd}} \text{ cycle}}, \dots, \overbrace{(x_1, \dots, x_p)}^{r^{\text{th}} \text{ cycle}}.$$

$$\mu(r x) = \mu_x$$

Observe that the mean of $x = (x_1, \dots, x_p)$ is equal to the mean of any p -cycle $r x$, since

$$\mu_x = \frac{1}{p}(x_1 + \dots + x_p)$$

and

$$\mu(r x) = \frac{1}{rp}(r(x_1 + \dots + x_p)).$$

EXAMPLE

Suppose that a list S of length 6 is a sum of a 2-cycle and a 3-cycle, say

$$S = (\overbrace{2, 5}, \overbrace{2, 5}, \overbrace{2, 5}) + (\overbrace{3, 0, -1}, \overbrace{3, 0, -1}) = (5, 5, 1, 8, 2, 4) .$$

There are of course an infinite number of 'similar' cycles that could have summed to yield S : given any constant K ,

$$\begin{aligned} S = & (2 + K, 5 + K, 2 + K, 5 + K, 2 + K, 5 + K) \\ & + (3 - K, 0 - K, -1 - K, 3 - K, 0 - K, -1 - K) . \end{aligned}$$

That is, one component can be shifted any given amount K , provided that the remaining component is shifted the opposite amount, $-K$. Thus, given *only* the sum list $S = (5, 5, 1, 8, 2, 4)$, it is impossible to recover the precise cycles that originally generated S . However, it *is* possible in this case to recover zero-mean 2 and 3-cycles which, when added to μ_S , yield S . Only some basic reshaping techniques are needed. The technique is presented in the proof of the next theorem.

THEOREM
*identifying
 periodic
 components
 in a finite list*

Let p and q be relatively prime positive integers; that is, p and q have no common factors other than 1. Let S be a list of length N , where N is a multiple of both p and q (and hence, N is a multiple of pq). Suppose that S is a sum of a p -cycle and a q -cycle, i.e.,

$$\begin{aligned} S = & (\overbrace{x_1, \dots, x_p}, \overbrace{x_1, \dots, x_p}, \dots, \overbrace{x_1, \dots, x_p}) \\ & + (\overbrace{y_1, \dots, y_q}, \overbrace{y_1, \dots, y_q}, \overbrace{y_1, \dots, y_q}, \dots, \overbrace{y_1, \dots, y_q}) . \end{aligned}$$

By letting

$$\mathbf{x} := (x_1, \dots, x_p) \text{ and } \mathbf{y} := (y_1, \dots, y_q) ,$$

and using cycle notation, S can be written more compactly as

$$S = (N/p)\mathbf{x} + (N/q)\mathbf{y} .$$

Then, given *only* the sum S (and *not* the component lists), there is a constructive method for finding zero-mean lists

$$\tilde{\mathbf{x}}_0 := (\tilde{x}_1, \dots, \tilde{x}_p) \text{ and } \tilde{\mathbf{y}}_0 := (\tilde{y}_1, \dots, \tilde{y}_q)$$

such that

$$S = (N/p)\tilde{\mathbf{x}}_0 + (N/q)\tilde{\mathbf{y}}_0 + \mathbf{M}_S ,$$

where \mathbf{M}_S is the list of length N having every entry equal to μ_S .

PROOF
producing the
lists \bar{x}_0 and \bar{y}_0

The proof illustrates the procedure that yields the zero-mean lists \bar{x}_0 and \bar{y}_0 .

Renaming, if necessary, suppose that $p > q$. In the following summation of the two components forming S , the notation $y_?$ is used to indicate that the exact subscript on y depends on the relationship between p and q :

$$\begin{aligned} S &= (\quad x_1, \quad x_2, \dots, \quad x_q, \quad x_{q+1}, \dots, \quad x_p, \quad x_1, \dots, \quad x_p) \\ &+ (\quad y_1, \quad y_2, \dots, \quad y_q, \quad y_1, \dots, \quad y_?, \quad y_?, \dots, \quad y_q) \\ &= (x_1 + y_1, x_2 + y_2, \dots, x_q + y_q, x_{q+1} + y_1, \dots, x_p + y_?, x_1 + y_?, \dots, x_p + y_q) . \end{aligned}$$

Subtract μ_S from each entry in S , and call the resulting list S_0 (since S_0 has mean 0). To find \bar{x}_0 , first reshape S_0 in rows of length p :

$$\overbrace{(x_1 + y_1 - \mu_S, x_2 + y_2 - \mu_S, \dots, x_q + y_q - \mu_S, \dots, x_p + y_? - \mu_S, x_1 + y_? - \mu_S, \dots)}^{\text{first row}},$$

to get

$$\begin{array}{cccc} x_1 + y_1 - \mu_S & x_2 + y_2 - \mu_S & \cdots & x_p + y_? - \mu_S \\ x_1 + y_? - \mu_S & x_2 + y_? - \mu_S & \cdots & x_p + y_? - \mu_S \\ \vdots & \vdots & \vdots & \vdots \\ x_1 + y_? - \mu_S & x_2 + y_? - \mu_S & \cdots & x_p + y_q - \mu_S \end{array}$$

There are $\frac{N}{p}$ rows in the arrangement above. Summing the entries in each of the p columns gives the column sums

$$\left(\frac{N}{p}(x_1 - \mu_S) + \sum_{\text{col } 1} y_i \right) \quad \left(\frac{N}{p}(x_2 - \mu_S) + \sum_{\text{col } 2} y_i \right) \quad \cdots \quad \left(\frac{N}{p}(x_p - \mu_S) + \sum_{\text{col } p} y_i \right) .$$

There are $\frac{N}{q}$ sets of (y_1, \dots, y_q) in the entire list, and these are equally divided among the p columns, since p and q are relatively prime. Thus, each column contains $\frac{N}{pq}$ sets of (y_1, \dots, y_q) , and hence

$$\sum_{\text{col } i} y_i = \frac{N}{pq}(y_1 + \cdots + y_q) = \frac{N}{p}\mu_y, \quad i = 1, \dots, p .$$

Dividing each column sum by $\frac{N}{p}$ gives the averages of each column:

$$\underbrace{x_1 - \mu_S + \mu_y}_{\text{1st column average}} \quad \underbrace{x_2 - \mu_S + \mu_y}_{\text{2nd column average}} \quad \cdots \quad \underbrace{x_p - \mu_S + \mu_y}_{\text{last column average}}$$

Since $\mu_S = \mu_x + \mu_y$, these column averages can be written more simply as

$$\underbrace{x_1 - \mu_x}_{\text{1st column average}} \quad \underbrace{x_2 - \mu_x}_{\text{2nd column average}} \quad \cdots \quad \underbrace{x_p - \mu_x}_{\text{last column average}} .$$

Therefore, the column averages of the reshaped list recover the list \mathbf{x} , with each entry decreased by $\mu_{\mathbf{x}}$. Define $\tilde{\mathbf{x}}_0$ to be the list consisting of these column averages:

$$\tilde{\mathbf{x}}_0 := (x_1 - \mu_{\mathbf{x}}, x_2 - \mu_{\mathbf{x}}, \dots, x_p - \mu_{\mathbf{x}}) .$$

Clearly, $\tilde{\mathbf{x}}_0$ has zero mean.

A similar reshaping of \mathbf{S}_0 into $\frac{N}{q}$ rows of q each yields the column averages

$$\tilde{\mathbf{y}}_0 := (y_1 - \mu_{\mathbf{y}}, y_2 - \mu_{\mathbf{y}}, \dots, y_q - \mu_{\mathbf{y}}) .$$

Then,

$$\begin{aligned} (N/p)\tilde{\mathbf{x}}_0 + (N/q)\tilde{\mathbf{y}}_0 &= (N/p)\mathbf{x} - \mathbf{M}_{\mathbf{x}} + (N/q)\mathbf{y} - \mathbf{M}_{\mathbf{y}} \\ &= \mathbf{S} - \mathbf{M}_{\mathbf{S}} , \end{aligned}$$

from which

$$\mathbf{S} = (N/p)\tilde{\mathbf{x}}_0 + (N/q)\tilde{\mathbf{y}}_0 + \mathbf{M}_{\mathbf{S}} . \blacksquare$$

The theorem does not hold if p and q are not relatively prime, even if N is a multiple of pq . Also, the theorem does not hold if p and q are relatively prime, but N is not a multiple of pq .

*extending the
previous theorem*

The constructive technique just discussed also works if \mathbf{S} is a sum of any finite number of relatively prime cycles, providing that N is a multiple of the product of the cycles.

To see this, suppose that the theorem holds for K cycles, and suppose that

$$\mathbf{S} = \overbrace{\mathbf{C}_1 + \dots + \mathbf{C}_K}^{:=\mathbf{C}} + \overbrace{\mathbf{C}_{K+1}} ,$$

where \mathbf{C}_i is a p_i -cycle.

The list \mathbf{C} is a $p_1 p_2 \dots p_K$ -cycle, which is relatively prime to p_{K+1} , by hypothesis. Thus, applying the theorem to \mathbf{C} and \mathbf{C}_{K+1} yields

$$\mathbf{S} = \tilde{\mathbf{C}} + \tilde{\mathbf{C}}_{K+1} + \mu_{\mathbf{S}} .$$

Then, using the inductive hypothesis on $\tilde{\mathbf{C}}$, and the fact that $\mu_{\tilde{\mathbf{C}}} = 0$, one obtains

$$\mathbf{S} = \tilde{\mathbf{C}}_1 + \dots + \tilde{\mathbf{C}}_{K+1} + \mu_{\mathbf{S}} .$$

uniqueness of
decomposition
into
relatively prime
cycles

In general, a positive integer N may be viewed as a product of relatively prime integers in more than one way. For example, $30 = 2 \cdot 15 = 3 \cdot 10 = 5 \cdot 6$. Thus, given a list S of length 30, one might separately investigate the hypotheses that:

- S is a sum of a 2-cycle and a 15-cycle;
- S is a sum of a 3-cycle and a 10-cycle;
- S is a sum of a 5-cycle and a 6-cycle.

Under what conditions will N have a *unique* decomposition as a product of relatively prime integers? The next proposition answers this question.

PROPOSITION Let N be a positive integer. Then, N has a unique representation as a product of relatively prime integers if and only if the prime factorization of N is of the form $p^m q^n$, where p and q are prime with $p \neq q$, and m and n are positive integers.

PROOF

“ \Leftarrow ”

Suppose that the prime decomposition of N is of the form $N = p^m q^n$ for primes $p \neq q$, m and n positive integers. Then, p^m and q^n are relatively prime integers with product N . Any other regrouping of the prime factors as a product of two numbers is necessarily of the form $N = d_1 d_2$, where either d_1 and d_2 have a common factor p ; or d_1 and d_2 have a common factor q .

“ \Rightarrow ”

Suppose that N has a unique representation as a product of relatively prime integers. If N has exactly one distinct prime in its prime decomposition, say $N = p^m$ for a prime p and positive integer m , then N cannot be written as a product of relatively prime integers. If N has three or more primes in its prime decomposition, say $N = p^m q^n r^j x$ where p , q and r are distinct primes, m , n and j are positive integers, and x is either 1, or a product of primes other than p , q and r , then N can be written as a product of two relatively prime integers in more than one way: say,

$$\begin{aligned} N &= (p^m) \cdot (q^n r^j x) \\ &= (q^n) \cdot (p^m r^j x). \end{aligned}$$

Thus, N must have precisely two distinct primes in its prime factorization. ■

MATLAB EXAMPLE

The following diary of an actual MATLAB session illustrates the reshaping procedure, while reviewing the necessary MATLAB commands.

The illustrated procedure could be implemented much more efficiently. However, efficient implementations often obscure simple underlying ideas. Therefore, in what follows, efficiency has been sacrificed for the sake of clarity.

```
% construct a 'known unknown': a 2-cycle, 3-cycle and 5-cycle
x = [1 2];
x = [x x x x x x x x x x x x x x x];
% x is a 2-cycle of length 30
y = [-1 0 4];
y = [y y y y y y y y y y];
% y is a 3-cycle of length 30
z = [-1 3 1 2 0];
z = [z z z z z z z z z z];
% z is a 5-cycle of length 30
% sum the components to get the 'known unknown'
S = x + y + z;
% subtract off the mean of S to get S0
S0 = S - mean(S);
% reshape S0 to test for a 2-cycle, using the 'reshape' command
r1 = reshape(S0,2,15)

r1 =

-4.5000    1.5000
 2.5000   -0.5000
-2.5000    1.5000
-0.5000   -0.5000
 3.5000   -2.5000
-3.5000    5.5000
-2.5000    0.5000
 1.5000   -3.5000
 0.5000    3.5000
-1.5000   -1.5000
 0.5000    0.5000
-1.5000    4.5000
-3.5000   -2.5000
 4.5000   -1.5000
-0.5000    2.5000

% average the columns, using the 'sum' command, dividing by # of rows
r1 = sum(r1) / 15

r1 =

-0.5000    0.5000

% Observe that this is precisely x - mean(x)!
temp = x - mean(x);
temp(1:2)

ans =

-0.5000    0.5000

% Now, repeat, testing for the 3-cycle and 5-cycle
r2 = reshape(S0,3,10);
r2 = sum(r2) / 10
```

```

r2 =
    -2    -1     3
r3 = reshape(S0,5,6)';

r3 = sum(r3) / 6
r3 =
    -2     2     0     1    -1

% Now, build r1, r2, and r3 into length-30 cycles
r1 = [r1 r1 r1 r1 r1 r1 r1 r1 r1 r1 r1 r1 r1 r1 r1];
r2 = [r2 r2 r2 r2 r2 r2 r2 r2 r2 r2 r2 r2 r2 r2 r2];
r3 = [r3 r3 r3 r3 r3 r3 r3 r3 r3 r3 r3 r3 r3 r3 r3];

% Sum r1, r2 and r3, and add mean(S) back in, to recover S:
predict = r1 + r2 + r3 + mean(S)

predict =
Columns 1 through 12
    -1     5     6     3     1     5     3     3     7     1     0     9
Columns 13 through 24
     1     4     5     0     4     7     2     2     4     4     2     8
Columns 25 through 30
     0     1     8     2     3     6
% Compare with S:
S - predict

ans =
Columns 1 through 12
     0     0     0     0     0     0     0     0     0     0     0     0
Columns 13 through 24
     0     0     0     0     0     0     0     0     0     0     0     0
Columns 25 through 30
     0     0     0     0     0     0

```


1.4 More on Periodic Functions

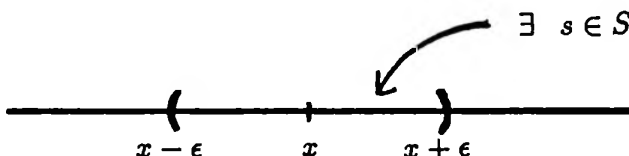
P ;
the set of
all periods

Let P denote the set of *all* periods of a real-valued function f defined on some subset of \mathbf{R} . If f has *any* nonzero period p , then Lemma 2 of Section 1.3 shows that P contains all integer multiples of p . Can there be anything else in P ? More generally, what can P (as a subset of \mathbf{R}) look like? This is the first question to be addressed in this section. First, a preliminary definition:

DEFINITION
dense in \mathbf{R}

A set S is *dense in \mathbf{R}* if there are elements of S arbitrarily close to *any* real number. Precisely, S is dense in \mathbf{R} if and only if for every $x \in \mathbf{R}$ and for every $\epsilon > 0$, there exists $s \in S$ with $|x - s| < \epsilon$.

The sketch below suggests a geometric view of this definition.



What can P ,
as a subset of \mathbf{R} ,
look like?

The next theorem completely characterizes what the set of all periods of a function can look like.

THEOREM 1
characterizing P

[adapted from Olm, p. 549] Let f be a real-valued function with $\mathcal{D}(f) \subset \mathbf{R}$, and let P be the set of all periods of f . The set P satisfies exactly one of the following properties:

- (a) $P = \{0\}$; in this case, f is not periodic.
- (b) P consists of all integral multiples of some least positive period P , also called the *fundamental period* of f .
- (c) P is dense in \mathbf{R} ; in this case, $\mathcal{D}(f)$ is also dense in \mathbf{R} , and there is no least positive period.

The next lemma is used in the proof of Theorem 1.

LEMMA 1
closure in P

The set P of all periods of a function f is closed under addition, subtraction, and multiplication by an arbitrary integer.

PROOF
of Lemma 1

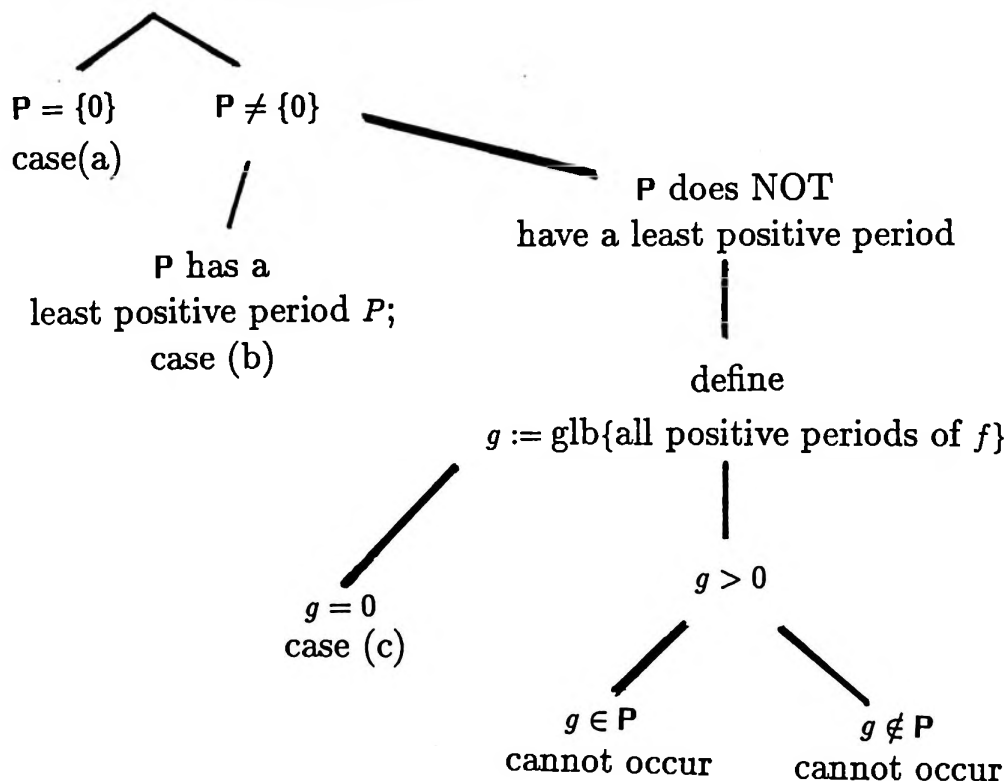
Let p_1 and p_2 be in \mathbf{P} , and let $x \in \mathcal{D}(f)$. Then, $x + p_1 \in \mathcal{D}(f)$, so also $(x + p_1) + p_2 \in \mathcal{D}(f)$. Furthermore,

$$\begin{aligned} f(x + (p_1 + p_2)) &= f((x + p_1) + p_2) \\ &= f(x + p_1) && (p_2 \text{ is a period}) \\ &= f(x) && (p_1 \text{ is a period}) \end{aligned}$$

This implies that $p_1 + p_2$ is a period of f , and thus \mathbf{P} is closed under addition. The fact that \mathbf{P} is closed under multiplication by an arbitrary integer follows from Lemma 2 of Section 1.3. In particular, $-p_2$ is a period of f , so that closure under addition implies that $p_1 + (-p_2) = p_1 - p_2$ is also a period of f . Thus, \mathbf{P} is also closed under subtraction. ■

Proof Structure
of Theorem 1

The logical structure of Theorem 1's proof is summarized in the flow chart below.



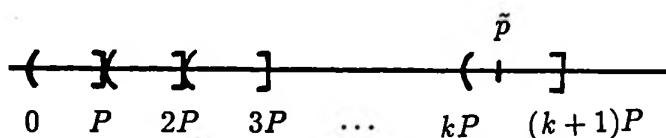
PROOF
of Theorem 1

The set \mathbf{P} always contains 0, since 0 is a period of every function. If 0 is the *only* element in \mathbf{P} , then f has no nonzero period, and is not periodic. This is case (a).

If $\mathbf{P} \neq \{0\}$, then there exists $p \neq 0$ in \mathbf{P} . Since both p and $-p$ are periods of f , it can be assumed that p is positive. Either \mathbf{P} has a least positive period, or not. If so, let P denote the least positive period, and let

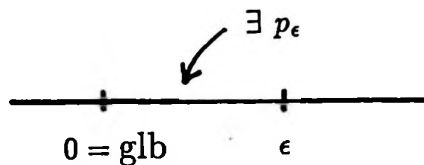
$$B := \{nP \mid n \in \mathbf{Z}\},$$

so that B consists of all integer multiples of P . Since every integer multiple of P is a period of f , it is always true that $B \subset \mathbf{P}$. Suppose for contradiction that $B \subsetneq \mathbf{P}$, so that there exists $\tilde{p} \in \mathbf{P}$ with $\tilde{p} \notin B$. Without loss of generality, suppose that $\tilde{p} > 0$. Covering \mathbf{R} by intervals of the form $(nP, (n+1)P]$, $n \in \mathbf{Z}$,



it must be that $\tilde{p} \in (kP, (k+1)P]$ for some positive integer k . But then, $\tilde{p} - kP \in (0, P]$, so that (via Lemma 1) $\tilde{p} - kP$ is a strictly positive period of f that is less than P ; a contradiction. Thus, $B = \mathbf{P}$; this is case (b).

It remains only to show that if \mathbf{P} does *not* have a least positive period, then \mathbf{P} is dense in \mathbf{R} . Define g to be the greatest lower bound of all positive periods in \mathbf{P} . Either $g = 0$, or $g > 0$. If $g = 0$, then there exist arbitrarily small positive periods. In particular, given any $\epsilon > 0$, there exists a positive period p_ϵ with $p_\epsilon < \epsilon$.



Then,

$$\{kp_\epsilon \mid k \in \mathbf{Z}\} \subset \mathbf{P},$$

and

$$\{x + kp_\epsilon \mid x \in \mathcal{D}(f), k \in \mathbf{Z}\} \subset \mathcal{D}(f).$$

Let y be any real number, and cover \mathbf{R} by intervals of the form $(kp_\epsilon, (k+1)p_\epsilon]$ for $k \in \mathbf{Z}$. Thus, there exists an integer j with

$$y \in (jp_\epsilon, (j+1)p_\epsilon] .$$

Since the interval $(jp_\epsilon, (j+1)p_\epsilon]$ has length p_ϵ which is less than ϵ , the distance from y to $(j+1)p_\epsilon$ is less than ϵ . Hence, $(j+1)p_\epsilon$ is an element of \mathbf{P} within ϵ of y , from which it is concluded that \mathbf{P} is dense in \mathbf{R} .

To see that $\mathcal{D}(f)$ is dense in \mathbf{R} , repeat the previous argument, this time covering \mathbf{R} by intervals of the form

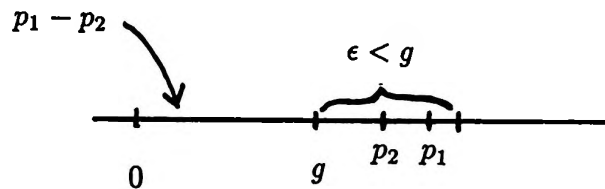
$$(x + kp_\epsilon, x + (k+1)p_\epsilon] ,$$

where x is a fixed element of $\mathcal{D}(f)$.

Thus, both \mathbf{P} and $\mathcal{D}(f)$ are dense in \mathbf{R} .

It is shown next that $g > 0$ cannot occur. For if $g > 0$, then either $g \in \mathbf{P}$ or $g \notin \mathbf{P}$. If $g \in \mathbf{P}$, then g would be a least positive period; but the current assumption is that \mathbf{P} has no least positive period.

Thus, it must be that $g \notin \mathbf{P}$. Then, there exist positive periods arbitrarily close to g . In particular, given ϵ with $0 < \epsilon < g$, there exist distinct positive periods p_1 and p_2 with $0 < p_1 - p_2 < \epsilon < g$.



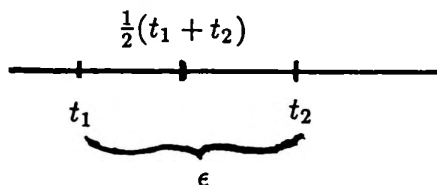
Via Lemma 1, $p_1 - p_2$ is a period of f , and $p_1 - p_2$ is strictly less than g ; contradicting the fact that g is the greatest lower bound. Case (c) has now been proven. ■

A useful consequence of Theorem 1 is:

COROLLARY 1 Every discrete-domain periodic function has a least positive period.

PROOF
of Corollary 1

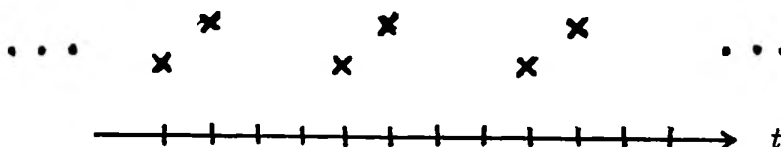
Let f be a discrete-domain periodic function. By definition of a discrete-domain function, the elements of $\mathcal{D}(f)$ come from a time list. No time list can be dense in \mathbb{R} . To see that this is the case, let t_1 and t_2 be two consecutive entries in the time list, so that $t_1 < t_2$. Define $\epsilon := t_2 - t_1$, so that $\epsilon > 0$. Then, the midpoint $t := \frac{t_1 + t_2}{2}$ is an element of \mathbb{R} with no time list entry within, say, $\frac{\epsilon}{3}$ of t .



Thus, the domain of a discrete-domain function cannot be dense in \mathbb{R} , which excludes case (c) of Theorem 1. Since f is periodic, case (a) does not hold. Thus, case (b) holds, and f has a least positive period. ■

*the domain of
a discrete-domain
periodic function
need not be
a uniform time list*

It is important to note that the domain of a discrete-domain periodic function need *not* be a *uniform* time list, as the example below illustrates.



LEAST POSITIVE PERIOD = 4

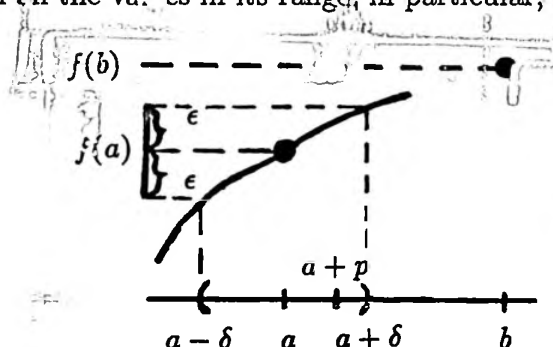
To avoid topological digressions, any discussion of *continuity* of periodic functions is restricted to periodic functions with domain \mathbb{R} (having its usual topology). The next theorem characterizes periodic functions defined on \mathbb{R} that have no least positive period.

THEOREM 2	Let $f: \mathbb{R} \rightarrow \mathbb{R}$ be a periodic function with no least positive period. Then f is either constant, or everywhere discontinuous.
------------------	--

PROOF
of Theorem 2

By Theorem 1, the set P of all periods of f is dense in \mathbb{R} . To show that f is either constant or everywhere discontinuous, it is shown, equivalently, that if f is *not* everywhere discontinuous, then it is constant.

So suppose that f is continuous at $x = a$, and suppose for contradiction that f is not constant. Then, there exists b with $f(b) \neq f(a)$. Choose ϵ with $0 < \epsilon < |f(b) - f(a)|$. By continuity of f at a , there exists $\delta > 0$ such that whenever $|x - a| < \delta$, one has $|f(x) - f(a)| < \epsilon$. Consequently, whenever x is within the δ -interval about a , $f(x) \neq f(b)$. However, since the periods of f are dense in \mathbb{R} , there exists a period p with $0 < p < \delta$. On the interval $[a, a + p] \subset [a, a + \delta)$, f must take on all the values in its range, in particular, $f(b)$.



This yields the desired contradiction. ■

a summary

To summarize: by Corollary 1, every discrete-domain periodic function has a least positive period and, by Theorem 2, if a periodic function defined on \mathbb{R} has *no* least positive period, then it is either 'uninteresting' (constant), or 'very unusual' (everywhere discontinuous).

To avoid the problems associated with periodic functions having no least positive period, in the remainder of this text, it is assumed that all periodic functions have a least positive (fundamental) period.

*sums of
periodic functions*

The remainder of this section investigates sums of periodic functions, where each summand has a fundamental period. The next definition provides a method of comparing real numbers that is very important in this context.

DEFINITION
*commensurable
numbers*

Two nonzero real numbers p and q are *commensurable* if and only if the ratio $\frac{p}{q}$ is a rational number.

A finite set of nonzero real numbers $\{p_1, p_2, \dots, p_n\}$ is *commensurable* if and only if the ratios $\frac{p_i}{p_j}$ are rational numbers, for all $i, j \in \{1, 2, \dots, n\}$.

EXAMPLES Every pair of nonzero rational numbers is commensurable, as is every finite set of nonzero rational numbers.

The numbers $\sqrt{2}$ and $3\sqrt{2}$ are commensurable.

The numbers 2 and π are not commensurable. Indeed, any finite set that contains at least one rational number and at least one irrational number is not commensurable.

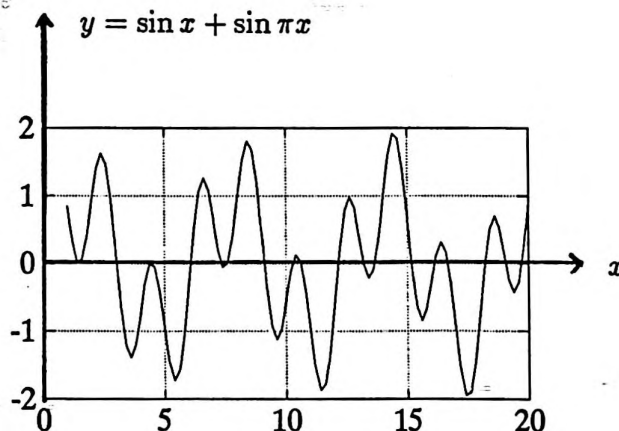
The questions addressed next arise naturally when investigating sums of periodic functions.

QUESTION 1: QUESTION: Must a sum of periodic functions be periodic?

ANSWER: No. The classic counterexample is provided by the sum

$$\sin x + \sin \pi x .$$

Here, $\sin x$ and $\sin \pi x$ have fundamental periods 2π and 2, respectively, but the sum is not periodic. (The proof is a simple consequence of Theorem 5 or Theorem 6 in [O&T].) Note that the fundamental periods of the summands are *not* commensurable in this case.



★

Computers cannot represent irrational numbers. Therefore, the 'computer' rendition of the sum $\sin x + \sin \pi x$ is necessarily a sum of sinusoids with commensurable periods. The following results will show that the 'computer' sum is indeed periodic, but with a very large period.

It will be shown in Theorem 3 that a sum of functions with commensurable least positive periods must be periodic. However, the next example shows that, even in this case, the sum need *not* have a least positive period.

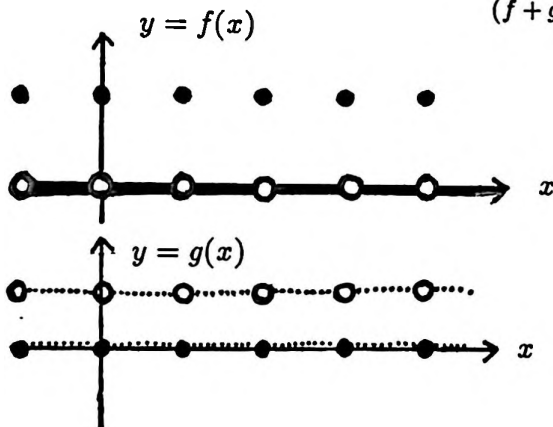
QUESTION 2: QUESTION: If a sum of functions with least positive periods is periodic, must the sum have a least positive period?

ANSWER: [C&P, p. 33] No. The functions f and g defined below each have fundamental period 1. Their sum $f + g$ is periodic, but (as shown in Section 1.3) has no least positive period.

$$f(x) = \begin{cases} 1 & \text{for } x \in \mathbb{Z} \\ 0 & \text{for } x \notin \mathbb{Z} \end{cases}$$

$$g(x) = \begin{cases} 1 & \text{for } x \in \mathbb{Q} \text{ and } x \notin \mathbb{Z} \\ 0 & \text{for } x \notin \mathbb{Q} \text{ or } x \in \mathbb{Z} \end{cases}$$

$$(f+g)(x) = \begin{cases} 1 & \text{for } x \in \mathbb{Q} \\ 0 & \text{for } x \notin \mathbb{Q} \end{cases}$$



QUESTION 3: QUESTION: Suppose that two functions have the same least positive period, and their sum has a least positive period. Must the sum have the *same* least positive period as the summands?

ANSWER: No. Consider functions f and g with common domain \mathbb{Z} , and with output lists given below. Both f and g have fundamental period 4, but their sum has fundamental period 2. For an example using continuous summands, see [C&P, p. 32].

$$f : (\dots, \overbrace{2, 0, -2, 0}, 2, 0, -2, 0, \dots)$$

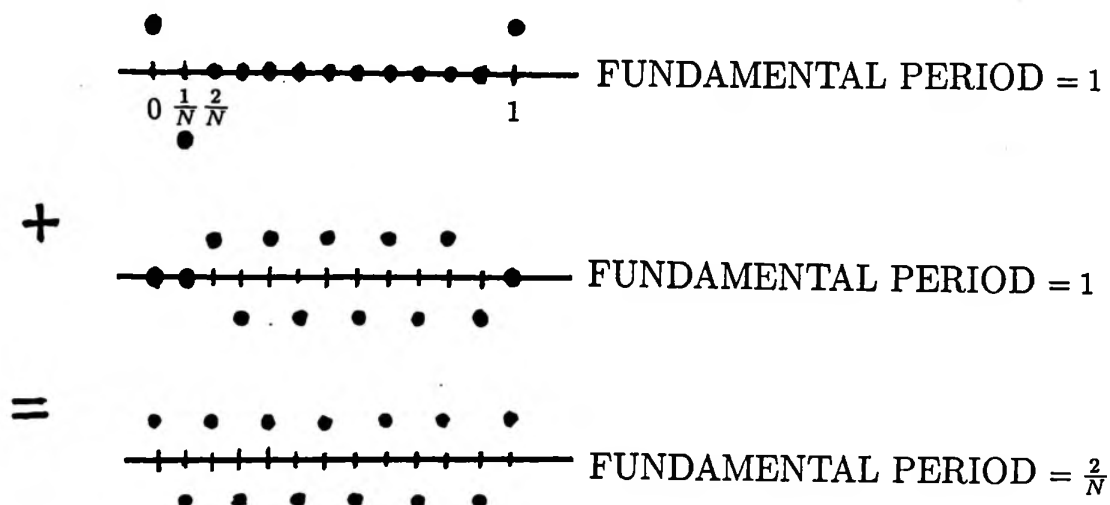
$$g : (\dots, \overbrace{-1, 0, 3, 0}, -1, 0, 3, 0, \dots)$$

$$f + g : (\dots, \overbrace{1, 0}, 1, 0, 1, 0, \dots)$$

the fundamental
period
can be made
as small
as desired

This last example can be further generalized. Indeed, the fundamental period of the sum can be made *as small as desired*, without changing the fundamental period of the summands, as illustrated by the output lists given below. For an example using continuous summands, see [C&P, p. 33].

Theorem 3 addresses the question: What are the possible *candidates* for the fundamental period of a sum?



The next theorem assures that a sum of periodic functions with commensurable least positive periods is itself periodic:

THEOREM 3 If functions f_1, f_2, \dots, f_N have a common domain \mathcal{D} and commensurable least positive periods P_1, P_2, \dots, P_N , then the sum $f_1 + \dots + f_N$ is periodic.

The proof of Theorem 3 makes use of the next three lemmas.

LEMMA 2 If p and q are commensurable, then there exist integers n and d ($d \neq 0$) with $p = (\frac{n}{d})q$.

PROOF
of Lemma 2

By definition of commensurable numbers, the ratio $\frac{p}{q}$ is rational. Thus, there exist integers n and d with $\frac{p}{q} = \frac{n}{d}$, from which $p = (\frac{n}{d})q$. ■

LEMMA 3

*scaling
periodic
functions*

Let s be a fixed positive real number.

A function f has fundamental period P if and only if the function g defined by

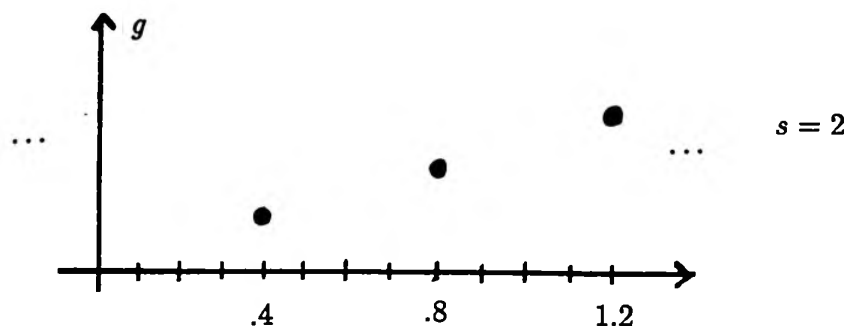
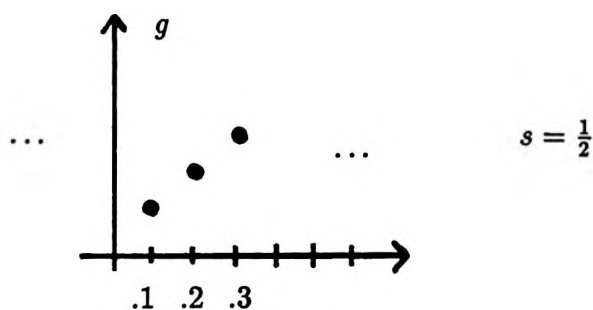
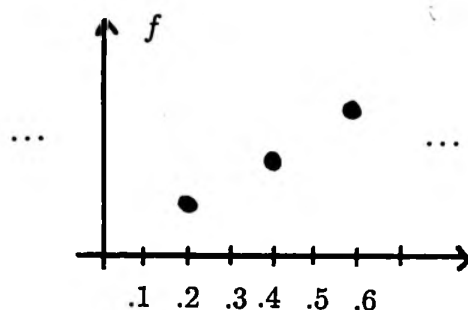
$$g(xs) := f(x) \text{ for all } x \in \mathcal{D}(f)$$

has fundamental period sP .

*the function f ,
with domain
scaled by s*

The function g in this lemma is referred to as '*the function f , with domain scaled by s* '.

A periodic function f and 'scaled' functions g are shown below, for two different choices of the scaling factor s .



PROOF
of Lemma 3

[adapted from C&P, p. 35]

" \Rightarrow " Let f have least positive period P . For a given positive real number s , the domain of g is:

$$\mathcal{D}(g) = \{y \in \mathbb{R} \mid y = xs \text{ for some } x \in \mathcal{D}(f)\} .$$

Whenever $x \in \mathcal{D}(f)$, so are $x \pm P$, which implies that $(x \pm P)s = xs \pm sP$ are in the domain of g . Let $y \in \mathcal{D}(g)$, and write $y = xs$ for some $x \in \mathcal{D}(f)$. Then,

$$\begin{aligned} g(xs \pm sP) &= g((x \pm P)s) && \text{(regroup)} \\ &:= f(x \pm P) && \text{(definition of } g) \\ &= f(x) && (f \text{ has period } P) \\ &:= g(xs) . && \text{(definition of } g) \end{aligned}$$

This shows that g has period sP .

A preliminary observation is necessary before showing that sP is the *least positive* period of g . Observe that $\mathcal{D}(f)$ can be written as

$$\mathcal{D}(f) = \left\{ \frac{y}{s} \mid y \in \mathcal{D}(g) \right\} . \quad (*)$$

To see this, let $x \in \mathcal{D}(f)$; then $xs \in \mathcal{D}(g)$, and so $\frac{xs}{s} \in \left\{ \frac{y}{s} \mid y \in \mathcal{D}(g) \right\}$. That is, $x \in \left\{ \frac{y}{s} \mid y \in \mathcal{D}(g) \right\}$. Also, if $x \in \left\{ \frac{y}{s} \mid y \in \mathcal{D}(g) \right\}$, then $x = \frac{\hat{x}s}{s} = \hat{x}$ for some $\hat{x} \in \mathcal{D}(f)$, so that $x \in \mathcal{D}(f)$.

Now it is shown that sP is the least positive period of g . If P_1 is any positive number less than sP , write $P_1 = s\tilde{P}$, where $\tilde{P} < P$. If g has period $s\tilde{P}$, then it must be that, for every $x \in \mathcal{D}(f)$,

$$g(xs \pm s\tilde{P}) = g(xs) . \quad (**)$$

Since $xs \pm s\tilde{P} \in \mathcal{D}(g)$, $(*)$ shows that $x \pm \tilde{P} \in \mathcal{D}(f)$. But since $g(xs + s\tilde{P}) = f(x \pm \tilde{P})$ and $g(xs) = f(x)$, $(**)$ implies that $f(x \pm \tilde{P}) = f(x)$ for every $x \in \mathcal{D}(f)$, so that \tilde{P} is a period of f that is strictly less than P . This contradicts the fact that P is the least positive period of f . Therefore, sP must be the least positive period of g .

" \Leftarrow " For the remaining argument, use the result just proven to scale the function g by $\frac{1}{s}$, to obtain f . ■

LEMMA 4

Let f_1, \dots, f_N be functions with a common domain \mathcal{D} , and let $s > 0$. Scale each f_i by s to obtain a corresponding scaled function g_i , as per Lemma 3. Then,

$$\begin{aligned} p \text{ is a (fundamental) period of } f &:= \sum_{i=1}^N f_i \iff \\ sp \text{ is a (fundamental) period of } g &:= \sum_{i=1}^N g_i . \end{aligned}$$

PROOF

of Lemma 4

Since the functions f_i have a common domain \mathcal{D} , the functions g_i have a common domain

$$\mathcal{D}_s := \{xs \mid x \in \mathcal{D}\} ,$$

which is also the domain of the sum g . Also, for every $x \in \mathcal{D}$,

$$g(xs) := \sum_{i=1}^N g_i(xs) := \sum_{i=1}^N f_i(x) := f(x) .$$

‘ \Rightarrow ’ If p is any period of f , and xs is any element in \mathcal{D}_s , then

$$g(xs \pm sp) = g((x \pm p)s) = f(x \pm p) = f(x) = g(xs) ,$$

so that sp is a period of g .

‘ \Leftarrow ’ If sp is a period of g , then, for every $x \in \mathcal{D}$,

$$f(x \pm p) = g((x \pm p)s) = g(xs) = f(x) ,$$

so that f has period p .

An argument similar to that in Lemma 3 shows that the current lemma holds true with ‘period’ replaced by ‘fundamental period’. ■

Now, the proof of Theorem 3:

PROOF

of Theorem 3

[adapted from C&P, p. 34] Suppose that functions f_1, \dots, f_N have a common domain \mathcal{D} and commensurable least positive periods P_1, \dots, P_N . Then, the pairs $\{P_1, P_i\}$ for $2 \leq i \leq N$ are commensurable, so that by Lemma 2 there exist integers n_i and d_i ($2 \leq i \leq N$) with

$$P_i = \left(\frac{n_i}{d_i}\right)P_1 . \quad (*)$$

Let M be *any* common multiple of the integers d_2, \dots, d_N , and define $s := \frac{M}{P_1}$. Since f_i has least positive period P_i , using Lemma 3 and scaling by s produces a counterpart scaled function g_i with least positive period

$$\begin{aligned} sP_i &:= \left(\frac{M}{P_1}\right)P_i \\ &= \left(\frac{Mn_i}{d_i P_i}\right)P_i && \text{(using (*))} \\ &= \frac{Mn_i}{d_i} \\ &\in \mathbb{Z}, \text{ for all } i = 2, \dots, N, \end{aligned}$$

where membership in \mathbb{Z} results from M being a common multiple of the d_i . Also, $sP_1 = M \in \mathbb{Z}$. So, the functions f_i have 'scaled' counterparts g_i which have *integer* least positive periods sP_i , for $i = 1, \dots, N$.

The integer

$$(sP_1)(sP_2) \cdots (sP_N) \quad (**)$$

is a period of each g_i , and hence a period of the sum g . By Lemma 4, any rearrangement of (**) in the form $s \cdot K$ shows that K is a period of f . Thus, f is periodic. ■

Although Theorem 3 guarantees that the sum of periodic functions with commensurable least positive periods is itself periodic, it *cannot* guarantee that the sum has a fundamental period (see the answer to Question 2). Fortunately, the situation with discrete-domain periodic functions is considerably nicer:

COROLLARY 2 Every finite sum of discrete-domain periodic functions with a common domain is a discrete-domain periodic function with a least positive period.

PROOF
of Corollary 2

Let f_1, \dots, f_N be discrete-domain periodic functions with a common domain \mathcal{D} . By Corollary 1, each function f_i has a least positive period P_i , for $i = 1, \dots, N$. It is argued next that these least positive periods *must* be commensurable.

Suppose for contradiction that there exists a ratio $\frac{P}{Q}$ that is irrational, where $P = P_i$ and $Q = P_j$ for some i and j between 1 and N , $i \neq j$.

Choose any $x \in \mathcal{D}$. Since \mathcal{D} is a common domain and P is a period of a summand, $x \pm kP \in \mathcal{D}$ for every $k \in \mathbb{Z}$. Then, since Q is a period of a summand, it must be that $(x \pm kP) \pm jQ \in \mathcal{D}$, for every $k, j \in \mathbb{Z}$. In particular, $x + (kP + jQ) \in \mathcal{D}$ for all integers k and j . However, the set $\{kP + jQ \mid k \in \mathbb{Z}, j \in \mathbb{Z}\}$ is dense in \mathbb{R} since P and Q are incommensurable positive numbers [Olm, Theorem 4]. Thus, \mathcal{D} must be dense in \mathbb{R} . However, the domain of a discrete-domain function cannot be dense in \mathbb{R} : this supplies the necessary contradiction. Thus, the least positive periods of the summands must be commensurable.

Theorem 3 shows that the sum is a *periodic* discrete-domain function, which must have a least positive period by Corollary 1. ■

QUESTION 4: QUESTION: Suppose that functions f_1, f_2, \dots, f_N , each defined on \mathbb{R} , have fundamental integer periods P_1, P_2, \dots, P_N , respectively. In this case, the fundamental periods are commensurable, so the sum is periodic. If the sum has a fundamental period, what are the *candidates* for this fundamental period?

NOTATION:

lcm ,
 $a|b$,
relatively prime

Caveny and Page have answered this question in [C&P, pp. 38–41], and their result is presented next. In what follows, the notation ' lcm ' means *least common multiple*, and $a|b$ means that a divides b , i.e., b/a is an integer. Furthermore, two integers are *relatively prime* if and only if they have no common factors other than 1.

THEOREM 4
*candidates for the
 fundamental period
 of a sum*

[C&P, pp. 38–41] Let P_1, P_2, \dots, P_N be positive integers.

If there exist real-valued functions f_1, f_2, \dots, f_N defined on \mathbb{R} with respective fundamental periods P_1, P_2, \dots, P_N , such that the sum $f := f_1 + \dots + f_N$ has fundamental period P , then the number P must satisfy the following conditions:

- (a) $P = \frac{u}{v}$, where u and v are relatively prime integers, and u satisfies the following two conditions:
- (b) $u \mid lcm(P_1, P_2, \dots, P_N)$; and
- (c) u is divisible by each prime power that divides *precisely one* P_i ($1 \leq i \leq N$).

Conversely, whenever a real number P satisfies (a), (b), and (c), then there exist real-valued functions f_1, \dots, f_N defined on \mathbb{R} with respective fundamental periods P_1, \dots, P_N , such that the sum $f_1 + \dots + f_N$ has fundamental period P .

The following examples illustrate the use of this theorem. The first example shows how the result can be used with summands that have commensurable (but not integer) fundamental periods. The second example shows how the result can be used to obtain some information about sums of discrete-domain periodic functions.

EXAMPLE
using Theorem 4
with non-integer
periods

[C&P, p. 40] Suppose that functions g_1, g_2, g_3 and g_4 are defined on \mathbb{R} and have respective fundamental periods $Q_1 = 3\alpha$, $Q_2 = \frac{13}{4}\alpha$, $Q_3 = \frac{7}{2}\alpha$, and $Q_4 = \frac{15}{4}\alpha$, where α is irrational.

Scaling each function by $s := \frac{4}{\alpha}$ yields functions f_i ($1 \leq i \leq 4$) with respective integer fundamental periods $P_1 := Q_1 s = 12$, $P_2 := Q_2 s = 13$, $P_3 := Q_3 s = 14$ and $P_4 := Q_4 s = 15$. (A general procedure for producing such a scaling factor is discussed following the example.)

The prime factorizations of the P_i are:

$$P_1 = 2^2 \cdot 3, \quad P_2 = 13, \quad P_3 = 2 \cdot 7, \quad P_4 = 3 \cdot 5.$$

Observe that $\text{lcm}(P_1, P_2, P_3, P_4) = 2^2 \cdot 3 \cdot 5 \cdot 7 \cdot 13$.

What are the prime powers that divide *precisely one* P_i ?

2^2 only divides P_1 ;

13 only divides P_2 ;

7 only divides P_3 ;

5 only divides P_4 .

If the sum $f := f_1 + f_2 + f_3 + f_4$ has a least positive period P , then P must be of the form $P = \frac{u}{v}$, where u and v are relatively prime, u divides $2^2 \cdot 3 \cdot 5 \cdot 7 \cdot 13$, and u is divisible by 2^2 , 13 , 7 and 5 .

Thus, either $u = 2^2 \cdot 13 \cdot 7 \cdot 5 = 1820$ or $u = 3 \cdot (2^2 \cdot 13 \cdot 7 \cdot 5) = 5460$.

So the candidates for P are

$$\left\{ \frac{1820}{v} \mid v \text{ is a positive integer relatively prime to } 1820 \right\}$$

or

$$\left\{ \frac{5460}{v} \mid v \text{ is a positive integer relatively prime to } 5460 \right\}.$$

Scaling back to the original functions g_i by using the scale factor $\frac{1}{s} = \frac{\alpha}{4}$, the candidates for the fundamental period of $g_1 + g_2 + g_3 + g_4$ are

$$\left\{ \frac{1820}{v} \cdot \frac{\alpha}{4} \mid v \text{ is a positive integer relatively prime to } 1820 \right\}$$

or

$$\left\{ \frac{5460}{v} \cdot \frac{\alpha}{4} \mid v \text{ is a positive integer relatively prime to } 5460 \right\}.$$

Any more specific information about the fundamental period of the sum $g_1 + g_2 + g_3 + g_4$ would require additional information about the functions g_i .

*scaling to
obtain
integer periods*

As illustrated in the preceding example, the requirement in Theorem 4 that the fundamental periods P_1, \dots, P_N be *integers* is easily overcome by applying a common scaling factor.

Indeed, whenever functions f_1, \dots, f_N have *commensurable* fundamental periods, then a common scaling factor s can always be applied (if necessary) to obtain a new, scaled, set of functions that have integer periods. To do this, suppose that the respective fundamental periods of the f_i are

$$r_1\alpha, \dots, r_N\alpha, \quad (*)$$

where the r_i are rational and α is a (possible) irrational factor. Rewrite the rational factors r_i as equivalent fractions with least common denominator D , so that the list in (*) becomes

$$\frac{a_1\alpha}{D}, \dots, \frac{a_N\alpha}{D}.$$

The a_i are integers. Scaling by $\frac{D}{\alpha}$ then produces a new set of functions with respective fundamental integer periods a_1, \dots, a_N .

*using Theorem 4
with discrete-
domain functions*

Before using Theorem 4 with discrete-domain functions, it is necessary to understand how the set of periods of a discrete-domain function compares to the set of periods of its extension to \mathbf{R} . This is the content of Lemma 5.

LEMMA 5
*the periods of
an extension*

Suppose that f is a discrete-domain periodic function, with domain \mathcal{D} . Let f_e denote the extension of f to \mathbf{R} defined by

$$f_e(x) = \begin{cases} f(x) & \text{for } x \in \mathcal{D} \\ 0 & \text{for } x \notin \mathcal{D}. \end{cases}$$

Let \mathbf{P} denote the set of periods of f , and let \mathbf{P}_e denote the set of periods of f_e . Then, $\mathbf{P} \subset \mathbf{P}_e$. It is possible for the containment to be proper.

PROOF
of Lemma 5

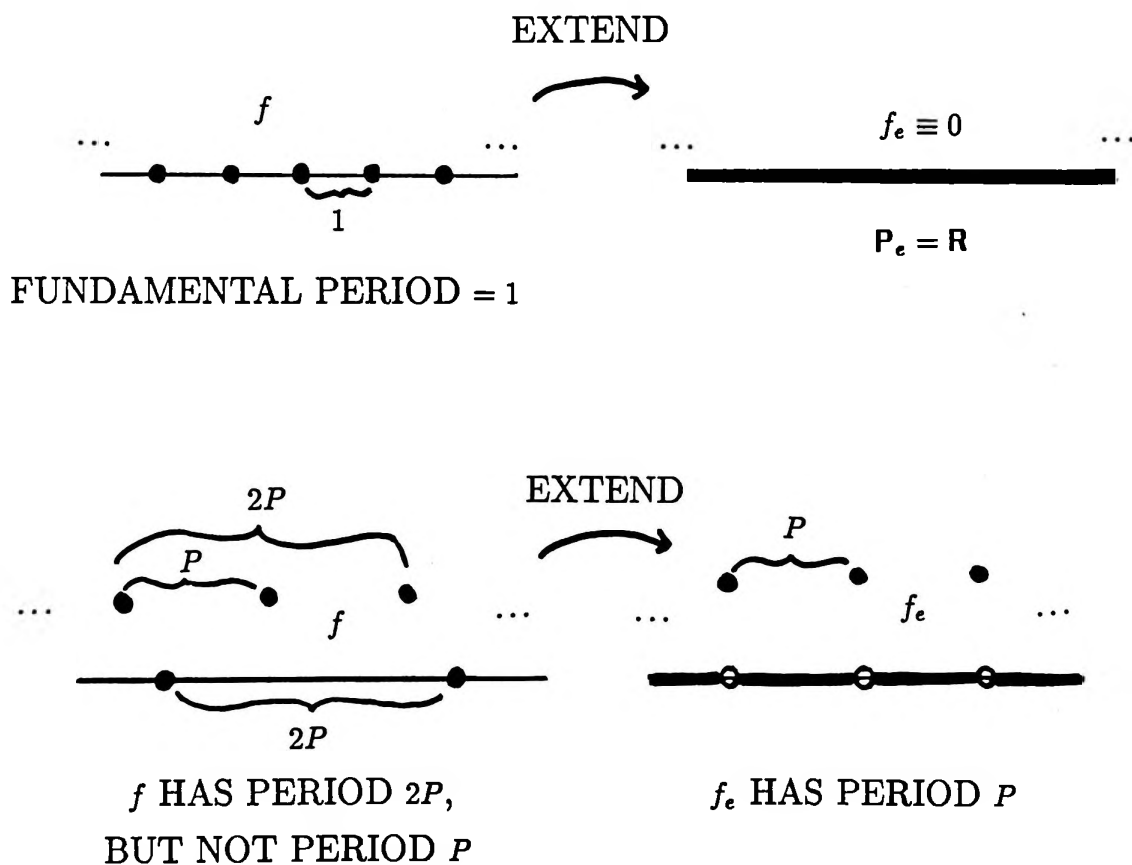
Let f, f_e, \mathcal{D}, P and P_e be as in the statement of the lemma. To show that $P \subset P_e$, suppose that P is a period of f , and let $x \in \mathbb{R}$. Either $x \in \mathcal{D}$ or $x \notin \mathcal{D}$.

If $x \in \mathcal{D}$, then so is $x \pm P$, and

$$\begin{aligned} f_e(x \pm P) &= f(x \pm P) && \text{(definition of } f_e) \\ &= f(x) && (P \text{ is a period of } f) \\ &= f_e(x). && \text{(definition of } f_e) \end{aligned}$$

If $x \notin \mathcal{D}$, then neither are $x \pm P$, so both $f_e(x) = 0$ and $f_e(x \pm P) = 0$. Thus, P is a period of f_e , so $P \in P_e$.

The examples below illustrate that P_e can be strictly larger than P . ■



It is a consequence of Lemma 5 that if $P \notin P_e$, then $P \notin P$. This observation is used in the next example.

EXAMPLE
*using Theorem 4
 with discrete-
 domain functions*

(It will be instructive for the reader to compare the discussion here with the answer to Question 3.)

Suppose that discrete-domain periodic functions f and g both have fundamental period 4, and a common domain. Suppose further that f and g are such that their extensions f_e and g_e to \mathbb{R} still have fundamental period 4.

Now, apply Theorem 4 to $(f + g)_e = f_e + g_e$. The candidates for the period of this (extended) sum are of the form $\frac{u}{v}$, where u and v are relatively prime and $u|4$. (In this case, there is *no* prime power that divides precisely one P_i .) Thus, u must equal 1, 2, or 4, so the candidates for the period of $f_e + g_e$ are

$$\left\{ \frac{1}{v} \mid v \text{ is a positive integer greater than } 1 \right\}$$

or

$$\left\{ \frac{2}{v} \mid v \text{ is any positive integer relatively prime to } 2 \right\}$$

or

$$\left\{ \frac{4}{v} \mid v \text{ is any positive integer relatively prime to } 4 \right\}.$$

If P is not one of the numbers in these sets, then P cannot be a period of $f + g$.

1.5 Using Identified Periodic Components for Prediction

Introduction

The next example raises a question that is of *fundamental* importance whenever an analyst wants to use identified periodic components for predictive purposes. Small numbers are used in the example, for ease of notation.

EXAMPLE conjecture that y is a sum of a 3-cycle and a 2-cycle

Suppose that an output list y is conjectured to be the sum of a 3-cycle (p) and a 2-cycle (q). That is,

$$p + q = y ,$$

or, in list form,

$$\begin{aligned} & (p_1, p_2, p_3, p_1, p_2, p_3, \dots) \\ + & (q_1, q_2, q_1, q_2, q_1, q_2, \dots) \\ = & (y_1, y_2, y_3, y_4, y_5, y_6, \dots) . \end{aligned} \tag{1}$$

This conjecture may have come from some preliminary data analysis on the already-observed output values y ; or from some understanding of the underlying mechanism(s) generating the observed data.

the 'knowns' and 'unknowns'

The y_i values are being observed by the analyst; once a given value is observed, then it is *known*. The entries p_i and q_i in the cycles are *unknown*. Based on the observed values of y_i , it is desired to either support or deny the conjecture that y is the sum of a 3-cycle and a 2-cycle; if supported, it is desired to identify lists p and q that can be used to predict future values of y .

equating entries in (1) yields a linear system

By summing and then equating entries in (1), a system of linear equations in the unknowns p_i and q_i emerges:

$$\begin{aligned} p_1 + q_1 &= y_1 \\ p_2 + q_2 &= y_2 \\ p_3 + q_1 &= y_3 \\ &\vdots \end{aligned} \tag{2}$$

some linear
algebra
considerations

Some linear algebra considerations regarding the linear system just produced are in order. Since there are 5 unknowns (p_1 , p_2 , p_3 , q_1 , and q_2), one needs 5 pieces of non-overlapping, non-contradictory information to uniquely determine the 5 unknowns. One *always* needs n pieces of non-overlapping, non-contradictory information to uniquely determine n unknowns in a linear system. However, it has been previously noted that p and q can *not* be uniquely determined, since adding any constant K to p and subtracting the same constant K from q gives the same sum. Thus, analysis of any system of the form indicated in (2) will *not* produce a unique solution.

augmented
matrix for (2)

How many pieces of data must the analyst observe, in order to 'identify' lists p and q that can be used for predictive purposes? If there is any *hope* of identifying the p_i and q_i , then these unknowns must actually *appear* in the system: thus, one must observe at least the first three values of y . This gives the linear system in the five variables p_1 , p_2 , p_3 , q_1 , and q_2 ,

$$p_1 + q_1 = y_1$$

$$p_2 + q_2 = y_2$$

$$p_3 + q_1 = y_3 ,$$

with matrix form

$$\begin{bmatrix} 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 \end{bmatrix} \begin{bmatrix} p_1 \\ p_2 \\ p_3 \\ q_1 \\ q_2 \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} ,$$

and with augmented matrix

$$\left[\begin{array}{ccc|cc} 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 \end{array} \right] \begin{array}{l} \vdots \\ y_1 \\ \vdots \\ y_2 \\ \vdots \\ y_3 \end{array} \quad (3)$$

Pay particular attention to the identity matrices and partial identity matrices that appear in the resulting augmented matrix. Similar patterns will always emerge when investigating sums of periodic lists.

The system in (3) has two degrees of freedom. Once values are chosen for q_1 and q_2 , the remaining values p_1 , p_2 , and p_3 are uniquely determined:

$$p_1 = y_1 - q_1$$

$$p_2 = y_2 - q_2$$

$$p_3 = y_3 - q_1 .$$

It is readily checked that the resulting lists p and q do indeed sum to give the first 3 observed values of y :

$$\begin{aligned} & (y_1 - q_1, y_2 - q_2, y_3 - q_1) \\ + & (q_1, q_2, q_1) \\ = & (y_1, y_2, y_3) , \end{aligned}$$

regardless of the choices made for q_1 and q_2 .

*the identified
lists
are useless for
predictive
purposes*

Can the 'identified' lists p and q be used to predict future values of y ? To investigate this question, use the lists p and q determined in (3) to 'predict' a fourth value of y , giving

$$\begin{aligned} y_4 &= p_1 + q_2 \\ &= (y_1 - q_1) + q_2 . \end{aligned}$$

Here is the critical observation: the predicted value of y_4 *depends on* the choices made for q_1 and q_2 ! That is, different choices of 3-cycles and 2-cycles that sum to give the first 3 observed values of y , can lead to *different* predicted values for y_4 , as illustrated next.

EXAMPLE

For example, choosing

$$y_1 = 1, y_2 = 2, y_3 = 3, q_1 = 1, \text{ and } q_2 = 0$$

yields

$$p_1 = y_1 - q_1 = 1 - 1 = 0$$

$$p_2 = y_2 - q_2 = 2 - 0 = 2$$

$$p_3 = y_3 - q_1 = 3 - 1 = 2$$

so that

$$(0, 2, 2, 0) + (1, 0, 1, 0) = (\overbrace{1, 2, 3}^{\text{known}}, \overbrace{0}^{\text{predicted}}) .$$

Here, the first three (known) values of y are obtained; and one 'predicts' the value 0, based on the 'identified' periodic lists.

Alternately, choosing the same values for y_1 , y_2 and y_3 , but choosing $q_1 = 0$ and $q_2 = 1$ yields

$$p_1 = 1 - 0 = 1$$

$$p_2 = 2 - 1 = 1$$

$$p_3 = 3 - 0 = 3$$

so that

$$(1, 1, 3, 1) + (0, 1, 0, 1) = (\overbrace{1, 2, 3}^{\text{known}}, \overbrace{2}^{\text{predicted}}) .$$

Here, the first three (known) values of y are obtained; and one 'predicts' the *different* value 2, based on the 'identified' periodic lists.

Thus, when only 3 values of y have been observed, any 'identified' periodic lists are *useless* for predictive purposes. Different periodic lists can lead to different predicted values.

one more piece
of data is
observed

Suppose that one more piece of data, y_4 , is observed (and hence now *known*). Then, p and q depend on the 4 numbers y_1 , y_2 , y_3 , and y_4 .

The new system has the augmented matrix

$$\begin{bmatrix} 1 & 0 & 0 & 1 & 0 & \vdots & y_1 \\ 0 & 1 & 0 & 0 & 1 & \vdots & y_2 \\ 0 & 0 & 1 & 1 & 0 & \vdots & y_3 \\ 1 & 0 & 0 & 0 & 1 & \vdots & y_4 \end{bmatrix} . \quad (4)$$

NOTATION

$$R'_m = kR_n + R_m$$

This augmented matrix is transformed, via Gauss-Jordan elimination, to a matrix for an equivalent system that is much easier to analyze. The notation

$$R'_m = kR_n + R_m$$

is used to denote the row operation 'replace (Row m) by (k times Row n) added to (Row m)'. That is,

$$\overbrace{R'_m}^{\text{new Row } m} = \overbrace{kR_n}^{k \text{ times Row } n} + \overbrace{R_m}^{\text{old Row } m} .$$

The row operations below transform the matrix in (4) to reduced row-echelon form:

$$\begin{aligned}
 R'_4 &= -R_1 + R_4 & \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & \vdots & y_1 \\ 0 & 1 & 0 & 0 & 1 & \vdots & y_2 \\ 0 & 0 & 1 & 1 & 0 & \vdots & y_3 \\ 0 & 0 & 0 & -1 & 1 & \vdots & -y_1 + y_4 \end{bmatrix} & \text{(zero first column)} \\
 R'_4 &= -R_4 & \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & \vdots & y_1 \\ 0 & 1 & 0 & 0 & 1 & \vdots & y_2 \\ 0 & 0 & 1 & 1 & 0 & \vdots & y_3 \\ 0 & 0 & 0 & 1 & -1 & \vdots & y_1 - y_4 \end{bmatrix} & \text{(leading 1 in last row)} \\
 R'_3 &= -R_4 + R_3 & \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & \vdots & y_1 \\ 0 & 1 & 0 & 0 & 1 & \vdots & y_2 \\ 0 & 0 & 1 & 0 & 1 & \vdots & -y_1 + y_4 + y_3 \\ 0 & 0 & 0 & 1 & -1 & \vdots & y_1 - y_4 \end{bmatrix} & \begin{array}{l} \text{fourth} \\ \text{(zero ~~third~~ column)} \end{array} \\
 R'_1 &= -R_4 + R_1 & \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & \vdots & y_4 \\ 0 & 1 & 0 & 0 & 1 & \vdots & y_2 \\ 0 & 0 & 1 & 0 & 1 & \vdots & -y_1 + y_4 + y_3 \\ 0 & 0 & 0 & 1 & -1 & \vdots & y_1 - y_4 \end{bmatrix} & \begin{array}{l} \text{fourth} \\ \text{(zero ~~third~~ column)} \end{array} \quad (5)
 \end{aligned}$$

The system in (5) has one degree of freedom. Once a value is chosen for q_2 , the values of p_1 , p_2 , p_3 and q_1 are uniquely determined:

$$\begin{aligned}
 p_1 &= y_4 - q_2 \\
 p_2 &= y_2 - q_2 \\
 p_3 &= -y_1 + y_4 + y_3 - q_2 \\
 q_1 &= y_1 - y_4 + q_2
 \end{aligned}$$

Summing the 'identified' components, and using this sum to predict the next value of y gives:

$$\begin{aligned}
 & (\quad y_4 - q_2, \quad y_2 - q_2, \quad -y_1 + y_4 + y_3 - q_2, \quad y_4 - q_2, \quad y_2 - q_2) \\
 & + (y_1 - y_4 + q_2, \quad q_2, \quad y_1 - y_4 + q_2, \quad q_2, \quad y_1 - y_4 + q_2) \\
 & = (\quad y_1, \quad y_2, \quad y_3, \quad y_4, \quad y_2 + y_1 - y_4) .
 \end{aligned}$$

The critical observation is that the predicted value for y_5 , $y_5 = y_2 + y_1 - y_4$, is *independent of the choice of q_2* . Indeed, since each entry in p is decreased by q_2 , and each entry in q is increased by q_2 , *all predicted values of y will be independent of q_2* .

summary

The example just completed points out some interesting facts. If data is indeed a sum of a 3-cycle and a 2-cycle, then, having 4 pieces of data is 'enough' to use any identified components for prediction, in the following sense: there will still be an infinite number of periodic components that could sum to give the first four observed values of y , but, *no matter what components are chosen, they will yield the same predicted values for y .*

From a practical point of view, this type of information is vitally important. Suppose that (say, based on preliminary data analysis), it is conjectured that observed data is a sum of a p -cycle and a q -cycle. IF:

- p and q are appropriately related (see Theorem 1);
- the observed data is truly a sum of a p -cycle and a q -cycle;
- enough data has been observed (see Theorem 1); and
- the investigator is able to identify *any* p -cycle and q -cycle that sum to give the observed data;

then these cycles can be used for prediction. It will not matter if they are indeed the 'right' cycles or not; since *any* choice yields the same predicted values.

THEOREM 1

Suppose that y is a sum of a p -cycle and a q -cycle, where p and q are positive integers satisfying $p = kq + 1$ for a positive integer k . If p and q are *any* p and q -cycles (respectively) that sum to give the first $p + q - 1$ observed values of y , then p and q can be used to predict future values of y .

PROOF
of Theorem 1

The proof is constructive, and gives an algorithm for producing all possible p and q -cycles that can sum to produce y . It is shown that each entry of the p -cycle, p , must have an additive constant K , and that each entry of the q -cycle, q , must have an additive constant $-K$, thus proving that predicted values are independent of the particular choice of components p and q .

$I_n, 0_{m,n}$

In what follows, the notation I_n is used for the $n \times n$ identity matrix; and the notation $0_{m,n}$ is used for the $m \times n$ zero matrix.

p, q

Let p and q be positive integers with $p = kq + 1$ for a positive integer k . Let \mathbf{p} and \mathbf{q} denote the p -cycle and q -cycle, respectively, with entries

$$\mathbf{p} = (x_1, x_2, x_3, \dots, x_p, x_1, x_2, \dots)$$

and

$$\mathbf{q} = (w_1, w_2, \dots, w_q, w_1, w_2, \dots).$$

Suppose that $\mathbf{p} + \mathbf{q} = \mathbf{y}$, where \mathbf{y} has entries

$$\mathbf{y} = (y_1, y_2, y_3, \dots, y_q, \dots, y_p, \dots),$$

and suppose that the first $p + q - 1$ values of \mathbf{y} have been observed. The augmented matrix below is used to solve for the unknown values of x_i and w_i :

$$\begin{array}{c}
 \begin{array}{l} \text{LAST} \\ q-1 \\ \text{ROWS} \end{array} \left\{ \begin{array}{c} \begin{array}{c} 1 \ 0 \ \dots \ 0 \ \dots \ 0 \\ \vdots \\ 0 \ 0 \ \dots \ 0 \ \dots \ 1 \\ \vdots \\ 0 \ 0 \ \dots \ 0 \ \dots \ 1 \\ \vdots \\ 1 \ 0 \ \dots \ 0 \ \dots \ 0 \\ \vdots \\ 0 \ 0 \ \dots \ 1 \ \dots \ 0 \end{array} \\ \vdots \\ \begin{array}{c} 0 \ 0 \ \dots \ 0 \ \dots \ 1 \\ 1 \ 0 \ \dots \ 0 \ \dots \ 0 \\ \vdots \\ 0 \ 0 \ \dots \ 1 \ \dots \ 0 \end{array} \end{array} \right. \begin{array}{c} \begin{array}{c} 1 \ 0 \ \dots \ 0 \ \dots \ 0 \ 0 \\ \vdots \\ 0 \ 0 \ \dots \ 0 \ \dots \ 0 \ 1 \\ \vdots \\ 1 \ 0 \ \dots \ 0 \ \dots \ 0 \ 0 \\ \vdots \\ 0 \ 0 \ \dots \ 0 \ \dots \ 0 \ 1 \\ \vdots \\ 1 \ 0 \ \dots \ 0 \ \dots \ 0 \ 0 \\ \vdots \\ 0 \ 1 \ \dots \ 0 \ \dots \ 0 \ 0 \end{array} \\ \vdots \\ \begin{array}{c} 1 \ 0 \ \dots \ 0 \ \dots \ 0 \ 0 \\ \vdots \\ 0 \ 1 \ \dots \ 0 \ \dots \ 0 \ 0 \\ \vdots \\ 0 \ 0 \ \dots \ 0 \ \dots \ 0 \ 1 \end{array} \end{array} \right. \begin{array}{c} y_1 \\ \vdots \\ y_q \\ y_{q+1} \\ \vdots \\ y_{2q} \\ \vdots \\ y_{(k-1)q+1} \\ \vdots \\ y_{p-1} \\ y_p \leftarrow \text{ROW } p \\ y_{p+1} \\ \vdots \\ y_{p+q-1} \end{array} \begin{array}{c} k \\ \text{BLOCKS} \\ \text{OF } I_q \end{array}
 \end{array}$$

The first p rows already have leading ones. Use these leading ones to zero out the last $q - 1$ rows, via the row operations

$$R'_{p+i} = -R_i + R_{p+i}, \quad 1 \leq i \leq q - 1.$$

This does not affect the first p rows, but gives the new last $q - 1$ rows:

$$\begin{array}{c} \text{LAST} \\ q-1 \\ \text{ROWS} \end{array} \left[\begin{array}{ccc|cccccccc} 0 & \dots & 0 & -1 & 1 & \dots & 0 & 0 & \dots & 0 & 0 & \vdots & -y_1 + y_{p+1} \\ & & & & & & \vdots & & & & & & \vdots \\ O_{q-1,p} & & & 0 & 0 & \dots & -1 & 1 & \dots & 0 & 0 & \vdots & -y_j + y_{p+j} \\ & & & & & & \vdots & & & & & & \vdots \\ 0 & \dots & 0 & 0 & 0 & \dots & 0 & 0 & \dots & -1 & 1 & \vdots & -y_{q-1} + y_{p+q-1} \end{array} \right] \begin{array}{c} \text{ROW} \\ p+1 \end{array}$$

\uparrow
 COLUMN
 $p+j$

Next, get leading ones in these last $q-1$ rows, via the row operations

$$R'_{p+j} = -R_{p+j}, \quad 1 \leq j \leq q-1.$$

This gives the new last $q-1$ rows:

$$\begin{array}{c} \text{LAST} \\ q-1 \\ \text{ROWS} \end{array} \left[\begin{array}{ccc|cccccccc} 0 & \dots & 0 & 1 & -1 & \dots & 0 & 0 & \dots & 0 & 0 & \vdots & y_1 - y_{p+1} \\ & & & & & & \vdots & & & & & & \vdots \\ O_{q-1,p} & & & 0 & 0 & \dots & 1 & -1 & \dots & 0 & 0 & \vdots & y_j - y_{p+j} \\ & & & & & & \vdots & & & & & & \vdots \\ 0 & \dots & 0 & 0 & 0 & \dots & 0 & 0 & \dots & 1 & -1 & \vdots & y_{q-1} - y_{p+q-1} \end{array} \right] \begin{array}{c} \text{ROW} \\ p+1 \end{array}$$

\uparrow
 COLUMN
 $p+j$

Next, use the leading one in row $p+1$ to zero out the $p+1$ column, via the row operations

$$R'_{j_q+1} = -R_{p+1} + R_{j_q+1}, \quad 0 \leq j \leq k.$$

Observe that $kq+1 = p$. The first $(p+q-1) \times p$ submatrix is not affected by any future row operations, and is not shown. The remaining matrix becomes:

$\begin{bmatrix} 0 & 1 & 0 & \dots & 0 & 0 & \dots & 0 & 0 \\ 0 & 1 & 0 & \dots & 0 & 0 & \dots & 0 & 0 \\ & & & & \vdots & & & & \\ 0 & 0 & 0 & \dots & 0 & 0 & \dots & 0 & 1 \end{bmatrix}$	y_{p+1} y_2 \vdots y_q	
$\begin{bmatrix} 0 & 1 & 0 & \dots & 0 & 0 & \dots & 0 & 0 \\ 0 & 1 & 0 & \dots & 0 & 0 & \dots & 0 & 0 \\ & & & & \vdots & & & & \\ 0 & 0 & 0 & \dots & 0 & 0 & \dots & 0 & 1 \end{bmatrix}$	$-y_1 + y_{p+1} + y_{q+1}$ y_{q+2} \vdots y_{2q}	
\vdots	\vdots	
$\begin{bmatrix} 0 & 1 & 0 & \dots & 0 & 0 & \dots & 0 & 0 \\ 0 & 1 & 0 & \dots & 0 & 0 & \dots & 0 & 0 \\ & & & & \vdots & & & & \\ 0 & 0 & 0 & \dots & 0 & 0 & \dots & 0 & 1 \end{bmatrix}$	$-y_1 + y_{p+1} + y_{(k-1)q+1}$ $y_{(k-1)q+2}$ \vdots y_{p-1}	
$\begin{bmatrix} 0 & 1 & 0 & \dots & 0 & 0 & \dots & 0 & 0 \\ 1 & -1 & 0 & \dots & 0 & 0 & \dots & 0 & 0 \\ 0 & 1 & -1 & \dots & 0 & 0 & \dots & 0 & 0 \\ & & & & \vdots & & & & \\ 0 & 0 & 0 & \dots & 1 & -1 & \dots & 0 & 0 \\ & & & & \vdots & & & & \\ 0 & 0 & 0 & \dots & 0 & 0 & \dots & 1 & -1 \end{bmatrix}$	$-y_1 + y_{p+1} + y_p$ $y_1 - y_{p+1}$ $y_2 - y_{p+2}$ \vdots $y_j - y_{p+j}$ \vdots $y_{q-1} - y_{p+q-1}$	\leftarrow ROW p \leftarrow ROW $p+1$ \leftarrow ROW $p+2$

Next, use the leading one in row $p+2$ to zero out the $p+2$ -column, via the row operations

$$R'_{jq+1} = -R_{p+2} + R_{jq+1}, \quad 0 \leq j \leq k,$$

$$R'_{jq+2} = -R_{p+2} + R_{jq+2}, \quad 0 \leq j \leq k-1,$$

and

$$R'_{p+1} = R_{p+2} + R_{p+1}.$$

The resulting matrix is:

$ \begin{array}{cccccccccc} 0 & 0 & 1 & 0 & \dots & 0 & 0 & \dots & 0 & 0 \\ 0 & 0 & 1 & 0 & \dots & 0 & 0 & \dots & 0 & 0 \\ 0 & 0 & 1 & 0 & \dots & 0 & 0 & \dots & 0 & 0 \\ & & & & & & \vdots & & & \\ 0 & 0 & 0 & 0 & \dots & 0 & 0 & \dots & 0 & 1 \\ \hline 0 & 0 & 1 & 0 & \dots & 0 & 0 & \dots & 0 & 0 \\ 0 & 0 & 1 & 0 & \dots & 0 & 0 & \dots & 0 & 0 \\ 0 & 0 & 1 & 0 & \dots & 0 & 0 & \dots & 0 & 0 \\ & & & & & & \vdots & & & \\ 0 & 0 & 0 & 0 & \dots & 0 & 0 & \dots & 0 & 1 \\ \hline & & & & & & \vdots & & & \\ & & & & & & \vdots & & & \\ \hline 0 & 0 & 1 & 0 & \dots & 0 & 0 & \dots & 0 & 0 \\ 0 & 0 & 1 & 0 & \dots & 0 & 0 & \dots & 0 & 0 \\ 0 & 0 & 1 & 0 & \dots & 0 & 0 & \dots & 0 & 0 \\ & & & & & & \vdots & & & \\ 0 & 0 & 0 & 0 & \dots & 0 & 0 & \dots & 0 & 1 \\ \hline 0 & 0 & 1 & 0 & \dots & 0 & 0 & \dots & 0 & 0 \\ \hline 1 & 0 & -1 & 0 & \dots & 0 & 0 & \dots & 0 & 0 \\ 0 & 1 & -1 & 0 & \dots & 0 & 0 & \dots & 0 & 0 \\ 0 & 0 & 1 & -1 & \dots & 0 & 0 & \dots & 0 & 0 \\ & & & & & & \vdots & & & \\ 0 & 0 & 0 & 0 & \dots & 1 & -1 & \dots & 0 & 0 \\ & & & & & & \vdots & & & \\ 0 & 0 & 0 & 0 & \dots & 0 & 0 & \dots & 1 & -1 \end{array} $	$ \begin{array}{c} -y_2 + y_{p+2} + y_{p+1} \\ y_{p+2} \\ y_3 \\ \vdots \\ y_q \\ -y_2 + y_{p+2} - y_1 + y_{p+1} + y_{q+1} \\ -y_2 + y_{p+2} + y_{q+2} \\ y_{q+3} \\ \vdots \\ y_{2q} \\ \vdots \\ \vdots \\ -y_2 + y_{p+2} - y_1 + y_{p+1} + y_{(k-1)q+1} \\ -y_2 + y_{p+2} + y_{(k-1)q+2} \\ y_{(k-1)q+3} \\ \vdots \\ y_{p-1} \\ -y_2 + y_{p+2} - y_1 + y_{p+1} + y_p \leftarrow \text{row } p \\ y_2 - y_{p+2} + y_1 - y_{p+1} \leftarrow \text{row } p+1 \\ y_2 - y_{p+2} \leftarrow \text{row } p+2 \\ y_3 - y_{p+3} \leftarrow \text{row } p+3 \\ \vdots \\ y_j - y_{p+j} \\ \vdots \\ y_{q-1} - y_{p+q-1} \end{array} $
---	---

The equivalent system represented by this augmented matrix has one degree of freedom. Once a choice is made for w_q , the remaining unknowns x_1, \dots, x_p and w_1, \dots, w_{q-1} are uniquely determined:

$$\begin{aligned} x_1 &= f_1 - w_q \\ &\vdots \\ x_p &= f_p - w_q \\ w_1 &= f_{p+1} + w_q \\ &\vdots \\ w_{q-1} &= f_{p+q-1} + w_q. \end{aligned}$$

Since each entry of p is decreased by w_q , and each entry of q is increased by w_q , the sum $p + q$ is uniquely determined by the numbers y_1 through y_{p+q-1} , and hence future values of the sum are uniquely determined. ■

It is interesting to note the following:

PROPOSITION Let p, q and k be positive integers with $p = kq + 1$. Then, p and q are relatively prime.
 $p = kq + 1$ implies that p and q are relatively prime

PROOF
of Proposition

Suppose for contradiction that p and q have a common factor other than 1. That is, suppose there exists a positive integer $j \neq 1$ for which both $p = jx$ and $q = jy$; here, x and y denote the remaining primes in the prime decompositions of p and q , respectively. The following list of equivalent equations is then obtained:

$$\begin{aligned} p = kq + 1 &\iff jx = k(jy) + 1 \\ &\iff j(x - ky) = 1 \\ &\iff x - ky = \frac{1}{j}. \end{aligned}$$

Since x, k and y are integers, and the integers are closed under addition and multiplication, $x - ky$ is an integer. However, $\frac{1}{j}$ is not an integer. Thus, the last equation $x - ky = \frac{1}{j}$ is false. By hypothesis, however, the first equation $p = kq + 1$ is true. This gives the desired contradiction. ■

Thus, the requirements on p and q in Theorem 1 force p and q to be relatively prime.

*contents of
the next two
sections*

The next section provides a thorough investigation of the sine and cosine functions. These functions have been of great historical importance in the context of periodicity. Also, they will become important at the end of Chapter 2 and in Chapter 3, when studying the periodogram of a data set, and mathematical filters.

The chapter closes with a discussion of historical contributions in the search for hidden periodicities.

1.6 Sinusoids

a point
traveling around
the unit circle

The x and y coordinates of a point traveling around the unit circle $x^2 + y^2 = 1$ exhibit periodic behavior: after each revolution, the coordinates of the point repeat. This simple observation is the basis for the definitions of the important, periodic, sine and cosine functions.

DEFINITION
sine function;
cosine function

Let t be any real number. Identify this real number t with a point on the unit circle $C: x^2 + y^2 = 1$, as follows:

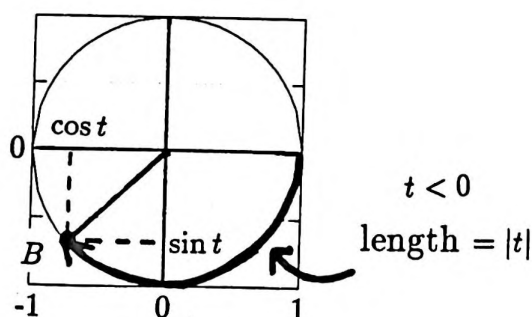
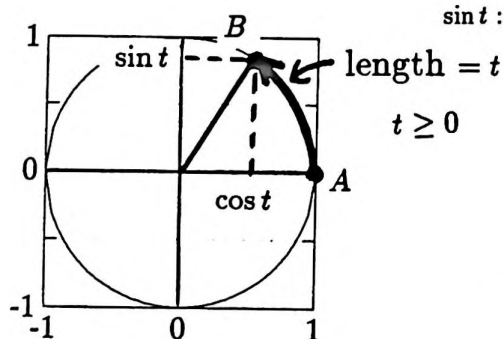
- Start at the point $A := (1, 0)$.
- If $t \geq 0$, lay off an arc of length t , in the counter-clockwise direction, on C .
- If $t < 0$, lay off an arc of length $|t|$, in the clockwise direction, on C .
- Denote the point on C at the terminal end of the arc by B . In this manner, every real number t is associated with a point B on the unit circle.

Define the cosine function $\cos: \mathbf{R} \rightarrow [-1, 1]$ by

$\cos t :=$ the x -coordinate of B ,

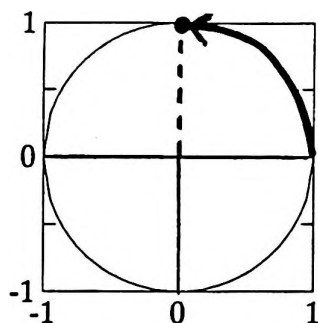
and define the sine function $\sin: \mathbf{R} \rightarrow [-1, 1]$ by

$\sin t :=$ the y -coordinate of B .



EXAMPLE Since the unit circle has circumference 2π , the real number $t = \frac{\pi}{2}$ corresponds to the point $(0, 1)$ on C ; hence, $\sin \frac{\pi}{2} = 1$ and $\cos \frac{\pi}{2} = 0$.

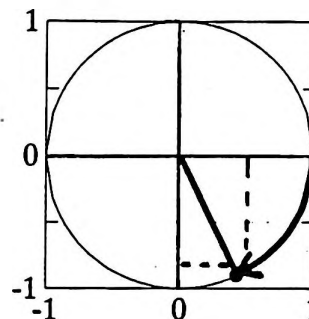
The number $t = -\frac{\pi}{3}$ corresponds to the point $(\frac{1}{2}, -\frac{\sqrt{3}}{2})$ on C ; hence, $\sin(-\frac{\pi}{3}) = -\frac{\sqrt{3}}{2}$ and $\cos(-\frac{\pi}{3}) = \frac{1}{2}$.



$$t = \frac{\pi}{2}$$

x -coordinate = 0

y -coordinate = 1



$$t = -\frac{\pi}{3}$$

x -coordinate = $\frac{1}{2}$

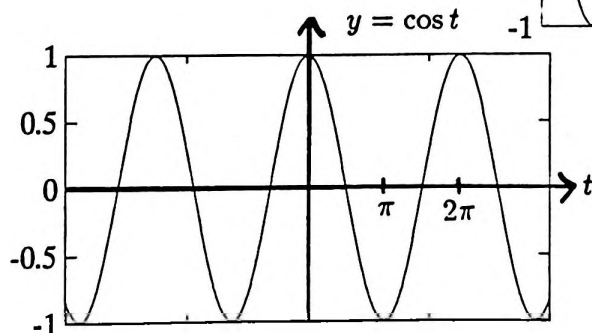
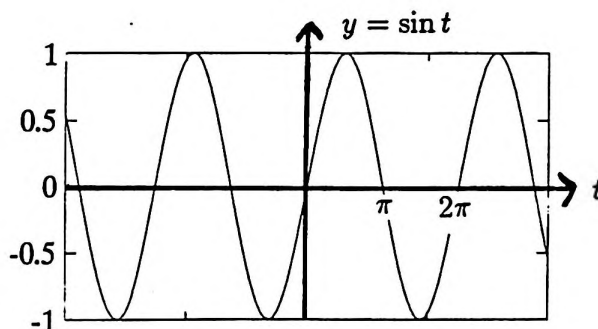
y -coordinate = $-\frac{\sqrt{3}}{2}$

*graphs of
the sine
and cosine*

The sine and cosine functions are graphed below. Since all real numbers $t \pm 2\pi k$, $k \in \mathbb{Z}$, are associated with the same point on C , one has

$$\sin(t \pm 2\pi k) = \sin t \quad \text{and} \quad \cos(t \pm 2\pi k) = \cos t \quad \forall t \in \mathbb{R}, k \in \mathbb{Z}.$$

The fundamental period of both the sine and cosine functions is 2π .



measures
for angles

radian;
degree;
grad

The process of identifying a real number t with a point on the unit circle C forms an angle in a natural way: use the positive x -axis as the beginning side of the angle, and the ray through the origin and B as the terminal side of the angle.

By definition, the number t is the *radian measure* of the angle thus formed. Therefore, the radian measure of an angle is a *real number* that represents a (signed) arc length on the unit circle. Note that 2π radians equals one complete revolution.

The *degree measure* of an angle is defined by dividing one complete revolution into 360 equal parts: by definition,

$$1^\circ = \frac{1}{360} \text{ revolution .}$$

Thus, $(2\pi \text{ radians}) = 360^\circ$.

The *grad measure* of an angle is defined by dividing one complete revolution into 400 equal parts: by definition,

$$1 \text{ grad} = \frac{1}{400} \text{ revolution .}$$

Thus, $400 \text{ grad} = 360^\circ = 2\pi \text{ radians}$. Only the radian measure of an angle will be used in this dissertation.

DEFINITION
sinusoid

In this dissertation, a *sinusoid* is a function of the form

$$a \sin(bt + c) \quad \text{or} \quad a \cos(bt + c) ,$$

for real numbers a , b and c with $a \neq 0$ and $b \neq 0$.

*fundamental
period;
phase shift;
amplitude*

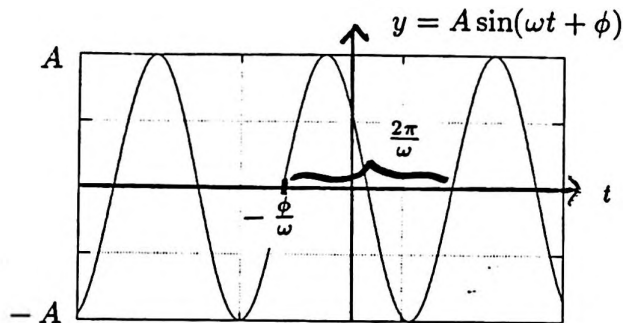
Let $f(t)$ denote either $a \sin(bt + c)$ or $a \cos(bt + c)$.

The function f makes one complete cycle as the argument of the sine or cosine goes from 0 to 2π . Observe that $bt + c = 0$ when $t = -\frac{c}{b}$; and, $bt + c = 2\pi$ when $t = \frac{2\pi - c}{b}$. The resulting time interval has length $|\frac{2\pi - c}{b} - (-\frac{c}{b})| = |\frac{2\pi}{b}|$. The positive number $|\frac{2\pi}{b}|$ is the *fundamental period* of f .

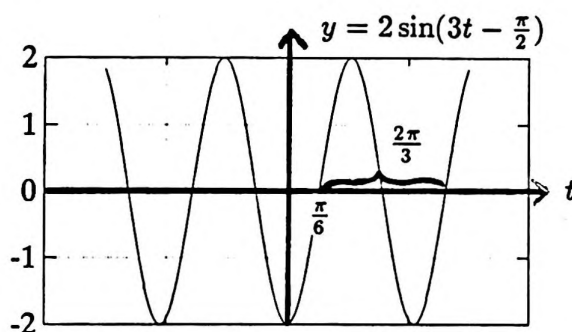
The number $-\frac{c}{b}$ is the *phase shift* of f . Observe that $a \sin(bt + c) = a \sin[b(t + \frac{c}{b})]$, so the curve $y = a \sin(bt)$ is shifted $\frac{c}{b}$ units to the left (if $\frac{c}{b} \geq 0$), or $\frac{c}{b}$ units to the right (if $\frac{c}{b} < 0$) to obtain the curve $y = a \sin(bt + c)$. A similar statement holds for $a \cos(bt + c)$.

For all real numbers t , $|f(t)| \leq |a|$, and on any interval of length $|\frac{2\pi}{b}|$, the function f takes on the values $\pm|a|$. The number $|a|$ is the *amplitude* of f .

Two sinusoids are graphed below.



$$\begin{aligned} A &> 0 \\ \omega &> 0 \\ 0 &< \phi < \pi \end{aligned}$$



*non-uniqueness
of representation*

The representation of a sinusoid in the form $a \sin(bt + c)$ or $a \cos(bt + c)$ is not unique. For example, for all real numbers a , b and c with $a \neq 0$ and $b \neq 0$, and for all integers k ,

$$\begin{aligned} a \cos(bt + c) &= a \sin(bt + c + \frac{\pi}{2}) \\ a \cos(bt + c) &= -a \cos(bt + c \pm \pi) \\ a \sin(bt + c) &= -a \sin(-bt - c + 2\pi k) . \end{aligned}$$

However, appropriate restrictions can be placed on a , b and c so that a unique representation is obtained. This is the content of the next proposition.

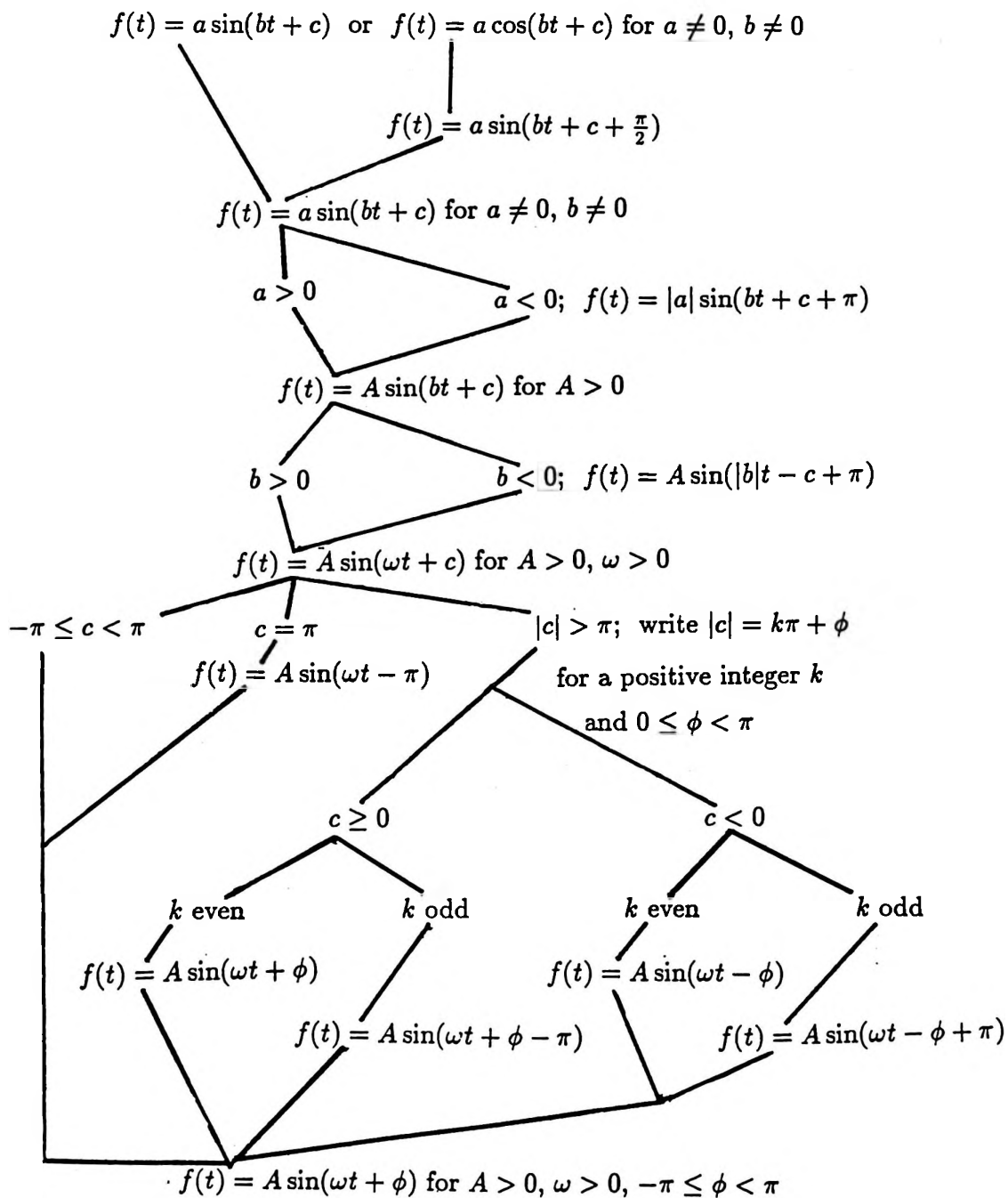
Proposition 1
*unique
representation
for a sinusoid*

Every sinusoid has a unique representation in the form

$$A \sin(\omega t + \phi) ,$$

for $A > 0$, $\omega > 0$, and $-\pi \leq \phi < \pi$.

FLOW CHART for PROPOSITION 1



PROOF

*an algorithm
for obtaining
the desired form*

For all real numbers t and odd integers k ,

$$\cos t = \sin\left(t + \frac{\pi}{2}\right) \quad (1)$$

$$\sin(t \pm k\pi) = -\sin t \quad (2)$$

These facts are used freely in the following proof.

Let f be any sinusoid. Thus, f is of the form $f(t) = a \sin(bt + c)$ or $f(t) = a \cos(bt + c)$ for real numbers a , b and c , with $a \neq 0$ and $b \neq 0$. The algorithm presented next produces a representation of f in the desired form. The flow chart on page 76 summarizes this algorithm.

If $f(t) = a \cos(bt + c)$, then

$$f(t) = a \cos(bt + c) \stackrel{(1)}{=} a \sin\left(bt + c + \frac{\pi}{2}\right);$$

this implies that f can be written in the form $a \sin(bt + c)$ for $a \neq 0$, $b \neq 0$, and $c \in \mathbf{R}$.

If $a < 0$, so that $a = -|a|$, then

$$a \sin(bt + c) = -|a| \sin(bt + c) \stackrel{(2)}{=} |a| \sin(bt + c + \pi);$$

thus, f is in the form $A \sin(bt + c)$ for $A > 0$, $b \neq 0$, and $c \in \mathbf{R}$.

If $b < 0$, so that $b = -|b|$, then use the fact that $\sin(-t) = -\sin t$ for all t , so that

$$\begin{aligned} A \sin(bt + c) &= A \sin(-|b|t + c) = A \sin[(-1)(|b|t - c)] \\ &= -A \sin(|b|t - c) \stackrel{(2)}{=} A \sin(|b|t - c + \pi). \end{aligned}$$

As a result, f can be put in the form $A \sin(\omega t + c)$ for $A > 0$, $\omega > 0$, and $c \in \mathbf{R}$.

If $c = \pi$, then

$$A \sin(\omega t + \pi) = A \sin(\omega t - \pi).$$

If $|c| > \pi$, then $|c| = k\pi + \phi$ for $0 \leq \phi < \pi$, for a positive integer k .

If $c \geq 0$, so that $|c| = c$, and k is an even integer, then

$$A \sin(\omega t + c) = A \sin(\omega t + |c|) = A \sin(\omega t + k\pi + \phi) = A \sin(\omega t + \phi) .$$

If $c \geq 0$ and k is odd, then

$$A \sin(\omega t + c) = A \sin(\omega t + k\pi + \phi) \stackrel{(2)}{=} -A \sin(\omega t + \phi) \stackrel{(2)}{=} A \sin(\omega t + \phi - \pi) ;$$

since $0 \leq \phi < \pi$, it follows that $-\pi \leq (\phi - \pi) < 0$.

If $c < 0$, so that $c = -|c|$, and k is even, then

$$A \sin(\omega t + c) = A \sin(\omega t - |c|) = A \sin(\omega t - k\pi - \phi) = A \sin(\omega t - \phi) ;$$

and if $c < 0$ and k is odd, then

$$\begin{aligned} A \sin(\omega t + c) &= A \sin(\omega t - |c|) = A \sin(\omega t - k\pi - \phi) \\ &\stackrel{(2)}{=} -A \sin(\omega t - \phi) \stackrel{(2)}{=} A \sin(\omega t - \phi + \pi) . \end{aligned}$$

Since $0 \leq \phi < \pi$, it follows that $0 < -\phi + \pi \leq \pi$. The only way that $-\phi + \pi$ can equal π is if $\phi = 0$, in which case $f(t) = A \sin(\omega t - k\pi)$ for an odd integer k , so that $f(t) = A \sin(\omega t - \pi)$.

In all cases, f has been written in the form $A \sin(\omega t + \phi)$ for $A > 0$, $\omega > 0$, and $-\pi \leq \phi < \pi$.

*uniqueness of
the representation*

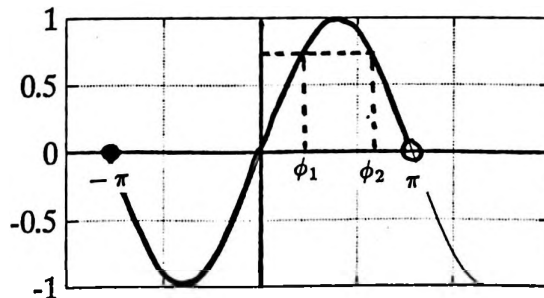
Let A_1, A_2, ω_1 and ω_2 be positive, and let ϕ_1 and ϕ_2 be in the interval $[-\pi, \pi)$. Two sinusoids with different amplitudes or frequencies cannot be equal, hence

$$A_1 \sin(\omega_1 t + \phi_1) = A_2 \sin(\omega_2 t + \phi_2) \implies (A_1 = A_2) \text{ and } (\omega_1 = \omega_2) .$$

If

$$A \sin(\omega t + \phi_1) = A \sin(\omega t + \phi_2) \quad \forall t, \quad -\pi \leq \phi_i < \pi ,$$

then for $t = 0$, one obtains $A \sin \phi_1 = A \sin \phi_2$, from which $\sin \phi_1 = \sin \phi_2$; and for $t = \frac{\pi}{2\omega}$, one obtains $\sin(\frac{\pi}{2} + \phi_1) = \sin(\frac{\pi}{2} + \phi_2)$. An analysis of the sketch below illustrates that the only way that both $\sin \phi_1 = \sin \phi_2$ and $\sin(\frac{\pi}{2} + \phi_1) = \sin(\frac{\pi}{2} + \phi_2)$ can be true, when ϕ_1 and ϕ_2 are in the interval $[-\pi, \pi)$, is for $\phi_1 = \phi_2$. ■



NOTATION $A \sin(\omega t + \phi)$	Whenever the notation $A \sin(\omega t + \phi)$ is used in this dissertation, it is assumed that $A > 0$, $\omega > 0$, and $-\pi \leq \phi < \pi$.
--	--

frequency of sinusoids The word *frequency* is often used in the context of sinusoids for *two different* types of frequencies. To prevent any possible confusion with regard to this word, the following definitions and notation are set forth.

DEFINITION <i>oscillation; cyclic frequency, f</i>	Suppose that a periodic function g has fundamental period P . Then, the function passes through all possible function values on any interval $[t, t + P)$, and the function is said to have made one <i>oscillation</i> . The <i>cyclic frequency</i> of g is denoted by f , and is defined by $f := \frac{1}{P} .$
---	--

DEFINITION <i>radian frequency, ω</i>	The <i>radian frequency</i> of $A \sin(\omega t + \phi)$ is the positive number ω .
---	--

These definitions imply that *cyclic* frequency can be computed for *any* periodic function, whereas radian frequency can be computed only for sinusoids. Note that cyclic frequency tells how many cycles the function makes in unit time.

relationship between cyclic and radian frequencies Since the function $A \sin(\omega t + \phi)$ has fundamental period $\frac{2\pi}{\omega}$, the relationship between radian frequency and cyclic frequency is given by $f = \frac{1}{(2\pi/\omega)}$, i.e.,

$$\omega = 2\pi f .$$

NOTATION ω, f	Whenever the variables ω and f are used in the context of sinusoids, they will always denote radian and cyclic frequency, respectively.
--------------------------------	--

Thus, many commonly-appearing sinusoids will take the form

$$\sin \omega t \quad \text{or} \quad \sin 2\pi f t .$$

Note that

$$\sin \frac{2\pi t}{P} = \sin 2\pi \left(\frac{1}{P}\right)t$$

has fundamental period P .

*trigonometric
formulas*

For the convenience of the reader, some important trigonometric identities are summarized next. For all real numbers t , α , and β ,

$$\sin(-t) = -\sin t$$

$$\cos(-t) = \cos t$$

$$\sin^2 t + \cos^2 t = 1$$

*addition
formulas*

$$\sin(\alpha \pm \beta) = \sin \alpha \cos \beta \pm \cos \alpha \sin \beta$$

$$\cos(\alpha \pm \beta) = \cos \alpha \cos \beta \mp \sin \alpha \sin \beta$$

*double-angle
formulas*

$$\sin 2t = 2 \sin t \cos t$$

$$\cos 2t = \cos^2 t - \sin^2 t = 1 - 2 \sin^2 t = 2 \cos^2 t - 1$$

*half-angle
formulas*

$$\sin \frac{t}{2} = \pm \sqrt{\frac{1 - \cos t}{2}}$$

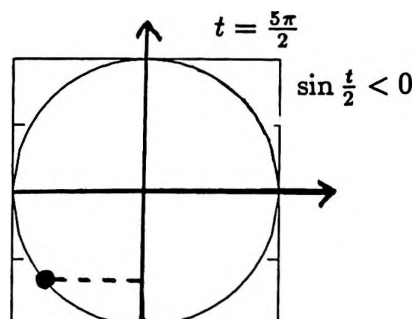
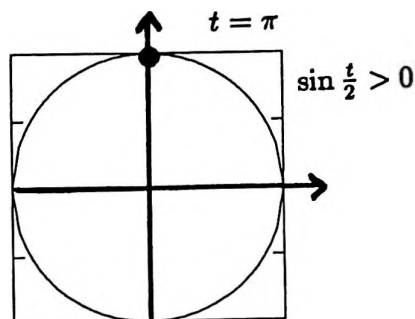
$$\cos \frac{t}{2} = \pm \sqrt{\frac{1 + \cos t}{2}}$$

In the half-angle formulas, the correct choice of sign is determined by the location of the point on the unit circle associated with $t/2$.

For example, letting $t = \pi$,

$$\sin \frac{\pi}{2} = +\sqrt{\frac{1 - \cos \pi}{2}} = 1 ,$$

since the point determined by $t/2$ has a positive y -coordinate.



Letting $t = \frac{5\pi}{2}$,

$$\sin \frac{5\pi}{4} = -\sqrt{\frac{1 - \cos \frac{5\pi}{2}}{2}} = -\frac{1}{\sqrt{2}},$$

since the point determined by $t/2$ has a negative y -coordinate.

*Fourier series
results
for continuous
functions*

The importance of sinusoids is largely due to the fact that *any* continuous periodic function can be well-approximated by a sum of sinusoids. The interested reader is referred to any real analysis text, say [Bar, 330–345], for proofs of the following classic results:

DEFINITION
*piecewise
continuous
periodic function*

Let g be a periodic real-valued function of one real variable, with fundamental period P . The function g is *piecewise continuous on \mathbb{R}* if g is defined and continuous on \mathbb{R} , except possibly for a finite number of points t_1, \dots, t_n in any interval of length P , at which g has both left and right hand limits,

$$g(t_i^-) := \lim_{h \rightarrow 0^+} g(t_i - h), \quad g(t_i^+) := \lim_{h \rightarrow 0^+} g(t_i + h),$$

where the notation $h \rightarrow 0^+$ means that h approaches 0 from the right-hand side.

DEFINITION
*Fourier series
for g*

Let g be a periodic real-valued function of one real variable. Suppose that g is piecewise continuous on \mathbb{R} , and has fundamental period P .

The *Fourier series of g* is the infinite sum

$$\frac{a_0}{2} + \sum_{k=1}^{\infty} \left(a_k \cos \frac{2\pi kt}{P} + b_k \sin \frac{2\pi kt}{P} \right), \quad (\text{CFS})$$

where the Fourier coefficients a_k and b_k are given by the formulas

$$a_k = \frac{2}{P} \int_P g(t) \cos \frac{2\pi kt}{P} dt, \quad b_k = \frac{2}{P} \int_P g(t) \sin \frac{2\pi kt}{P} dt.$$

The notation \int_P is used to mean that the integral may be evaluated over *any* interval of length P .

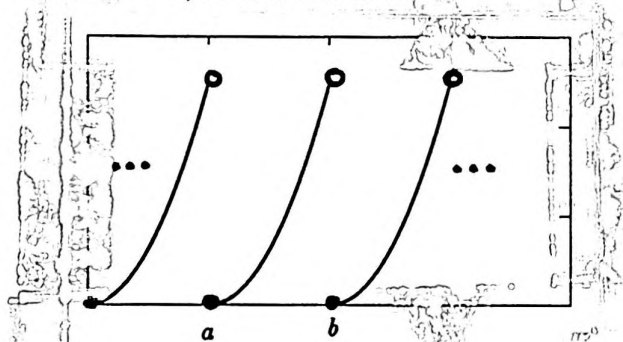
comments on
the definition

The label (CFS) stands for *continuous Fourier series*.

The requirement that g be piecewise continuous guarantees that the integrals defining a_k and b_k exist.

Since g has fundamental period P , and the functions $\cos \frac{2\pi kt}{P}$ and $\sin \frac{2\pi kt}{P}$ have a period P , the integrands in the integrals defining a_k and b_k have a period P . This is what makes it possible to integrate over *any* interval of length P .

Any function g defined on a finite interval $[a, b)$ can be extended periodically to \mathbb{R} , as illustrated below.



fundamental
period;
fundamental
frequency;
harmonics

Note that the arguments of the sinusoids in the Fourier series are of the form

$$\frac{2\pi kt}{P} = 2\pi(k \cdot \frac{1}{P})t,$$

so that the cyclic frequencies are multiples of $\frac{1}{P}$. In this context, P and $\frac{1}{P}$ are called the *fundamental period* and *fundamental (cyclic) frequency*, respectively, of the series. The multiples of the frequencies are called the *harmonics*.

The next two theorems describe the relationship between the function g and its Fourier series.

**Pointwise
Convergence
Theorem**

Let g be a periodic real-valued function of one real variable. Suppose that g is piecewise continuous on \mathbb{R} , and has fundamental period P . Suppose further that g has right and left-hand derivatives at $c \in \mathbb{R}$; that is, both

$$\lim_{h \rightarrow 0^+} \frac{g(c+h) - g(c^+)}{h} \quad \text{and} \quad \lim_{h \rightarrow 0^+} \frac{g(c-h) - g(c^-)}{h}$$

exist.

Then, the Fourier series for g converges to $\frac{1}{2}(g(c^-) + g(c^+))$ at c .

In particular, if g is continuous at c , then the Fourier series for g at c converges to $g(c)$.

Gibbs' phenomenon

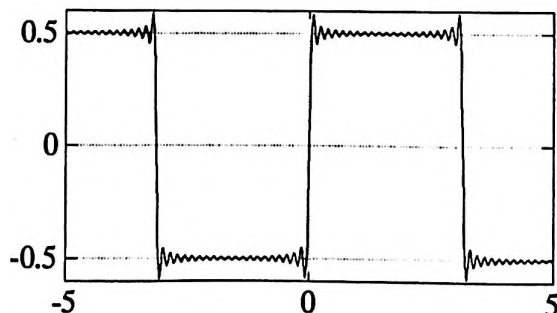
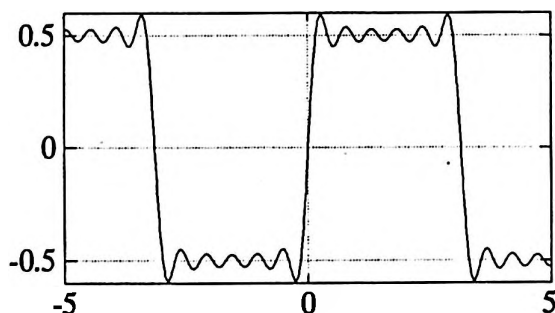
With only pointwise convergence, as guaranteed by the previous theorem, some undesirable types of behavior can occur. To illustrate what can 'go wrong', consider the function

$$g(t) = \begin{cases} -0.5 & \text{for } -\pi < t < 0 \\ 0.5 & \text{for } 0 < t < \pi. \end{cases}$$

Extend g periodically to all of \mathbf{R} , call this extension by the same name, and form its Fourier series. Observe that g meets the hypotheses of the Pointwise Convergence Theorem. Let c be *any* number in the interval $(0, \pi)$. Via pointwise convergence, the Fourier series for g must converge to $g(c) = 0.5$ at $x = c$. Therefore, if ϵ is *any* positive number, there exists a positive integer N such that whenever $n \geq N$,

$$\left| \left(\frac{a_0}{2} + \sum_{k=1}^n a_k \cos \frac{2\pi kc}{P} + b_k \sin \frac{2\pi kc}{P} \right) - 0.5 \right| \leq \epsilon,$$

where the a_k and b_k are the Fourier coefficients. Unfortunately, however, somewhere in the interval $(0, c)$, the truncated Fourier series will *always* take on a value that overshoots 0.5 by more than 0.089; and that undershoots 0.5 by more than 0.048 [HamD, 107–109], as illustrated below. This behavior, known as the *Gibbs' phenomenon* (after J. Willard Gibbs, who first publicized the effect but was not the first to publish it), is typical of the behavior of any truncated Fourier series near a jump discontinuity. An understanding of this behavior will become important in Chapter 3, when designing mathematical filters.



*uniform
convergence*

A much more desirable type of convergence is *uniform convergence*. Roughly, a sequence of functions $(g_i)_{i=1}^{\infty}$ converges uniformly to g if, given any ϵ -envelope about g , one can get all the functions g_n from the list to lie entirely within this ϵ -envelope, providing that $n \geq N$ for some sufficiently large positive integer N .

With slightly stronger hypotheses on g , one can actually get the Fourier series of g to converge uniformly to g :

**Uniform
Convergence
Theorem**

Let g be a periodic real-valued function of one real variable, defined on \mathbf{R} . Suppose that g is continuous with fundamental period P , and suppose that the derivative g' is piecewise continuous on \mathbf{R} . Then, the Fourier series for g converges uniformly to g on \mathbf{R} . That is, for every $\epsilon > 0$, there exists a positive integer N such that for all $n \geq N$ and for all $t \in \mathbf{R}$,

$$\left| \left(\frac{a_0}{2} + \sum_{k=1}^n a_k \cos \frac{2\pi kt}{P} + b_k \sin \frac{2\pi kt}{P} \right) - g(t) \right| \leq \epsilon .$$

The formulas for the coefficients a_k and b_k of the Fourier series can be derived by:

- *assuming* that a series of the form (CFS) exists and represents g ;

$$g(t) = \frac{a_0}{2} + \sum_{k=1}^{\infty} \left(a_k \cos \frac{2\pi kt}{P} + b_k \sin \frac{2\pi kt}{P} \right)$$

- multiplying both sides of this equation by $\cos \frac{2\pi mt}{P}$ (or $\sin \frac{2\pi mt}{P}$) for a fixed integer m ;
- integrating over one period; and
- using the following *orthogonality* properties of the sine and cosine: for all positive integers n and m ,

$$\int_P \cos\left(\frac{2\pi nt}{P}\right) \cos\left(\frac{2\pi mt}{P}\right) dt = \begin{cases} 0 & \text{if } n \neq m \\ \frac{P}{2} & \text{if } n = m \end{cases}$$

$$\int_P \sin\left(\frac{2\pi nt}{P}\right) \sin\left(\frac{2\pi mt}{P}\right) dt = \begin{cases} 0 & \text{if } n \neq m \\ \frac{P}{2} & \text{if } n = m \end{cases}$$

$$\int_P \sin\left(\frac{2\pi nt}{P}\right) \cos\left(\frac{2\pi mt}{P}\right) dt = 0 \quad \forall n, m .$$

These orthogonality properties cause most of the terms in the sum to vanish.

This same procedure will be followed in the derivation of the coefficients for the *discrete* Fourier series in Chapter 2.

The Fourier series also has the following desirable property:

THEOREM 1
a finite
Fourier series
minimizes the
mean-square error

[Weav, 113] Let g be a periodic real-valued function of one real variable. Suppose that g is piecewise continuous on \mathbf{R} , and has fundamental period P . If g is approximated by any finite series of the form

$$S_N(t) := \frac{a_0}{2} + \sum_{k=1}^N \left(a_k \cos \frac{2\pi kt}{P} + b_k \sin \frac{2\pi kt}{P} \right)$$

for real coefficients a_k and b_k , then the best approximation in terms of the mean-square error is obtained when the Fourier coefficients are used for a_k and b_k . That is, the mean-square error between S_N and g , defined by

$$\int_P (g(t) - S_N(t))^2 dt ,$$

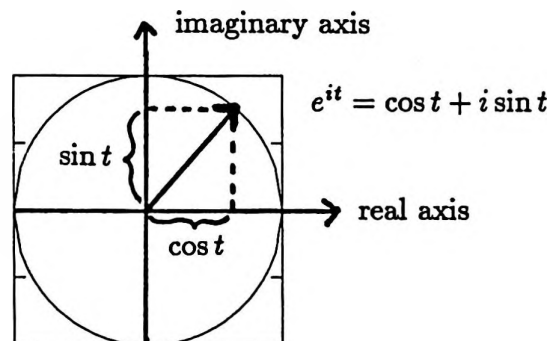
is minimized when S_N is the N^{th} partial sum of the Fourier series for g .

cosine and sine
as the real and
imaginary parts
of the complex
exponential
function

The *complex exponential function* is defined by

$$e^{it} := \cos t + i \sin t ,$$

for all real numbers t , and for $i := \sqrt{-1}$. Thus, e^{it} is a complex number, with real part equal to $\cos t$, and imaginary part equal to $\sin t$.



The Complex Plane

Here are some important consequences of this definition. Using the identities $\cos(-t) = \cos t$ and $\sin(-t) = -\sin t$, one obtains:

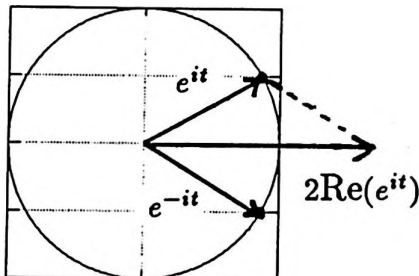
$$e^{it} = \cos t + i \sin t \quad (1)$$

$$e^{-it} = \cos t - i \sin t. \quad (2)$$

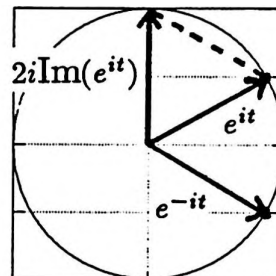
Adding (1) and (2), and dividing by 2, gives

$$\cos t = \frac{e^{it} + e^{-it}}{2}.$$

That is, $e^{it} + e^{-it} = 2 \cdot \operatorname{Re}(e^{it})$, where $\operatorname{Re}(e^{it})$ denotes the real part of e^{it} . The geometric interpretation of this identity is shown below.



$$e^{it} + e^{-it} = 2 \cdot \operatorname{Re}(e^{it})$$



$$e^{it} - e^{-it} = 2i \cdot \operatorname{Im}(e^{it})$$

Subtracting (2) from (1), and dividing by $2i$, gives

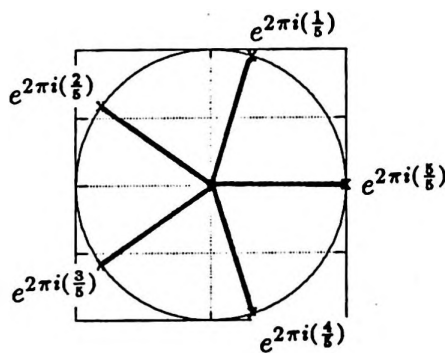
$$\sin t = \frac{e^{it} - e^{-it}}{2i}.$$

That is, $e^{it} - e^{-it} = 2i \cdot \operatorname{Im}(e^{it})$, where $\operatorname{Im}(e^{it})$ denotes the imaginary part of e^{it} . The geometric interpretation of this identity is shown above.

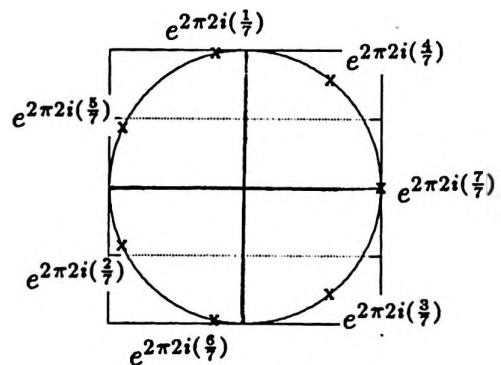
The identity in the next proposition reflects an important property of equal-spaced sampling from the complex exponential function. This identity will become important when discussing the discrete Fourier series in Chapter 2. Here's a geometric interpretation of the identity: let N be a positive integer greater than 1, and let k be a fixed integer between 1 and $N - 1$. Then, the complex numbers

$$\begin{aligned} &1, \\ &e^{2\pi ki(\frac{1}{N})}, \\ &e^{2\pi ki(\frac{2}{N})}, \\ &e^{2\pi ki(\frac{3}{N})}, \\ &\vdots, \\ &e^{2\pi ki(\frac{N-1}{N})}, \end{aligned}$$

divide $2\pi k$ revolutions of the unit circle in the complex plane into N equal arc lengths. When these complex numbers are summed, the result is 0. The result is illustrated below for two different combinations of k and N .



$$\begin{aligned} k &= 1 \\ N &= 5 \end{aligned}$$



$$\begin{aligned} k &= 2 \\ N &= 7 \end{aligned}$$

Proposition 2 Let $N > 1$ be a positive integer, and let k be a fixed integer between 1 and $N - 1$. Then,

$$\sum_{n=0}^{N-1} e^{2\pi kni(\frac{1}{N})} = 0 \quad \text{and} \quad \sum_{n=0}^{N-1} e^{-2\pi kni(\frac{1}{N})} = 0.$$

PROOF
of Proposition 2

Recall that, for all complex numbers $x \neq 1$,

$$1 + x + \cdots + x^{N-1} = \frac{1 - x^N}{1 - x}.$$

Let $x = e^{2\pi ki(\frac{1}{N})}$; since $k < N$, x is not equal to 1. Then,

$$\begin{aligned} \sum_{n=0}^{N-1} e^{2\pi ki(\frac{n}{N})} &= \sum_{n=0}^{N-1} x^n \\ &= \frac{1 - x^N}{1 - x} \\ &= \frac{1 - e^{2\pi ki(\frac{N}{N})}}{1 - x} = 0, \end{aligned}$$

since $e^{2\pi ki} = 1$ for all integers k . Letting $x = e^{-2\pi ki(\frac{1}{N})}$ and repeating the argument completes the proof. ■

sums of
sinusoids

Next, some questions concerning arbitrary sums of sinusoids are addressed.

In Question 3, Section 1.4, two periodic functions, each with fundamental period 4, were summed to yield a periodic function with fundamental period 2. Thus, the period of a sum certainly need not be the least common multiple of the summands. However, the following theorem, from [C&P, p. 35], shows that sinusoids are considerably nicer in this respect.

THEOREM 2 Let P_1, P_2, \dots, P_m be distinct positive integers, and let

$$g(t) = \sum_{k=1}^m a_k \cos \frac{2\pi t}{P_k} + b_k \sin \frac{2\pi t}{P_k},$$

where a_k and b_k are real numbers, not simultaneously zero ($a_k^2 + b_k^2 > 0$). Then, g is periodic, and has fundamental period equal to the least common multiple of the integers P_1, \dots, P_m .

EXAMPLE
actual period
versus
apparent period

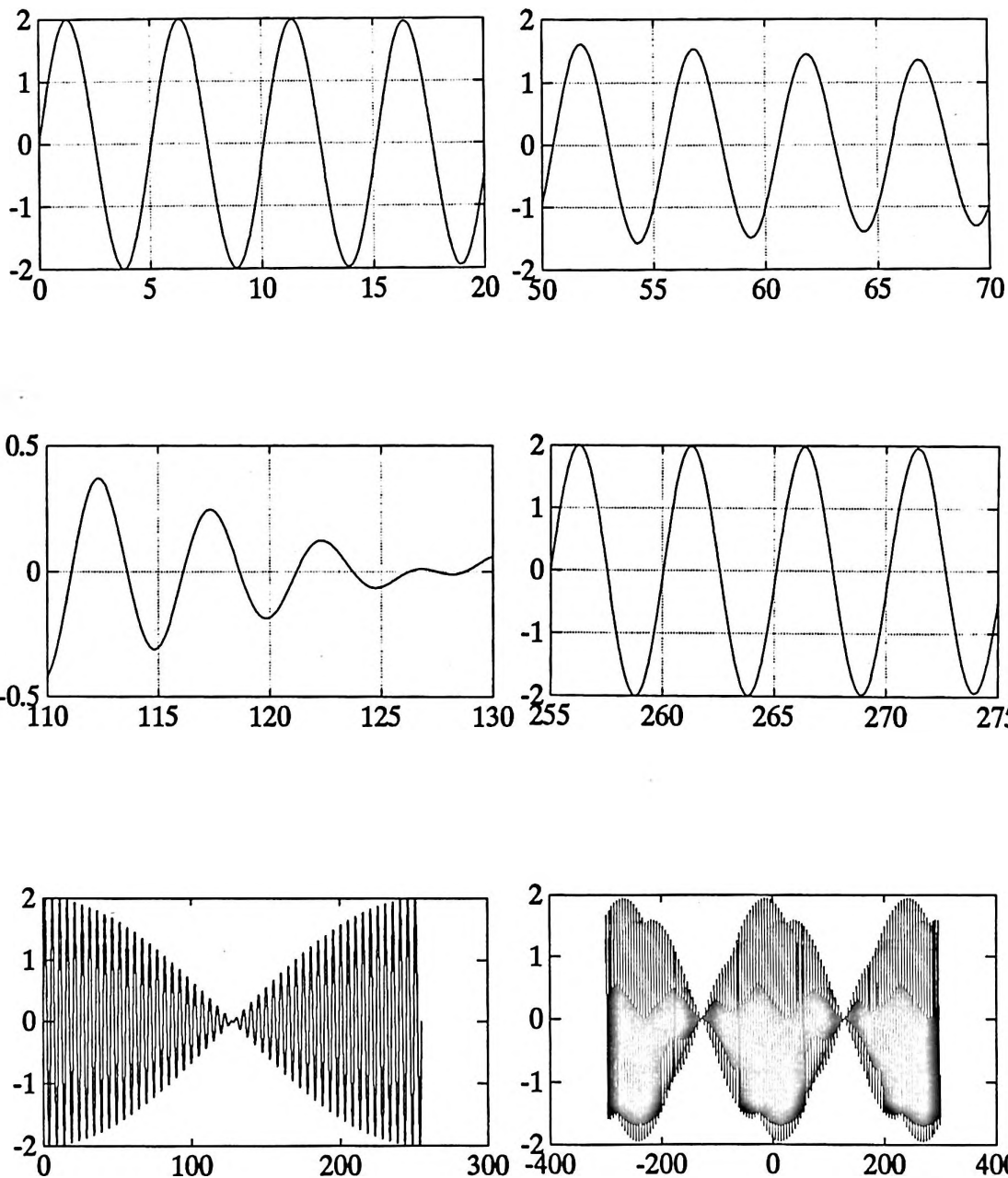
Consider the sum

$$g(t) = \sin 2\pi\left(\frac{1}{5}\right)t + \sin 2\pi\left(\frac{1}{5.1}\right)t.$$

Using Lemma 3 in Section 1.4, scaling g by $s = 10$ produces the scaled function

$$g_s(t) = \sin 2\pi\left(\frac{1}{50}\right)t + \sin 2\pi\left(\frac{1}{51}\right)t.$$

Using Theorem 2, the scaled function g , has fundamental period equal to the least common multiple of 50 and 51, i.e., $50 \cdot 51 = 2550$. Then, using Lemma 3 in Section 1.4, g has fundamental period $\frac{2550}{10} = 255$. The first graph below shows that the function g certainly 'appears' to have period 5. The remaining graphs illustrate that g actually has period 255.



'apparent'
sinusoids

It is interesting to note that when two sinusoids with close amplitudes A and $A + \Delta A$, close fundamental periods P and $P + \Delta P$, and close phase shifts ϕ and $\phi + \Delta\phi$ are summed, then the resulting function 'appears to be' (at least initially) a sinusoid with period P , amplitude $2A$, and phase shift ϕ . The following MATLAB function can be used to 'play with' this observation:

**MATLAB
implementation**

*sums of
sinusoids*

Let k be a positive integer, and let

$$A > 0, \quad P > 0, \quad -\pi \leq \phi < \pi$$

$$\Delta A \geq 0, \quad \Delta P \geq 0, \quad \Delta\phi \geq 0.$$

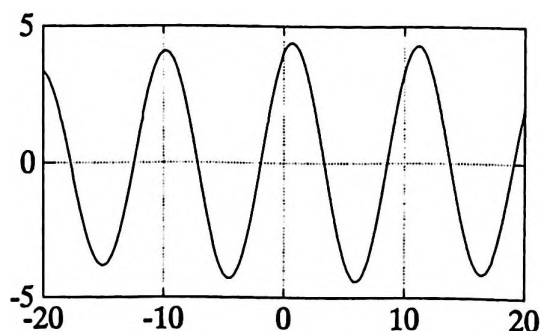
The following MATLAB function takes the inputs $k, A, P, \phi, \Delta A, \Delta P, \Delta\phi$, and plots a function y , where

$$y = \sum_{n=0}^{k-1} (A + n\Delta A) \sin\left(2\pi\left(\frac{1}{P + n\Delta P}\right)t + (\phi + n\Delta\phi)\right).$$

An example of the use of this function is given.

```
function [t,y] = testsum(k,A,P,phi,dinA,dinP,dinphi)
% k      = number of sinusoids in sum
% A      = amplitude of first sinusoid
% P      = fundamental period of first sinusoid
% phi    = phase shift of first sinusoid
% dinA   = difference in amplitude between sinusoids
% dinP   = difference in period between sinusoids
% dinphi = difference in phase shift between sinusoids
% The sum is plotted over an interval 4P.
t = [-2*P:(P/100):2*P];
y = A*sin(2*pi*(1/P)*t + phi);
for n = 1:(k-1)
    y = y + (A+n*dinA)*sin(2*pi*(1/(P+n*dinP))*t + phi + n*dinphi);
end
plot(t,y) =
endfunction
```

```
[t,y] = testsum(2,2,10,pi/3,.2,.5,pi/30)
```



aliasing

The last topic to be considered in this section is that of equally-spaced sampling from sinusoids, which can give rise to a phenomena known as *aliasing*.

The English usage of the word 'alias' as a noun means an *assumed name*. Generalize this usage a bit to mean an *assumed appearance*. With this generalization, the name 'aliasing' to describe the following phenomenon should seem very appropriate.

Anyone who has watched Westerns on television and focused on the wagon wheels has observed that as these wheels turn faster and faster, they may actually *appear* to slow down, or even to stop or reverse direction, as the wagon's speed increases. This *assumed appearance* (alias!) is due to equally-spaced sampling of the pictures in time, as described below.

*a circular
spinning wheel*

Imagine a circular wheel with one spoke which is spinning at some uniform rate, say, once every T seconds, in a dark room. If one was to flash a light precisely every T seconds (or any multiple of T seconds) then one would see the spoke in the *same position* each time, and perhaps perceive that the wheel was not spinning at all.

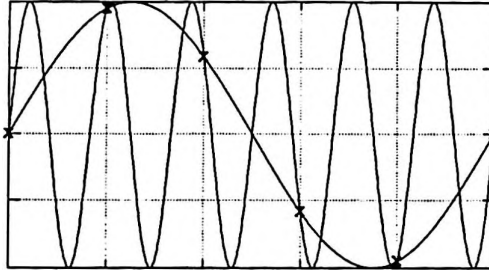
Now let S be a *little bit less* than T , and flash the light every S seconds. The first time the light flashes, the spoke is seen in some position. Then, S seconds later, the spoke has not quite reached that same position, so it appears to have *backed up* a little bit. Another S seconds, and it appears to have *backed up* a bit more. Thus it *appears* to be moving backwards slowly (which is, of course, not the case at all!)

One more time. Now let S be a *little more* than T . The first time the light flashes, the spoke is seen in some position. Then, S seconds later, the spoke has gone around once and a *little bit* more, so it appears to have moved forward slightly. Continue. The spoke *appears* to be moving forward at a rate that is much slower than the actual rate.

In the previous scenarios, it is said that *one frequency has been 'aliased' into another frequency* due to the process of taking equally spaced samples.

*aliasing of
sinusoids*

The same kind of aliasing can occur when one takes equally-spaced samples from sinusoids. Consider for example the sample data points below.



These sample points could have come from either of the sinusoids shown.

*complete
description
of all sinusoids
that pass through
the same
equally-spaced
sample points*

It is possible, in fact, to completely describe all the sinusoids that will pass through the same equally-spaced sample points.

Let

$$A \sin(2\pi f t + \phi) ,$$

be any sinusoid with amplitude A and cyclic frequency f .

Suppose that data samples are taken at intervals of time T ; that is, at times kT , for all integers k . Let m denote *any* integer, so that mk is also an integer. Then,

$$\begin{aligned} A \sin(2\pi f(kT) + \phi) &= A \sin(2\pi kTf + \phi) \\ &= A \sin(2\pi kTf + 2\pi mk + \phi) \\ &= A \sin(2\pi kT(f + \frac{mk}{kT}) + \phi) \\ &= A \sin(2\pi kT(f + \frac{m}{T}) + \phi) \end{aligned}$$

Comparing the sinusoids

$$A \sin(2\pi f(kT) + \phi)$$

and

$$A \sin(2\pi(f + \frac{m}{T})(kT) + \phi)$$

shows that the sinusoids $A \sin(2\pi f t + \phi)$ and $A \sin(2\pi(f + \frac{m}{T})t + \phi)$ are *indistinguishable* for every integer m with respect to samples taken at intervals of time T .

The final theorem of this section is the famous Sampling Theorem [S&H, 45–49], which relates sampling rates to truncated Fourier series:

**SAMPLING
THEOREM**

Let f be a sum of harmonically related sinusoids,

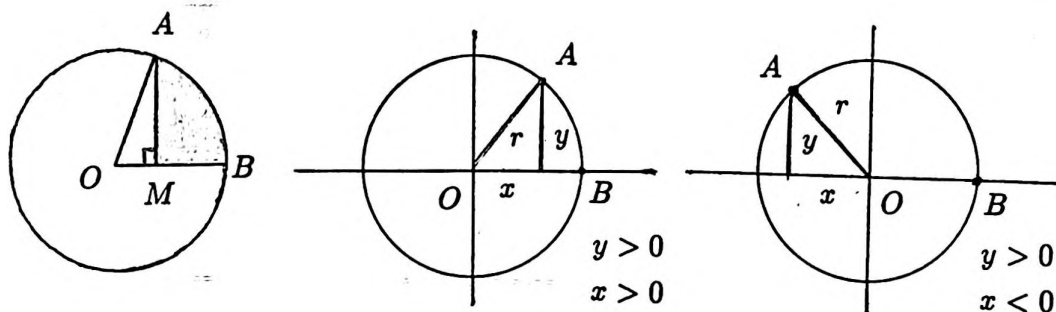
$$f(t) = \frac{a_0}{2} + \sum_{k=1}^K \left(a_k \cos \frac{2\pi kt}{P} + b_k \sin \frac{2\pi kt}{P} \right) .$$

To be able to recover f exactly, it is necessary to sample at a rate *greater* than twice the highest harmonic number K . That is, letting N denote the number of (equally-spaced) samples in a fundamental period P , one must have $N > 2K$.

1.7 Historical Contributions in the Search for Hidden Periodicities

The purpose of this section is to highlight the historical developments in the *search for hidden periodicities*. In so doing, it provides a gentle introduction to some important ideas that will be discussed in greater detail throughout the dissertation. The presentation is chronological.

The debut of the sine function. The ‘triangle’ approach to trigonometry appeared long before the emergence of the sine as a function of a continuous variable. In about 160 B.C. the Greek astronomer Hipparchus investigated the complete measurement of triangles from certain data. Originally, one spoke not of the *sine of an angle* but instead of the *sine of an arc*, as follows: consider the circle below, with center O and arc AB . The length of the line segment AM was defined to be *the sine of the arc AB* [New, 18]. Why the word ‘sine’? Observe that the highlighted shape resembles a stretched bow; indeed, the word *sine* derives from the Latin *sinus*, which means *a bend* [Oxf]. The word *cosine* uses the Latin prefix ‘co’, meaning *in the company with*, so that ‘cosine’ means *in the company with the sine*.



The view of sine and cosine as functions of a continuous variable is more advanced; it requires the use of a coordinate system to determine the appropriate signs (plus or minus) of the trigonometric functions for angles greater than ninety degrees. This tool was not available until René Descartes (1596–1650) introduced what is now called the *Cartesian coordinate system*. With this graphical device, the center of the circle described above could be placed at the origin of the coordinate system, with segment OB along the x -axis (see above). The ratio $\frac{\text{length of arc } AB}{\text{length of } OA}$ was known to be invariant under circle size, and is now known as the radian measurement of the angle. Call this ratio u . Then, the ratios $\frac{\text{length of } AM}{\text{length of } OA} := \frac{y}{r}$ and $\frac{\text{length of } OM}{\text{length of } OA} := \frac{x}{r}$ were defined as the *sine of u* and *cosine of u* , respectively, and these definitions could be preserved by paying attention to the signs of x and y in the various quadrants. Thus, sine and cosine became separated from a study of triangles; the trigonometric functions had emerged [New, 39].

1500s–1600s: Interest in naturally-occurring periodic phenomena.

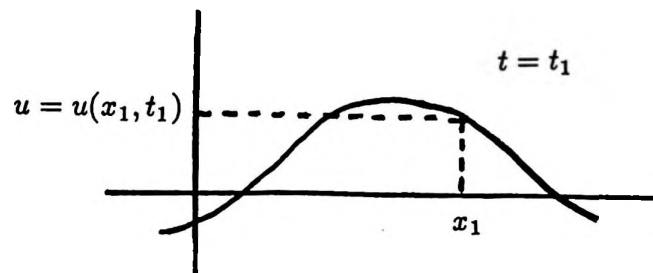
During the 16th and 17th centuries, investigations into many naturally occurring physical problems helped to forward the interest in periodicity. Johannes Kepler (1571–1630) derived the three basic laws of planetary motion that bear his name; for example, his *law of periods* states that the square of the period of a planet's orbit about the sun is proportional to the cube of the planet's mean distance from the sun [H&R, 262]. Galileo (1564–1642) observed that the periodic oscillation of a pendulum was independent of the mass of the suspended weight. Issac Newton (1642–1727) explained sound in terms of periodic waves [New, 412]. But the basis for modern theories of periodicity was to come in the 1700s, with the development of Fourier series.

1700s: Developments due to a vibrating string. Fourier analysis, which involves the representation of a function as a sum of sinusoids, is fundamental to many modern methods concerned with detecting hidden periodicities. Thus, early developments by Jean LeRond d'Alembert (1707–1783), Leonhard Euler (1707–1783), Daniel Bernoulli (1700–1782), Joseph Louis Lagrange (1736–1813), and Jean-Baptiste-Joseph Fourier (1768–1830) are important. This period of time is well documented, and the interested reader is referred to [Jef], [Car, 1–19], and [Gon, 427–441] for additional information.

It was interest in the partial differential equation (PDE) now known as the *one-dimensional homogeneous wave equation*,

$$u_{tt} - c^2 u_{xx} = 0,$$

that led to the first investigations into the trigonometric expansion of a function. A solution $u(x, t)$ of this PDE gives the displacement of a particle at distance x on a vibrating string at time t .



In 1746, the French mathematician d'Alembert obtained the solution

$$u(x, t) = \frac{1}{2}[f(x + ct) + f(x - ct)]$$

for an infinite length string, with zero initial vertical velocity and initial configuration $u(x,0) = f(x)$. The constant c depends on the string material. This solution was arrived at by a change of variables, and required no trigonometric series results. Thus, *existence* of a solution (at least in the infinite length case) was established; the controversy that waged over the next couple decades concerned the *form* of the solution.

In 1753, Bernoulli [Ber, 173] claimed that the solution $u(x,t)$ for a finite string of length l is expressible in the form

$$u(x,t) = \sum_{k=1}^{\infty} A_k \sin \frac{k\pi x}{l} \cos \frac{k\pi ct}{l} .$$

This is the form obtained when one pursues the modern ‘separation of variables’ technique. When $t = 0$, this reduces to an expression for the initial string displacement in the form

$$u(x,0) = \sum_{k=1}^{\infty} A_k \sin \frac{k\pi x}{l} .$$

Bernoulli believed that *any* initial displacement could be written in this way. Euler and d’Alembert disagreed: they argued that since sums of sine functions are both periodic and odd, any function which did *not* possess both these properties could not be expanded in such a way. Thus began the animated debate over the representation of a function as a sum of sinusoids.

In 1777, Euler showed that if a function $f(x)$ could be represented in the form

$$\frac{1}{2}a_0 + \sum_{k=1}^{\infty} (a_k \cos \frac{k\pi x}{l} + b_k \sin \frac{k\pi x}{l})$$

then the coefficients must be given by

$$a_k = \frac{1}{l} \int_{-l}^l f(t) \cos \frac{k\pi t}{l} dt , \quad b_k = \frac{1}{l} \int_{-l}^l f(t) \sin \frac{k\pi t}{l} dt .$$

However, it was Fourier, in his investigations of the PDE modeling the temperature distribution in a metal plate, who first applied these integral formulas to the representation of a completely arbitrary function. In particular, Fourier allowed the function to have different analytical expressions in different parts of the interval. Also, he was the first to recognize that the use of sine series was not restricted to odd functions; and the use of cosine series was not restricted to even functions.

Further work by Cauchy (1789–1857), Dirichlet (1805–1859), Weierstrass (1815–1897), Riemann (1826–1866), Jordan (1838–1922), Cantor (1845–1918), and Lebesgue (1875–1941) passed into the realm of the pure mathematician. These

people made precise the notions of convergence, uniform convergence, functions, and integration which had been somewhat loose in earlier work, and thus laid the foundation for all modern analysis.

Backing up a bit in time, one of the first methods for detecting hidden periodicities was used by Lagrange in 1772 to analyze the orbit of a comet [Lag]. Lagrange had a good deal of interest in rational functions; one may recall the *Lagrange Interpolation Formula* [S&B, 39]

$$P(x) = \sum_{i=0}^n f_i \prod_{\substack{k=0 \\ k \neq i}}^n \frac{x - x_k}{x_i - x_k}$$

for the unique polynomial passing through $n + 1$ distinct data points $(x_i, f_i)_{i=0}^n$. Thus it should not be surprising that Lagrange's method used rational functions to analyze the data. Lagrange's method was improved by Dale [Dale, 628] in 1778.

The next major contributions appeared in the mid 1800s, and are discussed in some detail in the following two sections.

1850–1900. Tabular methods for detecting hidden periodicities.

In 1847, Buys-Ballot introduced a tabular method for detecting hidden periodicities that was used in his analysis of temperature variations [B-B, 34]. These methods were further developed by Strachey in 1877 [Str] and Stewart and Dodgson in 1878 [S&D]. The next example illustrates the 'flavor' of these tabular methods; note the similarity to the reshaping techniques discussed in Section 1.3. For more details, the reader is referred to [W&R, 345–362].

The scatter plot in Figure 1a shows sixty unit-spaced data points; consecutive points have been connected by line segments in Figure 1b. This data was generated by the function

$$f(x) = 2 \sin\left(\frac{2\pi x}{5}\right) + \cos\left(\frac{2\pi x}{13}\right) + \text{noise} ;$$

thus, it is a sum of a period 5 sine curve of amplitude 2, a period 13 cosine curve of amplitude 1, and noise, generated by the uniform distribution on $[-.2, .2]$. Now that you are privy to the generator of the data, forget it. The question is: "With only the sixty data points at hand—is it possible to recover any information regarding sinusoidal components?"

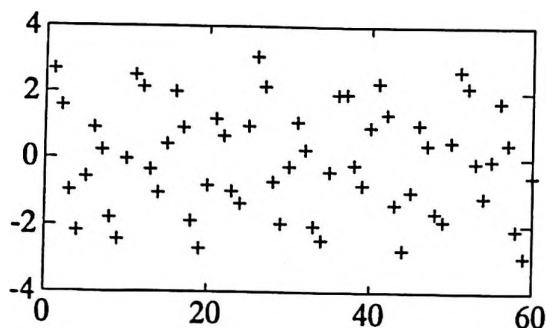


Figure 1a. A scatterplot of a function with two sinusoidal components, and noise.

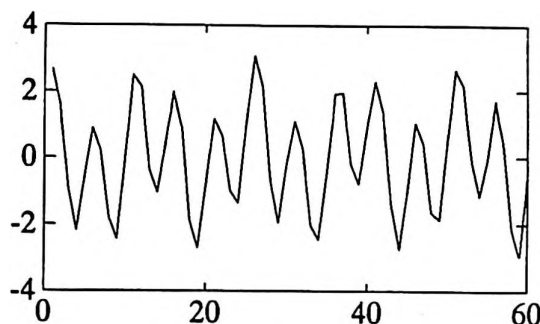


Figure 1b. The data at left, with consecutive points connected by line segments.

The approach taken is roughly as follows: for a given period p , get a number that reflects 'how much' this period is present in the data. Repeat the procedure for lots of values of p and plot the results. The graph should peak at periodicities that are present in the data.

For example, let us test the data of Figure 1 for period 5. Let (i, y_i) denote the i^{th} data point. First the data $\{y_1, y_2, y_3, \dots, y_{60}\}$ must be arranged into rows of 5; there will be $60/5$ such rows, as shown.

y_1	y_2	y_3	y_4	y_5					
y_6	y_7	y_8	y_9	y_{10}	2.6752	1.5625	-0.9835	-2.1850	-0.5746
y_{11}	y_{12}	y_{13}	y_{14}	y_{15}	0.8846	0.2124	-1.7917	-2.4429	-0.0581
y_{16}	y_{17}	y_{18}	y_{19}	y_{20}	2.4821	2.1295	-0.3725	-1.0633	0.3948
y_{21}	y_{22}	y_{23}	y_{24}	y_{25}	1.9896	0.8957	-1.8885	-2.7009	-0.8325
y_{26}	y_{27}	y_{28}	y_{29}	y_{30}	1.1644	0.6578	-0.9935	-1.3676	0.9659
y_{31}	y_{32}	y_{33}	y_{34}	y_{35}	= 3.0662	2.1659	-0.7025	-1.9626	-0.2602
y_{36}	y_{37}	y_{38}	y_{39}	y_{40}	1.0849	0.2577	-2.0439	-2.4542	-0.4085
y_{41}	y_{42}	y_{43}	y_{44}	y_{45}	1.9215	1.9367	-0.2011	-0.8008	0.9461
y_{46}	y_{47}	y_{48}	y_{49}	y_{50}	2.2993	1.3488	-1.3763	-2.7415	-0.9964
y_{51}	y_{52}	y_{53}	y_{54}	y_{55}	1.0378	0.4182	-1.6351	-1.8716	0.5118
y_{56}	y_{57}	y_{58}	y_{59}	y_{60}	2.6542	2.1702	-0.1311	-1.1704	-0.0552
					1.7094	0.4289	-2.1400	-2.9454	-0.5539

Next, sum the entries in each column and divide by the number of rows, thus obtaining the average value of each column. Let M_i denote the average of the i^{th} column. For the data above, we obtain the five numbers

$$M_1 = 1.9141, \quad M_2 = 1.1820, \quad M_3 = -1.1883, \quad M_4 = -1.9755, \quad M_5 = -0.0767.$$

Finally, the standard deviation of the M_i ,

$$\text{standard deviation} = \sqrt{\frac{1}{5-1} \sum_{i=1}^5 \left(M_i - \left(\frac{M_1 + \dots + M_5}{5} \right) \right)^2}$$

is the number used to indicate 'how much' of period 5 is present in the data.

Of course, a reasonable question at this point is: Why *this* number? Here's the rationale: *Assume* that the data is composed of sinusoidal components and random noise. *If* there is a component of period 5, then each column will contain the same phase of this component, as Figure 2a illustrates. Averaging the column gives the correct amplitude of *this component*. But what of the other components—periodicities other than 5, and noise? Since positive and negative deviations will tend to cancel each other, the averaging process will tend to annul these other components. So, the numbers M_1 through M_5 should be, approximately, as shown in Figure 2b (which they are!), and the standard deviation measures the spread of these values about their mean. Of course, the larger the amplitude of the component, the larger the spread about the mean.

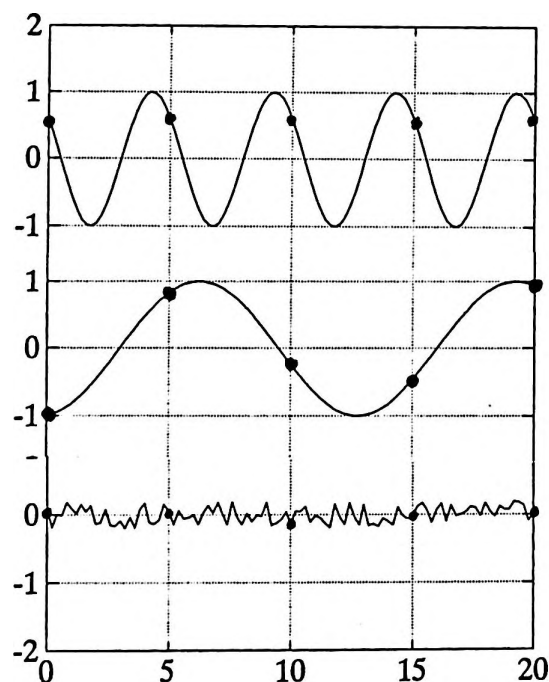


Figure 2a. Three components: period 5, period 13, and noise. All 'dotted' points will contribute to the same column in a tabular test for period 5.

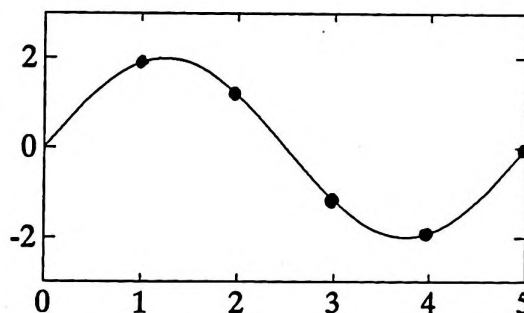


Figure 2b.

Figure 3 shows the results of this test for periods up to 20. Indeed, a peak occurs at 5 (and multiples of 5). There is also a peak at 13, albeit less noticeable; partly because the amplitude of the cosine component is only half that of the sine component.

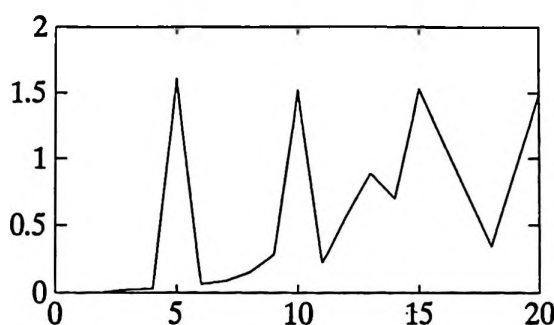


Figure 3. The 'period detector number' versus period. Note the peaks at 5 (and multiples of 5) and 13.

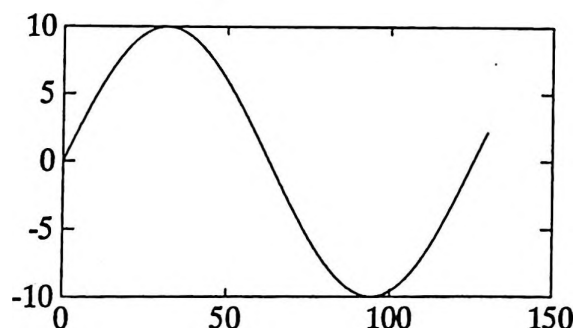
About the same time that this tabular method was being pursued, Stokes [Sto] presented an integral approach to the problem. In 1879, he suggested that to test a function $f(x)$ for period $\frac{2\pi}{n}$, one might do well to multiply by a sinusoid with this period, $\sin nx$, and then investigate the integral

$$\int f(x) \sin nx \, dx .$$

For if $f(x)$ has a component $A \sin n'x$, then this integral will involve (among other things)

$$\int (A \sin n'x) \sin nx \, dx = \frac{A \sin(n' - n)x}{2(n' - n)} - \frac{A \sin(n' + n)x}{2(n' + n)} ;$$

the important observation is that if n' is close to n , then the sinusoid $\frac{A \sin(n' - n)x}{2(n' - n)}$ has an amplitude of large magnitude and a long period, and should be easily detectable. For example, taking $A = 1$, $n' = 0.65$ and $n = 0.60$ yields the sine curve of amplitude $\frac{1}{2(0.05)} = 10$ shown below.



The graph of the sinusoid

$$\frac{A \sin(n' - n)x}{2(n' - n)}$$

when $A = 1$, $n' = 0.65$ and $n = 0.60$.

1897: Schuster's Periodogram. In 1897, Schuster introduced the *periodogram* as a tool for detecting hidden periodicities [Sch], and he used it to investigate meteorological phenomena. The method is computationally intensive, which made

it difficult to apply practically until the advent of the electronic digital computer in 1946. A brief description and example follows.

Suppose there are N available data points $(1, y_1), \dots, (N, y_N)$. Consider an approximation to the data of the form

$$a_0 + \sum_{k=1}^K \left(a_k \cos \frac{2\pi k}{N} t + b_k \sin \frac{2\pi k}{N} t \right) .$$

It is desired to choose the $2K + 1$ coefficients $a_0, a_1, b_1, \dots, a_K, b_K$ so that a least-squares fit is obtained. Provided that $N \geq 2K + 1$, there is a unique solution to this problem, found by taking

$$\begin{aligned} a_0 &= \frac{1}{N} \sum_{n=1}^N y_n \\ a_k &= \frac{2}{N} \sum_{n=1}^N y_n \cos\left(\frac{2\pi kn}{N}\right), \quad k = 1, \dots, K \\ b_k &= \frac{2}{N} \sum_{n=1}^N y_n \sin\left(\frac{2\pi kn}{N}\right), \quad k = 1, \dots, K . \end{aligned}$$

This is the *discrete* analogue of the (continuous) Fourier series, and is called the 'discrete Fourier series corresponding to the data' [S&H, 18-22]. Note that the fundamental frequency is $\frac{1}{N}$; let $f_i := \frac{i}{N}$ denote the i^{th} harmonic. The *intensity at frequency f_i* is defined by $I(f_i)$, where

$$I(f_i) := \frac{N}{2} (a_i^2 + b_i^2) .$$

This intensity is measuring how much of the frequency f_i is present in the data. The *periodogram* is the graph of $I(f_i)$ versus f_i .

To illustrate, consider again the data of Figure 1. Figure 5a shows this data, superimposed with its discrete Fourier series. Since N is 60 in this example, K was taken to be 29, so that K is the largest integer satisfying $2 \cdot K + 1 \leq 60$. Figure 5b shows the periodogram, which indeed peaks at both $f = .2$ and $f \approx .08$ (corresponding to periods $1/.2 = 5$ and $1/.08 = 12.5$).

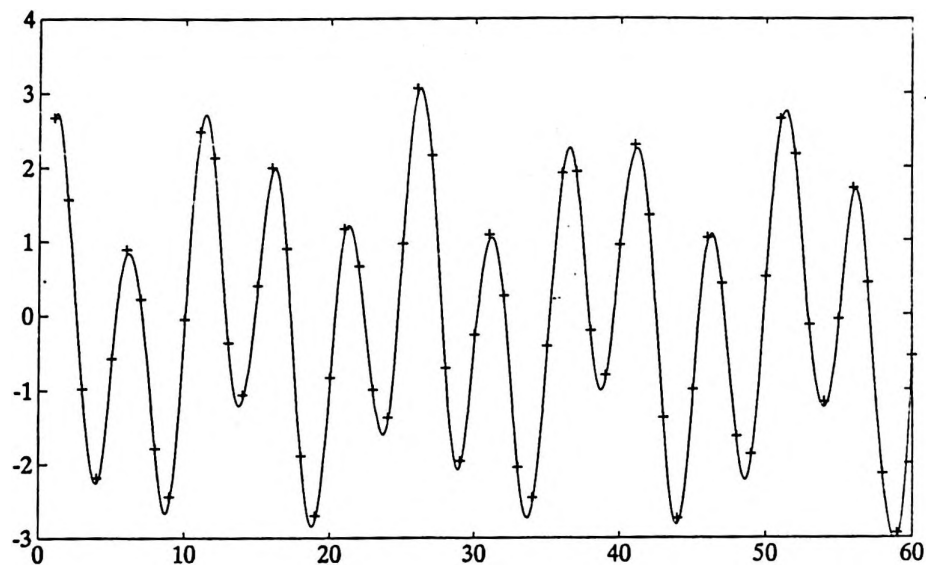


Figure 5a. A function having 2 sinusoidal components and noise, superimposed with its discrete Fourier series.

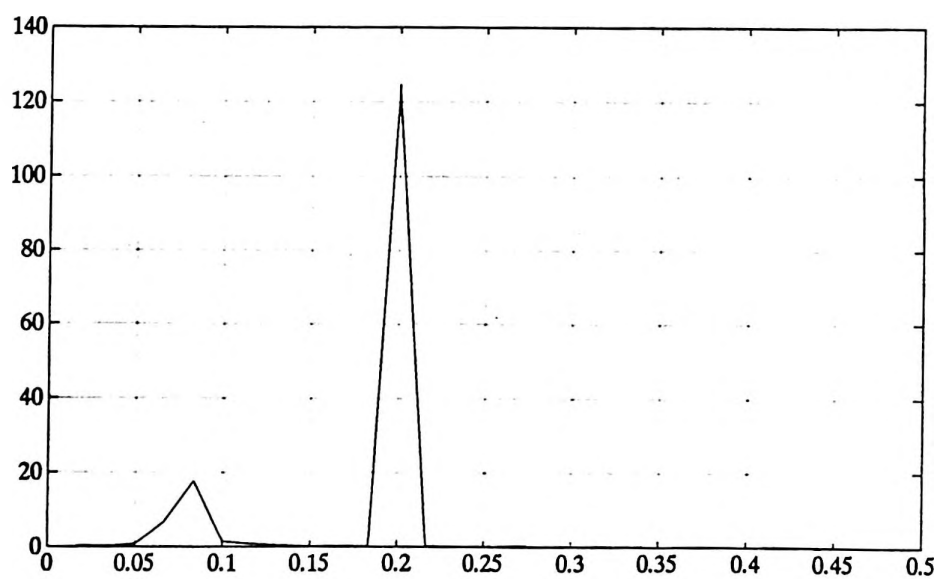


Figure 5b. The periodogram corresponding to the data in Figure 5a. Note the peaks, which detect the two sinusoidal components present in the data.

1900s: Time Series Analysis. The area of *time series analysis* really emerged—at least by this name—in the twentieth century. A *time series* is just an *ordered* list of numerical observations; this ‘ordering’ is usually provided by time, and hence the label. Up until about 1925, a time series was analyzed as if it was *deterministic*: that is, a ‘generator’ (function of time) was sought that could subsequently be used to ‘determine’ values of the series. If the predicted values did not agree with the series values, blame was placed on an improper determination of the generator, or noise.

In 1927, an important new viewpoint emerged, due primarily to work on sunspot numbers by Udny Yule. He was struck by the ‘apparent’ periodicity of the data, speckled by irregularities, and drew this analogy:

If we have a rigid pendulum swinging under gravity through a small arc, its motion is well known to be harmonic, that is to say, it can be represented by a sine or cosine wave, and the amplitudes are constant, as are the periods of the swing. But if a small boy now pelts the pendulum irregularly with peas the motion is disturbed. The pendulum will swing, but with irregular amplitudes and irregular intervals. The peas, in fact, instead of leading to behaviour in which any difference between theory and observation is attributable to an evanescent error, provide a series of shocks which are incorporated into the future motion of the system [Ken, 4].

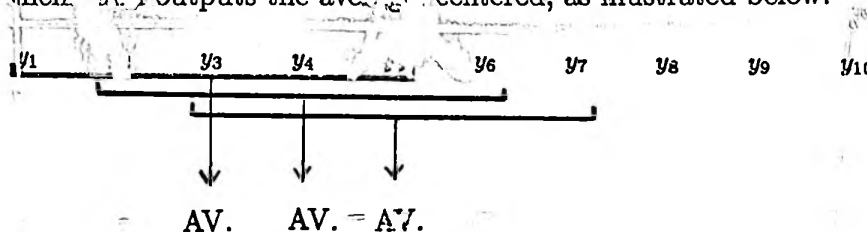
From this viewpoint was to come the notion of a *stochastic process*, an important sub-area of time series analysis.

The probabilistic and statistical theory of time series was developed during the 1920s and 1930s, and the concept of *spectrum* was introduced [Blo, 6]. Roughly speaking, *spectrum analysis* is used to mean methods that describe the *tendency* for oscillations of a given frequency to appear in the data, rather than the oscillations themselves [Blo, 2]; spectrum analysis combines the methods of Fourier analysis and statistics. The increased availability of computers in the 1950s and 1960s to carry out the extensive computations involved in spectrum analysis contributed to a wealth of developments in this decade: some important names are Grenander and Rosenblatt [G&R, 537–558], Parzen [Par], and Blackman and Tukey [B&T].

Signal Analysis. Parallel to the development of time series analysis is *signal analysis*. An important sub-area of signal analysis is *filter theory*—a filter being a device that transforms input in some specified way to yield a desired output [Joh, 1–3]. Filter theory began in 1915 when G.A. Campbell in America and Wagner in Germany independently invented the electric wave filter, a physical device to pass signals in a certain frequency band and suppress signals in all

other bands. By providing different filters for different frequency ranges, more than one telephone conversation can be carried out simultaneously on the same lines [Mab, 31].

It is interesting to note that in early filter theory literature, the word ‘filter’ usually referred to the actual hardware—resistors, capacitors, inductors, and such—providing the signal processing. It was not until the 1930s (and the advent of so-called ‘modern filter theory’) that developments by Cauer, Darlington and others led to the idea of a *mathematical filter* and its associated *transfer function*, which describes precisely what the filter does. As an example, consider the *five-point moving average filter* which takes five consecutive data points as input, averages them and outputs the average centered, as illustrated below.



By letting a_n denote the average of data values $y_{n-2}, y_{n-1}, y_n, y_{n+1}, y_{n+2}$ this filter can be described mathematically by

$$a_n := \frac{1}{5} \sum_{m=-2}^2 y_{n-m} . \quad (\dagger)$$

The *transfer function* then describes what the filter does to each input frequency, as follows: notation is simplified tremendously if we work with the complex exponential

$$e^{i\omega t} := \cos \omega t + i \sin \omega t$$

instead of real sines and cosines.

Suppose we ‘input’ a complex exponential of frequency ω ,

$$y(n) := e^{i\omega n} ,$$

into the moving average filter (\dagger). Here, function notation $y(n)$ has been used instead of subscript notation y_n , for convenience. The output obtained is

$$\begin{aligned} a_n &= \frac{1}{5} \sum_{m=-2}^2 e^{i\omega(n-m)} \\ &= \frac{1}{5} e^{i\omega n} \sum_{m=-2}^2 e^{-i\omega m} \\ &= \frac{1}{5} e^{i\omega n} (e^{2i\omega} + e^{i\omega} + 1 + e^{-i\omega} + e^{-2i\omega}) \\ &= \frac{1}{5} e^{i\omega n} (2 \cos 2\omega + 2 \cos \omega + 1) . \end{aligned}$$

The symmetry in the coefficients allowed use of the identity

$$2 \cos t = e^{it} + e^{-it}$$

to simplify the expression. If one defines

$$H(\omega) := \frac{1}{5}(2 \cos 2\omega + 2 \cos \omega + 1) ,$$

then it is seen that whenever $e^{i\omega n}$ is input to the filter, the corresponding output is $H(\omega)e^{i\omega n}$. The critical observation is that the *same function* $e^{i\omega n}$ *emerges from the filter*, except multiplied by $H(\omega)$.

In practice, it is usually more convenient to work with cyclic frequency f than radian frequency ω ; these are related by $\omega := 2\pi f$. In terms of f , the transfer function can be rewritten as

$$\tilde{H}(f) := H(\omega) = \frac{1}{5}(2 \cos 4\pi f + 2 \cos 2\pi f + 1) .$$

This transfer function $\tilde{H}: f \mapsto \tilde{H}(f)$ is shown in Figure 6a [Ham, 39]. Let's reiterate how it works: if a sinusoidal component $S(t)$ of frequency \tilde{f} is processed with this filter, the output will be $\tilde{H}(\tilde{f}) \cdot S(t)$. That is, the transfer function \tilde{H} allows one to take an input frequency \tilde{f} and 'transfer' to the output simply by multiplying by $\tilde{H}(\tilde{f})$. Observe that the transfer function of Figure 6a has a zero at $f = .2$, corresponding to period 5. Of course—if 5 consecutive unit-spaced points are averaged on a period 5 sinusoid, the result is zero! But if this filter acts on a sinusoid of period 13 ($f = 1/13 \approx .08$), this component should emerge at approximately .7 times its original amplitude. To illustrate, this five-point moving average filter was applied to the sample data considered twice earlier; recall it has an amplitude 2 component of period 5, an amplitude 1 component of period 13, and some noise. The filter output is shown in Figure 6b. As expected, the period 5 component has been entirely erased; all that remains is .7 times the amplitude 1 component!

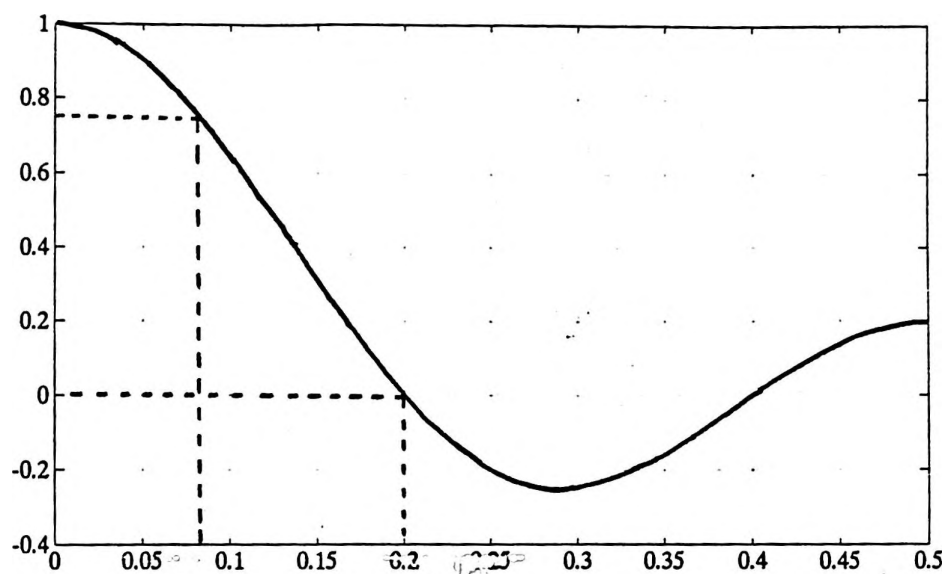


Figure 6a. The transfer function for a five-point moving average filter, $\tilde{H}(f)$ versus frequency f . Note that $\tilde{H}(.2) = 0$; a frequency .2 (period 5) sinusoid is completely suppressed.

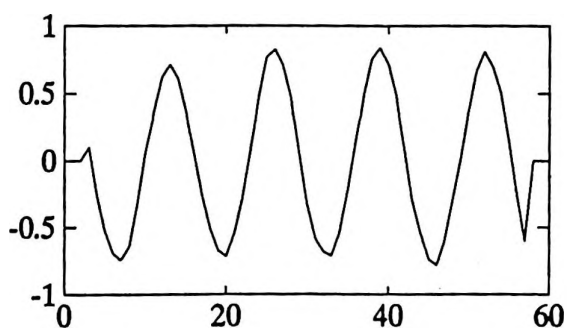


Figure 6b. The data of Figure 1a was filtered with a 5-point moving average filter, yielding this output. Observe that only the period-13 component remains, at .7 times its original amplitude.

Recent developments. Often, the characteristics of a signal vary with time; for example, a 'sinusoid' may be present, but with period gradually increasing from 5 to 7. A filter that tries to adapt to such changes is known as an *adaptive filter*. That is, the performance of the filter suggests changes in its coefficients [Ham, 277–278]. In 1960, R.E. Kalman introduced an algorithm since known as the *discrete Kalman filter* which is the main tool for such problems. His paper, titled *A new approach to linear filtering and prediction problems*, was published in the Journal of Basic Engineering, and is considered by many to be one of the most important results in its area in the last thirty years. From a mathematical viewpoint, the Kalman filter theorem is a beautiful application of functional analysis; for statistics students, an example of Bayesian statistics in action [Cat, Preface & p. 137].

In 1965, Cooley and Tukey [C&T, 297–301] introduced an algorithm since known as a *fast Fourier transform*. Coupled with the smaller, faster and less expensive computers that appeared in the 1970s, many practical applications of spectrum analysis on large data sets became possible. Techniques and methods that take advantage of increased computing powers will undoubtedly continue to advance the 'search for hidden periodicities'.

CHAPTER 2

'FITTING' A DATA SET WITH A FUNCTION

2.1 Random Behavior

The Turning Point Test

What is
'random'
behavior?

What is *random* behavior? For the purposes of this dissertation, a list is called *random* if the list entries are completely non-deterministic; i.e., if the occurrence of any observation in the list in no way influences the occurrence of any other observation. Should a list be random, then there is no sense in searching for deterministic components.

This section presents a test for random behavior, called the *turning point test*. The idea behind the test is this: if data is truly random, then certain behavior dictated by randomness is *expected*. A probabilistic comparison of properties of the actual data with what is *expected* should the data be truly random is used to support or deny the hypothesis of random behavior.

The turning point test is adapted from [Ken, 21–24]. Several other tests for randomness are discussed in the same cited section.

Some review of probability and statistics concepts is interspersed throughout this section. The reader is referred to [Dgty] for additional material. A MATLAB program to apply the Turning Point Test is included in this section.

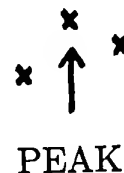
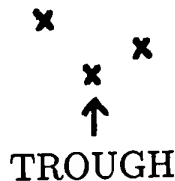
DEFINITION

neighbor;
peak;
trough;
turning point

Given an entry y_i in a list $(\dots, y_{i-1}, y_i, y_{i+1}, \dots)$, the adjacent entries y_{i-1} and y_{i+1} are called the *neighbors* of y_i .

An entry in a list is called a *peak* if it is strictly greater than each neighbor; and is called a *trough* if it is strictly less than each neighbor.

A list entry is a *turning point* if it is either a peak or a trough.



TURNING POINT TEST

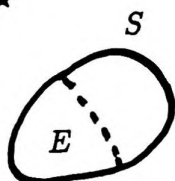
The *turning point test* is so named because it counts the number of turning points in a finite list of data, and compares this number with what is expected, should the data be truly random.

*hypothesis for
the turning
point test*

For the turning point test that is developed in this section, it is assumed that the entries in a finite list are allowed to come from an *interval* of real numbers. Since there are an infinite number of choices for any list entry, the probability that two entries in the list are equal is zero. In particular, the probability that there are equal adjacent entries is zero.

If a finite list contains a large number of duplicate values, then the hypothesis that the list entries come from some interval is probably unwarranted, and the turning point test as developed here will not apply.

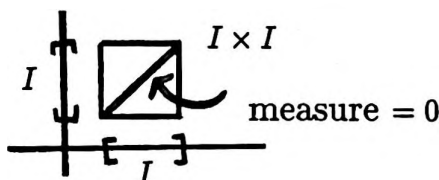
★



$$P(E) = \frac{1}{2}$$

An *event* E is a subset of a sample space S . The probability of E is determined by how much 'room' E takes up in S .

Suppose that two numbers are chosen from an interval I . The corresponding sample space is $I \times I$. The phrase 'the probability that there are equal adjacent entries is zero', means, precisely, that the measure of the set $\{(x, x) \mid x \in I\}$ (as a subset of $I \times I$) is zero.



Suppose that three numbers are chosen from an interval I . The corresponding sample space is $I \times I \times I$. The phrase 'the probability that there are equal adjacent entries is zero', means, precisely, that the set

$$S := \{(x, x, y) \mid x, y \in I\} \cup \{(x, y, y) \mid x, y \in I\},$$

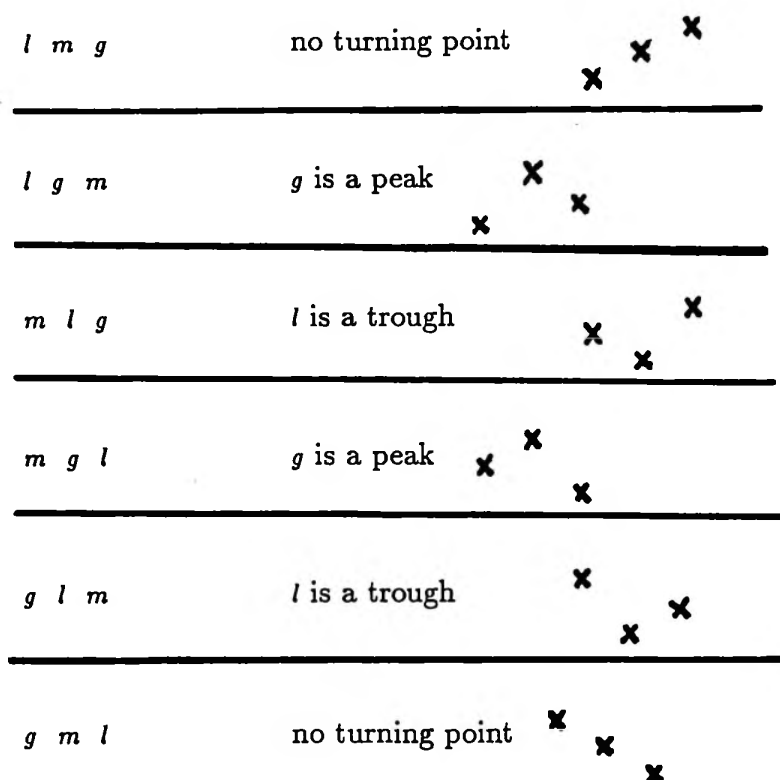
(as a subset of $I \times I \times I$), has measure zero. The set S is the intersection of two planes in \mathbb{R}^3 ($x = y$ and $y = z$) with the cube $I \times I \times I$; the resulting set is 'thin' in \mathbb{R}^3 .

The hypothesis that the entries in a finite list come from an *interval* justifies taking the sample space, in the following development of the turning point test, to be all possible arrangements of three *distinct* values. If, on the other hand, the entries come from some *finite* set, then the probability that there are equal adjacent entries is *nonzero*, and one would have to enlarge the sample space to account for these possible repeat values.

probability of
finding a
turning point
in a set of
3 distinct values

Let l , m , and g be three distinct real numbers, with $l < m < g$. The letters were chosen to remind the reader of this ordering: l for 'least', m for 'middle', and g for 'greatest'.

There are $3 \cdot 2 \cdot 1 = 6$ ways that these three numbers can be arranged in three slots. If the ordering is random, then these 6 possible arrangements will occur with equal probability. Only four arrangements yield a turning point:



Thus, the probability of finding a turning point in a set of three distinct real values is $\frac{4}{6} = \frac{2}{3}$.

S , sample space

Let S denote the sample space containing all possible arrangements of three distinct values, so that S contains the six 3-tuples investigated above.

a 'counting'
random variable,
 $C: S \rightarrow \{0, 1\}$

Define a 'counting' random variable $C: S \rightarrow \{0, 1\}$ via

$$C(a, b, c) := \begin{cases} 1 & \text{if } b \text{ is a turning point} \\ 0 & \text{otherwise} \end{cases}$$

$$E(C) = \mu_C = \frac{2}{3}$$

The probability that b is a turning point is $\frac{2}{3}$. It follows that the expected value of C , denoted by both $E(C)$ and μ_C , is

$$E(C) = \mu_C = (1)\left(\frac{2}{3}\right) + (0)\left(\frac{1}{3}\right) = \frac{2}{3}.$$

Here, E is the expected value operator.

$$E(C^2) = \frac{2}{3}$$

Define the function C^2 by $C^2(a, b, c) := C(a, b, c) \cdot C(a, b, c)$. Since $1^2 = 1$ and $0^2 = 0$, it follows that

$$C^2(a, b, c) = \begin{cases} 1 & \text{if } b \text{ is a turning point} \\ 0 & \text{otherwise} \end{cases}.$$

Therefore, $E(C^2)$ also equals $\frac{2}{3}$.

$$\text{var}(C) = \frac{2}{9}$$

Let $\text{var}(C)$ denote the variance of C . Using the definition of variance, and the linearity of the expected value operator, one computes

$$\begin{aligned} \text{var}(C) &:= E((C - \mu_C)^2) = E(C^2 - 2\mu_C C + \mu_C^2) \\ &= E(C^2) - 2\mu_C E(C) + E(\mu_C^2) = E(C^2) - 2(\mu_C)^2 + (\mu_C)^2 \\ &= E(C^2) - (\mu_C)^2 = \frac{2}{3} - \left(\frac{2}{3}\right)^2 = \frac{2}{9}. \end{aligned}$$

*count the number
of turning points
in a list,
 C_i*

Consider now a list $\mathbf{y} := (y_1, \dots, y_n)$ of length n . It is desired to count the number of turning points in this list. Since knowledge of both neighbors is required to classify a turning point, the first and last entries in a list cannot be turning points; so the maximum possible number of turning points present is $n - 2$.

Using the list \mathbf{y} , define C_i , for $i = 2, \dots, n - 1$, by

$$C_i(y_{i-1}, y_i, y_{i+1}) := \begin{cases} 1 & \text{if } y_i \text{ is a turning point} \\ 0 & \text{otherwise} \end{cases}.$$

*the C_i
do NOT form
a random sample*

Each random variable C_i is distributed identically to the random variable C . However, it is important to note that this collection of random variables $\{C_i\}_{i=2}^{n-1}$ is *not* a random sample corresponding to C , because C_i and C_j are *not* independent for $0 < |j - i| \leq 2$; that is, when the 3-tuples acted on by C_i and C_j overlap. This issue is addressed later on in this section.

T
gives the
total number
of turning points
in a list

Define a random variable T by

$$T = \sum_{i=2}^{n-1} C_i .$$

Then, T gives the total number of turning points in the list.

$$E(T) = \mu_T \\ = \frac{2}{3}(n-2)$$

Via linearity of the expected value operator,

$$E(T) = \sum_{i=2}^{n-1} E(C_i) = \frac{2}{3}(n-2) := \mu_T .$$

computing $\text{var}(T)$ Next, $\text{var}(T)$, the variance of the random variable T , is computed.

Since

$$\text{var}(T) = E(T^2) - (\mu_T)^2 ,$$

one first computes $E(T^2)$:

$$E(T^2) = E \left(\left(\sum_{i=2}^{n-1} C_i \right)^2 \right) \\ = E((C_2 + \cdots + C_{n-1})^2) .$$

counting terms
of the form $C_i C_j$

There are $(n-2)(n-2) = n^2 - 4n + 4$ terms in the product $(C_2 + \cdots + C_{n-1})^2$. It is necessary to count the number of terms of the form $C_i C_j$ for $j = i$, $|j - i| = 1$, $|j - i| = 2$, and $|j - i| > 2$; that is, when the indices on C are the same, or differ by exactly 1, exactly 2, or more than 2. This 'counting' is easily accomplished by performing the multiplication as a matrix product, and analyzing the result:

$$\begin{bmatrix} C_2 \\ C_3 \\ \vdots \\ C_{n-1} \end{bmatrix} [C_2 \ C_3 \ \cdots \ C_{n-1}] = \begin{bmatrix} C_2^2 & C_2 C_3 & C_2 C_4 & \cdots & C_2 C_{n-1} \\ C_3 C_2 & C_3^2 & C_3 C_4 & \cdots & C_3 C_{n-1} \\ C_4 C_2 & C_4 C_3 & C_4^2 & \cdots & C_4 C_{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ C_{n-1} C_2 & C_{n-1} C_3 & C_{n-1} C_4 & \cdots & C_{n-1}^2 \end{bmatrix} .$$

$|j - i| = 1$

The main diagonal has $n-2$ entries, each of the form C_i^2 . Thus, there are $n-2$ terms of the form C_i^2 .

There are $(n-2) - 1 = n-3$ entries on the first diagonal above and below the main diagonal, and these are the terms for which $|j - i| = 1$. Thus, there are $2(n-3)$ terms of the form $C_i C_{i+1}$.

There are $(n-2) - 2 = n-4$ entries on the second diagonal above and below the main diagonal, and these are the terms for which $|j - i| = 2$. Thus, there are $2(n-4)$ terms of the form $C_i C_{i+2}$.

The remaining terms are those for which $|j - i| > 2$; thus, there are

$$\begin{aligned}(n^2 - 4n + 4) - (n - 2) - 2(n - 3) - 2(n - 4) &= n^2 - 9n + 20 \\ &= (n - 4)(n - 5)\end{aligned}$$

terms of the form $C_i C_j$ for $|j - i| > 2$.

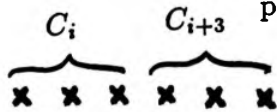
With a slight abuse of summation notation, the findings thus far are summarized as:

$$\begin{aligned}E(T^2) &= E\left(\left(\sum_{i=2}^{n-1} C_i\right)^2\right) \\ &= E\left(\sum_{n-2} C_i^2 + \sum_{2(n-3)} C_i C_{i+1} + \sum_{2(n-4)} C_i C_{i+2} + \sum_{\substack{(n-4)(n-5) \\ |j-i|>2}} C_i C_j\right) \quad (*)\end{aligned}$$

In each sum, the index denotes the number of terms, and the argument depicts the form of the terms being added. The expectations of each term in (*) must be considered separately.

investigating
 $E(C_i^2)$ and
 $E(C_i C_j), |j - i| > 2$

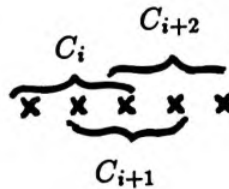
It has already been observed that $E(C_i^2) = \frac{2}{3}$, since $C_i^2 = C_i$. When $|j - i| > 2$, the random variables C_i and C_j have non-overlapping domains. Under the assumption of randomly generated data, the occurrence or non-occurrence of a turning point for C_i in no way influences the existence of a turning point for C_j in this case; i.e., C_i and C_j are independent. Thus,



$$E(C_i C_j) = E(C_i)E(C_j) = \frac{2}{3} \cdot \frac{2}{3} = \frac{4}{9}, \quad \text{for } |j - i| > 2.$$

investigating
 $E(C_i C_{i+1})$

However, for $j \leq 2$, the random variables C_i and C_{i+j} have overlapping domains, and $E(C_i C_{i+j}) \neq E(C_i)E(C_{i+j})$; i.e., C_i and C_{i+j} are *not* independent for $j \leq 2$.

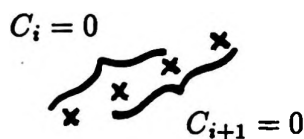


The proof of this statement follows. To evaluate $E(C_i C_{i+1})$ requires the investigation of existence of turning points in 4 consecutive slots. For convenience of notation, let four distinct real numbers be labeled in order of increasing magnitude as a , b , c and d . There are $4 \cdot 3 \cdot 2 \cdot 1 = 24$ ways that these four numbers can be arranged in four slots, as shown below:

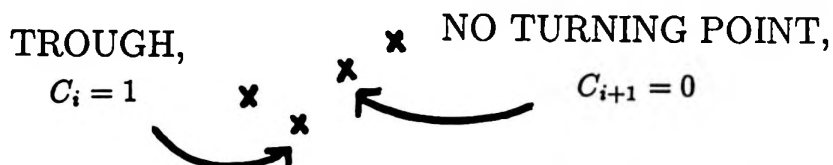
$abcd$	$bacd$	$cabd$	$dabc$
$abdc$	$badc$	$cadb$	$dacb$
$acbd$	$bcad$	$cbad$	$dbac$
$acdb$	$bcda$	$cbda$	$dbca$
$adbc$	$bdac$	$cdab$	$dcab$
$adcb$	$bdca$	$cdba$	$dcba$

investigating
particular
arrangements

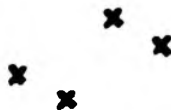
For the arrangement $abcd$, $C_i = 0$ and $C_{i+1} = 0$. Thus, $C_i C_{i+1} = 0$.



For the arrangement $bacd$, $C_i = 1$ (there is a trough). However, $C_{i+1} = 0$ (no turning point). Again, $C_i C_{i+1} = 0$.



For the arrangement $badc$, $C_i = 1$ (there is a trough), and $C_{i+1} = 1$ (there is a peak). Thus, $C_i C_{i+1} = (1)(1) = 1$.



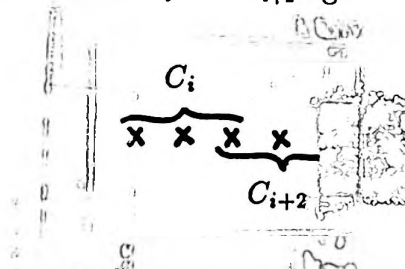
Indeed, for the product random variable $C_i C_{i+1}$ to be nonzero, the arrangement of a , b , c and d must display both a trough and a peak. This occurs in 10 of the 24 possible arrangements, and hence

$$E(C_i C_{i+1}) = \frac{10}{24} = \frac{5}{12}.$$

Note that $\frac{5}{12} \neq \frac{2}{3} \cdot \frac{2}{3}$, confirming that C_i and C_{i+1} are not independent.

investigating
 $E(C_i C_{i+2})$

The random variables C_i and C_{i+2} again have overlapping domains.



To evaluate $E(C_i C_{i+2})$ requires the investigation of turning points in 5 consecutive slots. By methods similar to those just discussed, it can be shown that

$$E(C_i C_{i+2}) = \frac{54}{120} = \frac{9}{20}.$$

combining
results

Substitution of the computed expectations into (*) gives

$$\begin{aligned} E(T^2) &= \frac{2}{3}(n-2) + \frac{5}{12} \cdot 2(n-3) + \frac{9}{20} \cdot 2(n-4) + \frac{4}{9}(n-4)(n-5) \\ &= \dots = \frac{40n^2 - 144n + 131}{90}. \end{aligned}$$

var(T)

Thus,

$$\begin{aligned} \text{var}(T) &= E(T^2) - (\mu_T)^2 \\ &= \frac{40n^2 - 144n + 131}{90} - \left(\frac{2}{3}(n-2)\right)^2 \\ &= \dots = \frac{16n - 29}{90}. \end{aligned}$$

With both the mean and variance of the random variable T now known, Chebyshev's Inequality (stated next) can be used to compare the *actual* number of turning points from a given data set with the number that is expected under the hypothesis of random behavior.

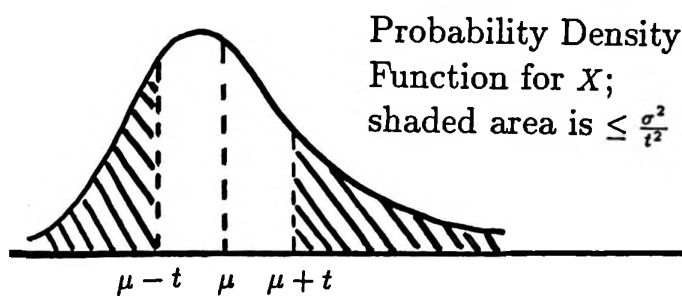
CHEBYSHEV's INEQUALITY [Dghty, 121] If the random variable X has mean μ and variance σ^2 , then, for any $t > 0$,

$$P(|X - \mu| \geq t) \leq \frac{\sigma^2}{t^2}.$$

Equivalently,

$$P(|X - \mu| < t) \geq 1 - \frac{\sigma^2}{t^2}.$$

This theorem states that for *any* random variable X with mean μ and variance σ^2 , the probability that X takes on a value which is at least distance t from the mean, is at most $\frac{\sigma^2}{t^2}$.



Observe that Chebyshev's Inequality is a 'distribution-free' result; that is, it is independent of the form of the probability density function for X . It is interesting to note that no 'tighter' bound on $P(|X - \mu| \geq t)$ is possible, without additional information about the actual distribution of X . That is, there exists a random variable for which equality is obtained in Chebyshev's Inequality: $P(|X - \mu| \geq t) = \frac{\sigma^2}{t^2}$ (see, e.g., [Dghty, 123–124]).

*using
Chebyshev's
Inequality
to test for
random behavior*

Here is how Chebyshev's Inequality and the turning point test are used to investigate the hypothesis that a given data set is random.

Suppose that a finite list of data values is given, where it is assumed that the entries in the list are allowed to come from some interval of real numbers. As cautioned earlier, if there are a large number of identical adjacent values in the data set, then the hypothesis that the values come from some *interval* of real numbers is probably unwarranted, and the turning point test as developed here does not apply.

*adjusting
the list
for occasional
identical
adjacent values*

For any *occasional* adjacent data values that are identical, delete the repeated value, and decrease n (the length of the list) by 1. For example, the data list

$$(1, 3, 5, \widehat{2, 2}, 7, 6, 4, 3, 9, 0, 1, 5, 8)$$

of length 14 would be transformed to the list

$$(1, 3, 5, 2, 7, 6, 4, 3, 9, 0, 1, 5, 8)$$

of length 13, before applying the turning point test.

Let N denote the length of the (possibly adjusted) data set.

*T_{act} ,
the actual number
of turning points*

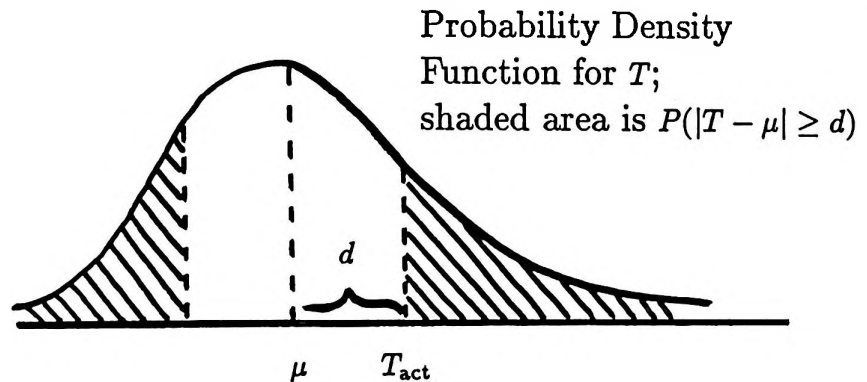
Let T_{act} denote the actual number of turning points in the (adjusted) list. Under the hypothesis of random behavior, the expected value and variance of the random variable T that counts the number of turning points in the list are given by

$$E(T) = \frac{2}{3}(N - 2) := \mu, \quad \text{and} \\ \text{var}(T) = \frac{16N - 29}{90} := \sigma^2.$$

$$d := |T_{\text{act}} - \mu|$$

Let $d := |T_{\text{act}} - \mu|$ denote the distance between the actual and expected number of turning points. By Chebyshev's Inequality, the probability that the distance of d or greater between μ and T would be observed, should the data be truly random, is

$$P(|T - \mu| \geq d) \leq \frac{\sigma^2}{d^2}.$$

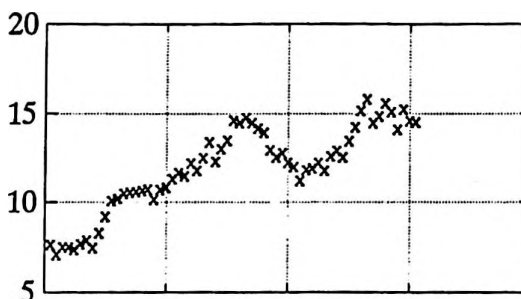


If $\frac{\sigma^2}{\sigma^2}$ is close to 0, then it is unlikely that T_{act} turning points would be observed if the data were truly random. In this case, the hypothesis that the data is random would be rejected, and the search for deterministic components could begin.

If $\frac{\sigma^2}{\sigma^2}$ is close to 1, then there is no reason to reject the hypothesis of random behavior. In this case, it may be fruitless to search for deterministic components.

EXAMPLE 1
*applying the
 turning point test*

The data graphed below give the biweekly stock price of a mutual fund over a two-year time period.



There are no identical adjacent values. The total number of data points is $N = 61$.

The expected number of turning points, under the hypothesis of random behavior, is

$$\mu = \frac{2}{3}(N - 2) \approx 39.33 .$$

The actual number of turning points is

$$T_{\text{act}} = 28 ,$$

so that the distance between the actual and expected values is

$$d := |39.33 - 28| = 11.33 .$$

The variance of T is

$$\sigma^2 = \frac{16N - 29}{90} = \frac{16(61) - 29}{90} \approx 10.52 .$$

Chebyshev's Inequality yields:

$$P(|T - \mu| \geq 11.33) \leq \frac{10.52}{(11.33)^2} \approx 0.08 .$$

Thus, it is quite unlikely that only 28 turning points would be observed, if the data were truly random. The hypothesis of random behavior is therefore rejected, and a search for deterministic components can begin.

'local'
random behavior

Some data sets, as in the next example, are 'locally' random, and yet exhibit some deterministic behavior from a more 'global' point of view. In such instances, short-term data prediction may be unwarranted, whereas longer-term prediction may be possible. For sufficiently large data sets, the turning point test can be used to help determine the 'breadth of local random behavior'. This idea is explored in the next example.

EXAMPLE 2

The data list graphed below was generated within MATLAB by first producing some pure data, via the MATLAB commands

```
i = [1:100];
```

```
y = i/6;
```

and then introducing noise by use of the MATLAB command `rand(A)`.

**MATLAB
COMMAND**
`rand(A)`

The MATLAB command `rand(A)` produces a matrix the same size as `A`, with random entries. By default, the random numbers are uniformly distributed in the interval $(0, 1)$. Then,

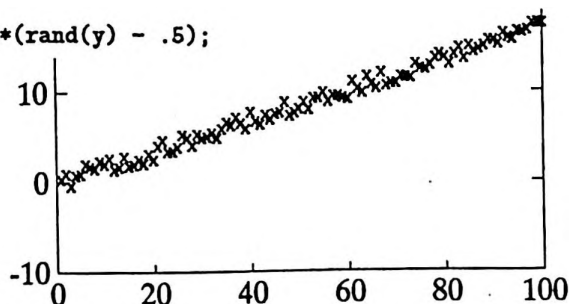
```
2*(rand(A) - .5)
```

gives numbers uniformly distributed in $(-1, 1)$.

The command `rand('normal')` can be used to switch to a normal distribution with mean 0 and variance 1. The command `rand('uniform')` then switches back to the uniform distribution.

The list `noisy` graphed below was generated by the MATLAB command

```
noisy = y + 2*(rand(y) - .5);
```



applying the
turning point test
to `noisy`

The list `noisy` has 63 turning points, so $T_{\text{act}} = 63$.

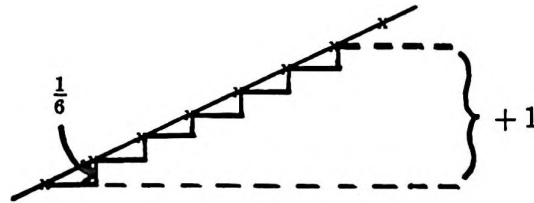
The list `noisy` has length 100, so $N = 100$, and thus $\mu = \frac{2}{3}(100) \approx 66.67$ and $\sigma^2 = \frac{16(100)-29}{90} \approx 17.46$.

Then, $d = |66.67 - 63| = 3.67$.

Chebyshev's Inequality yields:

$$P(|T - \mu| \geq 3.67) \leq \frac{17.46}{(3.67)^2} \approx 1.3 .$$

Even though the data clearly illustrate a linear 'trend', there is no reason, based on this test, to reject the hypothesis of random behavior. Here is what the turning point test is revealing in this situation: in moving through the list entry-by-entry, the numbers rise and fall in such a way that they could certainly have been produced by an entirely random process. Indeed, since the slope of the 'pure' line is $\frac{1}{6}$, and the noise is ± 1 , it could take more than 6 data entries before any increase due to the linear trend is observed. Prediction of one data value into the future is unwarranted.



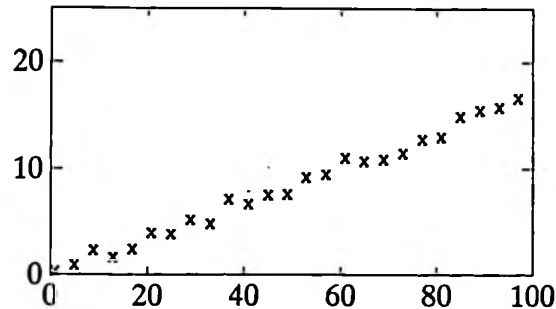
However, if one were to move through the list by taking, say, every *fourth* entry, then the 'local' random behavior may be overshadowed by the 'global' linear trend. This idea is investigated in a second, slightly different, application of the turning point test.

EXAMPLE 2,
continued

Generate a new list from `noisy`, by taking every *fourth* piece of data. Call the new list `noisy4`. This is accomplished via the MATLAB command

```
noisy4 = noisy(1:4:100);
```

Whereas the time list corresponding to `noisy` has spacing $T = 1$, the time list corresponding to `noisy4` has spacing $T = 4$. The new list `noisy4` is graphed below, and has length $N = 25$.



The list `noisy4` has 10 turning points, so $T_{\text{act}} = 10$.

The list `noisy4` has length 25, so $N = 25$, and thus $\mu = \frac{2}{3}(25) \approx 16.67$ and $\sigma^2 = \frac{16(25)-29}{90} \approx 4.12$.

Then, $d = |16.67 - 10| = 6.67$.

Chebyshev's Inequality yields:

$$P(|T - \mu| \geq 6.67) \leq \frac{4.12}{(6.67)^2} \approx .09.$$

The hypothesis of random behavior is rejected for `noisy4`. Thus, predicting future values of the list `noisy4`, based on identified components, may be warranted. In other words, prediction of 4 or more days into the future for the original list `noisy` may be warranted.

In this example, the data clearly exhibit a linear trend. A method of 'fitting' data with a function of a specific form is discussed in Section 2.2.

In general, it is possible, with sufficient data, to continually produce sublists, \mathbf{xk} , of a list \mathbf{x} , by taking every k^{th} entry from \mathbf{x} . If the turning point test, when applied to \mathbf{xk} , concludes that the hypothesis of random behavior is rejected, then prediction of k or more units into the future for the original list \mathbf{x} may be warranted.

**MATLAB
FUNCTION**
*Turning Point
Test*

The following MATLAB function is used by typing

`y = tpctest(x)`

where:

`x` is the INPUT row or column vector;

`y` is the program OUTPUT.

The output matrix `y` consists of rows, where each row is of the form: [length nofdup mu k TP P]

The variable `length` is the length of the list produced by taking every k^{th} entry from `x`, and adjusting the resulting sublist to account for adjacent identical values. The number of duplicates found in the list is recorded in `nofdup`.

The variable `mu` is the expected number of turning points, if the behavior is truly random.

The variable `TP` is the actual number of turning points.

The variable `P` is $\frac{\sigma^2}{d^2}$, from Chebyshev's Inequality,

$$P(|T - \mu| \geq d) \leq \frac{\sigma^2}{d^2}.$$

The test is repeatedly applied to sublists, until either the list is depleted, or until $P < 0.1$.

```
function T = tpctest(x)
P = 1;
T = [ ];
k = 1;
cx = x; % cx is a copy of x
M = length(x);
while ((P >= 0.1) & (M > 3))
Ndup = 0;
j = 1;
while j < M,
    if cx(j) == cx(j+1)
        cx(j) = [ ];
        j = j-1;
        M = M-1;
        Ndup = Ndup+1;
    end
    j=j+1;
end
mu = (2/3)*M;
sigma = (16*M - 29)/(90);
TP = 0;
for i = 2:M-1,
    if [(cx(i-1)>cx(i))&(cx(i+1)>cx(i))] | [(cx(i-1)<cx(i))&(cx(i+1)<cx(i))],
        TP = TP + 1;
    end
end
d = abs(TP - mu);
P = sigma/(d.^2);
T = [T; M Ndup mu k TP P];
k = k+1;
cx = x(1:k:length(x));
M = length(cx);
end
```

The following diary of an actual MATLAB session shows the application of this Turning Point Test to the list `noisy` in Example 2.

```
i = [1:100];
y = i/6;
noisy = y + 2*(rand(y) - .5);
z = tpctest(noisy)
```

z =

100.0000	0	66.6667	1.0000	63.0000	1.2983
50.0000	0	33.3333	2.0000	25.0000	0.1234
34.0000	0	22.6667	3.0000	16.0000	0.1288
25.0000	0	16.6667	4.0000	10.0000	0.0928

LENGTH	nofdup	mu	k	TP	P
--------	--------	----	---	----	---

Economics Application: Taking Advantage of Turning Points

Introduction

If the hypothesis of random behavior is *not* rejected for given data, then it may be fruitless to seek deterministic components. However, the economics application discussed in this section shows how one can, even in this situation, often take advantage of the turning points (rises and falls) in stock market data. The strategy is due to Eliason [Eli]. First, the underlying mathematical theory is presented. Then, an example illustrating the application of this theory to stock market trading is given.

preliminary notation

Let $x(0)$ be a given finite list of real numbers. Let $x(n)$ denote the list present at the completion of step n ($n \geq 1$) in the Martingale Algorithm (below), where $x(0)$ is the initial input. Let $A: \{1, 2, 3, \dots\} \rightarrow \{W, L\}$ be a given function; for $n \geq 1$, either $A(n) = W$ or $A(n) = L$.

Let P and B be functions,

$$P: \{0, 1, 2, 3, \dots\} \rightarrow \mathbf{R}, \quad B: \{0, 1, 2, 3, \dots\} \rightarrow \mathbf{R};$$

the values assigned to $P(n)$ and $B(n)$ (for $n \geq 0$) are determined by the Martingale Algorithm.

MARTINGALE ALGORITHM

Initialization, $n = 0$

THE MARTINGALE ALGORITHM:

INITIALIZATION ($n = 0$): Define $P(0) = 0$. If $x(0)$ has two or more entries, then let $B(0)$ be the sum of the first and last entries in $x(0)$. If $x(0)$ has only one entry, x , then let $B(0) = x$.

STEP n ; $n \geq 1$

STEP n , for $n \geq 1$:

$A(n) = W$; find $P(n)$ and $x(n)$

If $A(n) = W$, then let $P(n) = P(n-1) + B(n-1)$. In this case, adjust the list $x(n-1)$ to get the list $x(n)$, as follows:

- If $x(n-1)$ has more than two entries, then delete the first and last entries to obtain $x(n)$.
- If $x(n-1)$ has only one or two entries, then delete these entries, and STOP the algorithm.

$A(n) = L$; find $P(n)$ and $x(n)$

If $A(n) = L$, then let $P(n) = P(n-1) - B(n-1)$. In this case, adjust the list $x(n-1)$ to get the list $x(n)$, by appending $B(n-1)$ to the end of $x(n-1)$.

find $B(n)$

FIND $B(n)$: If $x(n)$ has two or more entries, then let $B(n)$ be the sum of the first and last entries in $x(n)$. If $x(n)$ has only one entry, x , then let $B(n) = x$. Go to the next value of n .

the letters
 A , P , and B ,
 W and L

The variable names used in the Martingale Algorithm are suggestive of common roles that these variables play in applications of the algorithm. The function A is the 'Action' function; W denotes a 'Win' and L denotes a 'Loss'. The function P is the 'Profit' function, and B is the 'Bet' function. When a WIN occurs, the profit is increased by the previous bet; when a LOSS occurs, the profit is decreased by the previous bet.

EXAMPLE 1
 applying the
 Martingale
 Algorithm

Let $\mathbf{x} = (1, 2, 3, 5)$, and let $A(n) = (L, W, W, L, W, \dots)$. The table below summarizes the algorithm:

n	$A(n)$	$P(n)$	$\mathbf{x}(n)$	$B(n)$
0	—	0	1 2 3 5	6
1	L	$0 - 6 = -6$	1 2 3 5 6	7
2	W	$-6 + 7 = 1$	2 3 5	7
3	W	$1 + 7 = 8$	3	3
4	L	$8 - 3 = 5$	3 3	6
5	W	$5 + 6 = 11$	STOP	—

Observe that the algorithm STOPPED at $N = 5$, and $P(5) = 11 = 1 + 2 + 3 + 5$; that is, when the algorithm stopped, the value of P is the sum of the digits in the initial list.

EXAMPLE 2
 applying the
 Martingale
 Algorithm
 with a different
 action list

Suppose that a different action function is used:

$$A(n) = (L, L, L, W, L, W, W, \dots).$$

The algorithm is summarized in the table below:

n	$A(n)$	$P(n)$	$\mathbf{x}(n)$	$B(n)$
0	—	0	1 2 3 5	6
1	L	$0 - 6 = -6$	1 2 3 5 6	7
2	L	$-6 - 7 = -13$	1 2 3 5 6 7	8
3	L	$-13 - 8 = -21$	1 2 3 5 6 7 8	9
4	W	$-21 + 9 = -12$	2 3 5 6 7	9
5	L	$-12 - 9 = -21$	2 3 5 6 7 9	11
6	W	$-21 + 11 = -10$	3 5 6 7	10
7	W	$-10 + 10 = 0$	5 6	11
8	L	$0 - 11 = -11$	5 6 11	16
9	W	$-11 + 16 = 5$	6	6
10	W	$5 + 6 = 11$	STOP	—

This time, the algorithm stopped at $N = 10$, but again $P(N) = 11$. The next theorem shows that this behavior is no coincidence:

THEOREM <i>the series number associated with \mathbf{x}</i>	Let $\mathbf{x}(0) = (x_1, \dots, x_m)$ be a finite list of real numbers, with $m \geq 1$. Let $M := x_1 + \dots + x_m$ be the sum of the entries in $\mathbf{x}(0)$. If a STOP occurs in the Martingale Algorithm at step N , where $\mathbf{x}(0)$ has been used as the initial input, then $P(N) = M$. The number M is called the <i>series number associated with the list $\mathbf{x}(0)$</i> .
--	--

PROOF

definition of $T(n)$, the induction statement

The proof is by induction.

For $n \geq 1$, let $T(n)$ be the statement,

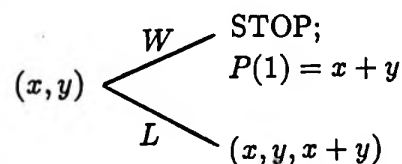
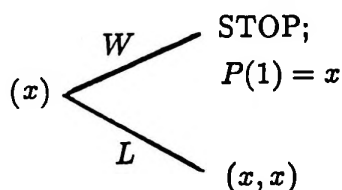
'If the algorithm stops at step n , then $P(n) = M$ '.

a series of W 's is the quickest way to STOP

A series of W 's in the corresponding action list is the quickest way to stop the algorithm. First, action lists containing only W 's are considered. Then, the induction step is applied to action lists that contain at least one L .

$\mathbf{x}(0)$ has one or two entries; $T(1)$ is true

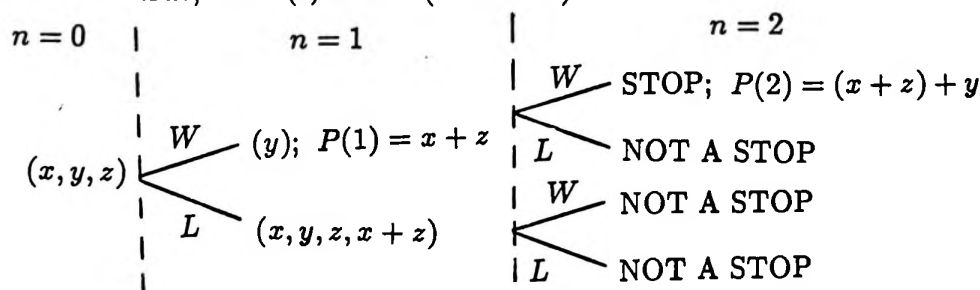
If $\mathbf{x}(0)$ has only one entry, $\mathbf{x}(0) = (x)$, then $T(1)$ is true (see below).



If $\mathbf{x}(0)$ has two entries, $\mathbf{x}(0) = (x, y)$, then $T(1)$ is true (see above).

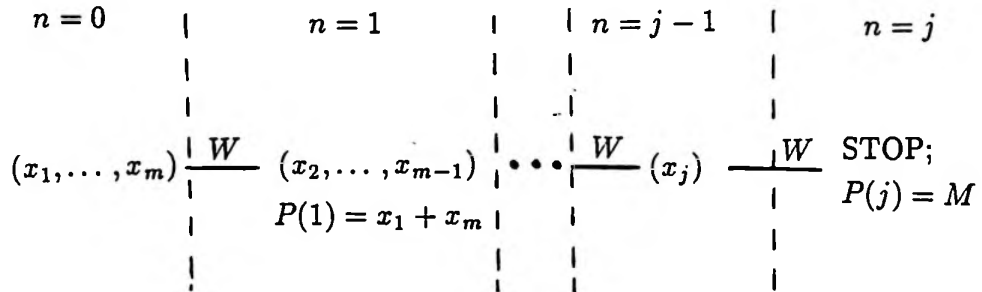
$\mathbf{x}(0)$ has three entries

If $\mathbf{x}(0)$ has three entries, $\mathbf{x}(0) = (x, y, z)$, then it takes at least two steps to STOP the algorithm. In this case, $T(1)$ is vacuously true, and $T(2)$ is true (see below).



$x(0)$ has
 m entries,
 m is odd

Now suppose $x(0)$ has m entries, $x(0) = (x_1, \dots, x_m)$, where $m \geq 4$. If m is odd, then let $m = 2j - 1$ for $j \geq 3$. A series of W 's produces a STOP in j steps, and $T(j)$ is true. For $1 \leq k < j$, $T(k)$ is vacuously true. Only the W 's are shown in the flow chart below.



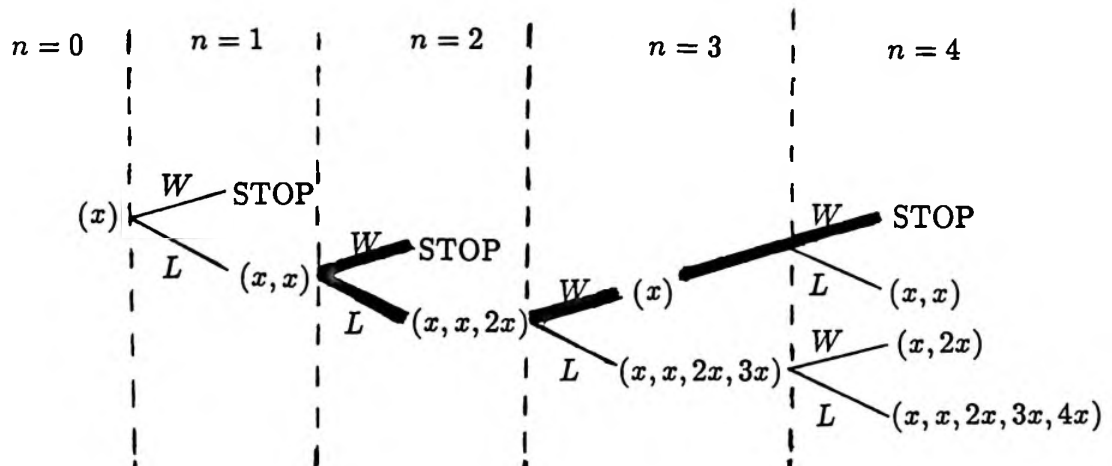
$x(0)$ has
 m entries,
 m is even

If m is even, then let $m = 2j$ for $j \geq 2$. A series of W 's produces a STOP in j steps, and $T(j)$ is true. For $1 \leq k < j$, $T(k)$ is vacuously true.

the action list
has at least one
 L

Suppose now that the action list has at least one L . Suppose that $T(k)$ is true for all $k = 1, \dots, N - 1$, and consider the statement $T(N)$.

To motivate what follows, consider a typical flow chart that summarizes all possible actions on a list:



important
observations

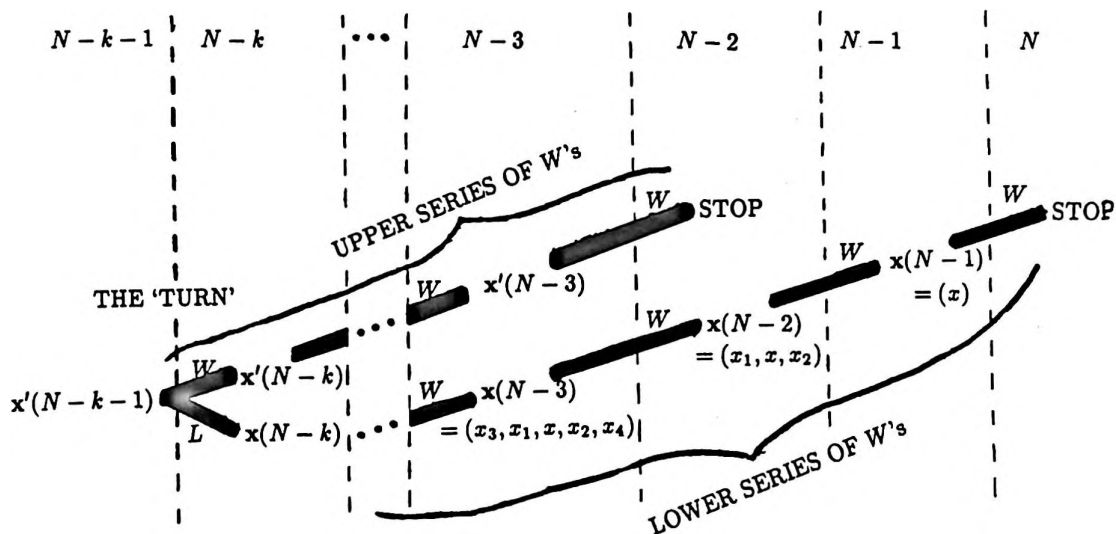
The following observations are important:

- Every STOP at step N comes from a W in step N . Also, in these cases, $x(N-1)$ has either 1 or 2 entries.
- Tracing back from a W , there is a *first* L , say in step $N-k$, for $1 \leq k \leq N-1$.
- Taking only W 's from step $N-k$ leads to a STOP that is earlier than step N . The inductive hypothesis will be applied to this earlier STOP, to show that $T(N)$ is true.

suppose a STOP
occurs at step N ,
 $x(N-1)$ has only
one entry

Now, the induction argument. Suppose that a STOP occurs in step N , and suppose that $x(N-1)$ has one entry.

The flow chart below is useful in summarizing the results:



notation:

$P(n)$ and $x(n)$,
 $P'(n)$ and $x'(n)$

For ease of notation in what follows, the profit functions and lists are denoted by $P(n)$ and $x(n)$, respectively, along the lower series of W 's, and are denoted by $P'(n)$ and $x'(n)$ around the TURN and along the upper series of W 's.

lower series
of W 's

- Let $\mathbf{x}(N-1) = (x)$. Since $P(N) = P(N-1) + x$, it follows that $P(N-1) = P(N) - x$.
- Then, $\mathbf{x}(N-2) = (x_1, x, x_2)$ for real numbers x_1 and x_2 . Since

$$P(N-1) = P(N-2) + (x_1 + x_2) ,$$

it follows that

$$\begin{aligned} P(N-2) &= P(N-1) - (x_1 + x_2) \\ &= P(N) - (x + x_1 + x_2) . \end{aligned}$$

- Then, $\mathbf{x}(N-3) = (x_3, x_1, x, x_2, x_4)$ for real numbers x_3 and x_4 . Since

$$P(N-2) = P(N-3) + (x_3 + x_4) ,$$

it follows that

$$\begin{aligned} P(N-3) &= P(N-2) - (x_3 + x_4) \\ &= P(N) - (x + x_1 + x_2 + x_3 + x_4) . \end{aligned}$$

- Continuing in this fashion, one has, in step $N-k$, for $2 \leq k \leq N-1$,

$$\begin{aligned} \mathbf{x}(N-k) &= (x_{2k-3}, x_{2k-5}, \dots, x_1, x, x_2, \dots, x_{2k-4}, x_{2k-2}) , \text{ and} \\ P(N-k) &= P(N) - (x + x_1 + x_2 + \dots + x_{2k-3} + x_{2k-2}) . \end{aligned}$$

the TURN

Taking the first L that occurs in step $N-k$, one has:

$$\mathbf{x}'(N-k-1) = (x_{2k-3}, \dots, x_1, x, x_2, \dots, x_{2k-4}) ,$$

where x_{2k-2} must equal $x_{2k-3} + x_{2k-4}$, since the first and last entries of $\mathbf{x}'(N-k-1)$ are summed and appended to $\mathbf{x}'(N-k-1)$ to produce $\mathbf{x}(N-k)$.

Since

$$P(N-k) = P'(N-k-1) - (x_{2k-3} + x_{2k-4}) ,$$

it follows that

$$\begin{aligned} P'(N-k-1) &= P(N-k) + (x_{2k-3} + x_{2k-4}) \\ &= P(N) - (x + x_1 + x_2 + \dots + x_{2k-3} + \overbrace{x_{2k-2}}^{=x_{2k-3}+x_{2k-4}}) + (x_{2k-3} + x_{2k-4}) \\ &= P(N) - (x + x_1 + x_2 + \dots + x_{2k-3}) . \end{aligned}$$

*upper series
of W 's*

Note that $x'(N-k-1)$ has $2k-2 = 2(k-1)$ entries. Each successive W will delete two of these entries, so a STOP will occur in $k-1$ steps, that is, in step $(N-k-1) + (k-1) = N-2$.

Since $P'(N-k) = P'(N-k-1) + (x_{2k-3} + x_{2k-4})$, it follows that

$$P'(N-k) = P(N) - (x + x_1 + x_2 + \cdots + x_{2k-6} + x_{2k-5}) .$$

Continuing in this fashion, the remaining $k-2$ W 's will produce a STOP in step $N-2$, with

$$P'(N-2) = P(N) .$$

By the inductive hypothesis, $P'(N-2) = M$, thus proving that $P(N) = M$.

*the case where
 $x(N-1)$ has
two entries*

In the case where $x(N-1)$ has two entries, the preceding argument goes through, mutatis mutandis, to show that $P'(N-1) = P(N)$; and, by the inductive hypothesis, $P'(N-1)$ equals M , thus proving that $P(N) = M$.

*combining
results*

Combining results, it has been proven that if a STOP occurs in step N , then $P(N) = M$, thereby showing that $T(N)$ is true, and completing the proof. ■

Next, the application of this theory to stock market data is given.

**ECONOMICS
ALGORITHM**
*choose x ,
with
series number
 $M > 0$*

Let $x(0) = (x_1, \dots, x_m)$ be a list of positive integers to be input into the Martingale Algorithm, with $m > 1$. Then, the series number $M = x_1 + \cdots + x_m$ is positive.

Let B and P be the functions described in the Martingale Algorithm. Since each list $x(n)$ in the Martingale Algorithm will have positive entries (since $x(0)$ does), it follows that $B(n)$ will be positive, for all n .

Note that $B(0) = x_1 + x_m$, and $P(0) = 0$.

It may be helpful to study Examples 3 and 4 as you read through this algorithm.

*$y(t)$ gives the
stock price
at time t*

Let t_0 be some starting time, and let $y(t_0)$ be the price of a selected stock at time t_0 . In general, $y(t)$ will denote the price of the stock for $t > t_0$. The units of $y(t)$ will typically be dollars.

shares are
purchased in
multiples of a
positive integer S

Let S (for 'Shares') be a positive integer. Shares of stock will be purchased in multiples of S . Since broker's fees are usually much smaller when shares are purchased in multiples of 100, S is typically a multiple of 100.

initial purchase:
 $B(0) \cdot S$

To begin the algorithm, the trader purchases $B(0) \cdot S$ shares of stock, at price $y(t_0)$. The algorithm discussed below will determine future buys and sells of stock.

the
Point Spread,
 PS ,
determines when
action is taken

Let PS (for 'Point Spread') be a fixed positive real number, that will determine when Action (buying or selling of stocks) is to take place, as described next. The units of PS are the same as the units of $y(t)$.

step 1

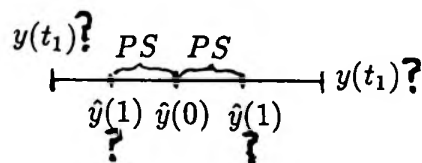
Define $\hat{y}(0) := y(t_0)$. Let $t_1 > t_0$ be a time for which $|y(t_1) - \hat{y}(0)| \geq PS$; that is, t_1 is a time at which the price of stock has deviated from the beginning price by PS or more.

Ideally, (to speed the algorithm to its completion), t_1 will be the *first* time (after t_0) that the stock price changes by PS or more.

Define

$$\hat{y}(1) := \begin{cases} \hat{y}(0) + PS & \text{if } y(t_1) > \hat{y}(0) \\ \hat{y}(0) - PS & \text{if } y(t_1) < \hat{y}(0) \end{cases}$$

Then, $\hat{y}(1)$ gives the 'ideal' trading price at step 1; but a broker may not be able to buy/sell at this 'ideal' price $\hat{y}(1)$. The value $y(t_1)$ gives the *actual* stock price at the time of transaction.



step n , for
 $n > 1$

The procedure followed in step 1 is now repeated for steps 2, 3, 4, ...

Let $t_n > t_{n-1}$ be a time for which $|y(t_n) - \hat{y}(n-1)| \geq PS$. Define

$$\hat{y}(n) := \begin{cases} \hat{y}(n-1) + PS & \text{if } y(t_n) > \hat{y}(n-1) \\ \hat{y}(n-1) - PS & \text{if } y(t_n) < \hat{y}(n-1) \end{cases}$$

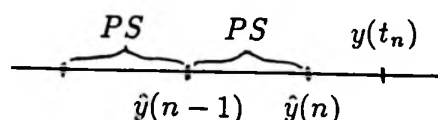
Then, $\hat{y}(n)$ gives the 'ideal' trading price at step n . The value $y(t_n)$ give the *actual* stock price at the time of transaction.

*trader's strategy
is assumed to be
'buying long'*

For the Economics Algorithm discussed here, it is assumed that the trader is *buying long*; that is, buying low, with the hopes of selling high. In this case, a rise in stock price is favorable, and is considered to be a WIN; a fall in stock price is unfavorable, and is considered to be a LOSS.

The changes in stock prices (of PS or more) will determine the ACTION function A , as follows: a W (for WIN) is recorded if the stock price increases, and a L (for LOSS) is recorded otherwise.

For example, suppose that $|y(t_n) - \hat{y}(n-1)| \geq PS$, and $y(t_n) > \hat{y}(n-1)$. Then, the price of stock has *risen* (from the previous 'ideal') by PS or more, and one sets $A(n) = W$.



If $|y(t_n) - \hat{y}(n-1)| \geq PS$, and $y(t_n) < \hat{y}(n-1)$, then the price of stock has *fallen* (from the previous 'ideal') by PS or more, and one sets $A(n) = L$.

*sgn(n) keeps
track of
wins and losses*

Define

$$\text{sgn}(n) := \begin{cases} 1 & \text{if } A(n) = W \\ -1 & \text{if } A(n) = L \end{cases}.$$

*adjust the
list $x(n-1)$*

Based on $A(n)$, the list $x(n-1)$ is adjusted, and the number $B(n)$ is computed, as per the Martingale Algorithm.

*$TS(n) := B(n) \cdot S$
gives the number
of shares the
trader must own
at the completion
of step n*

The number $B(n) \cdot S$ gives the number of shares of stock that the trader must own at the completion of step n . This total number of shares is denoted by $TS(n)$ in the following table.

$SH(n)$

In order to own $TS(n)$ shares, an appropriate number of shares is bought or sold (whichever is appropriate) at price $y(t_n)$. The number of shares that must be bought or sold at step n is denoted by $SH(n)$ in the following table. If $SH(n)$ is positive, then shares are purchased; if $SH(n)$ is negative, then shares are sold.

$ACT(n)$ The account value at the completion of step n is denoted by $ACT(n)$ in the following table. If the trader is in debt at the completion of step n , then $ACT(n)$ is negative. Otherwise, $ACT(n) \geq 0$.

Notice that if $SH(n)$ is positive, then $SH(n)$ shares are purchased at (positive) price $y(t_n)$, and $ACT(n) = ACT(n-1) - y(t_n)SH(n)$.

On the other hand, if $SH(n)$ is negative, then $-SH(n)$ is positive, and $-SH(n)$ shares are sold at (positive) price $y(t_n)$, and $ACT(n) = ACT(n-1) + y(t_n)(-SH(n))$.

In both cases,

$$ACT(n) = ACT(n-1) - y(t_n)SH(n) .$$

stopping the algorithm The algorithm STOPS when the list $x(n)$ is depleted. At this step, $B(n) = 0$, and all shares of stock currently owned are sold.

symbolic table The table below summarizes the algorithm symbolically. All the symbols used in this table have been discussed in the previous paragraphs.

n	$y(t_n)$	$\dot{y}(n)$	$A(n)$	$\text{sgn}(n)$	$P(n)$	$x(n)$	$B(n)$	$TS(n)$	$SH(n)$	$ACT(n)$
0	$y(t_0)$	$\dot{y}(0)$	-	-	0	x	$B(0)$	$B(0)S$	$B(0)S$	$-B(0)Sy(t_0)$
1	$y(t_1)$	$\dot{y}(1)$	$A(1)$	$\text{sgn}(1)$	$B(0)\text{sgn}(1)$	$x(1)$	$B(1)$	$B(1)S$	$(B(1) - B(0))S$	$ACT(0) - y(t_1)SH(1)$
2	$y(t_2)$	$\dot{y}(2)$	$A(2)$	$\text{sgn}(2)$	$P(1) + B(1)\text{sgn}(2)$	$x(2)$	$B(2)$	$B(2)S$	$(B(2) - B(1))S$	$ACT(1) - y(t_2)SH(2)$

EXAMPLE 3 The next table illustrates the algorithm in the case where $x(0) = (1, 2, 3, 4, 5, 6)$, the trader is buying long, the rise and fall of stock prices is such that $A(n) = (L, L, W, W, W, L, W, W)$, $PS = \$1$, $S = 100$, and transactions are made at the 'ideal' trading prices.

Observe that the series number for $x(0)$ is 21, and the account value at the completion of the algorithm is

$$\$2100 = (PS) \cdot (S) \cdot (\text{series number}) .$$

The theorem following Example 4 proves that this is no coincidence.

n	$y(t_n)$	$\hat{y}(n)$	$A(n)$	$\text{sgn}(n)$	$x(n)$	$B(n)$	$TS(n)$	$SH(n)$	$ACT(n)$
0	9	9	-	-	(1,2,3,4,5,6)	7	700	700	$-700(9) = -6300$
1	8	8	L	-1	(1,2,3,4,5,6,7)	8	800	100	$-6300 - 100(8) = -7100$
2	7	7	L	-1	(1,2,3,4,5,6,7,8)	9	900	100	$-7100 - 100(7) = -7800$
3	8	8	W	1	(2,3,4,5,6,7)	9	900	0	-7800
4	9	9	W	1	(3,4,5,6)	9	900	0	-7800
5	10	10	W	1	(4,5)	9	900	0	-7800
6	9	9	L	-1	(4,5,9)	13	1300	400	$-7800 - 400(9) = -11400$
7	10	10	W	1	(5)	5	500	-800	$-11400 + 800(10) = -3400$
8	11	11	W	1	STOP	0	0	-500	$-3400 + 500(11) = 2100$

EXAMPLE 4 The next two tables illustrate that when the actual prices $y(t_i)$ deviate from the 'ideal' prices $\hat{y}(i)$, the end profit may be either higher or lower than the 'ideal' profit.

The initial list $x(0) = (1, 2, 2, 1)$ is used, $PS = \$1$, and $S = 100$. Thus, the 'ideal' profit is \$600.

n	$y(t_n)$	$\hat{y}(n)$	$A(n)$	$\text{sgn}(n)$	$x(n)$	$B(n)$	$TS(n)$	$SH(n)$	$ACT(n)$
0	9	9	-	-	(1,2,2,1)	2	200	200	$-200(9) = -1800$
1	10.1	10	W	1	(2,2)	4	400	200	$-1800 - 200(10.1) = -3820$
2	11	11	W	1	STOP	0	0	-400	$-3820 + 400(11) = 580$

n	$y(t_n)$	$\hat{y}(n)$	$A(n)$	$\text{sgn}(n)$	$x(n)$	$B(n)$	$TS(n)$	$SH(n)$	$ACT(n)$
0	9	9	-	-	(1,2,2,1)	2	200	200	$-200(9) = -1800$
1	10	10	W	1	(2,2)	4	400	200	$-1800 - 200(10) = -3800$
2	11.1	11	W	1	STOP	0	0	-400	$-3800 + 400(11.1) = 640$

THEOREM

Let $x(0) = (x_1, \dots, x_m)$ be a list of positive integers to be input to the Martingale Algorithm, with $m > 1$, and with series number M . Let PS be a positive real number, and let S be a positive integer. Let all other notation be as described in the Economics and Martingale Algorithms. Suppose that all transactions are made at the ideal prices $\hat{y}(n)$. If a STOP occurs in the Economics Algorithm at step N , then

$$ACT(N) = (S) \cdot (PS) \cdot (M) .$$

PROOF

Suppose that a STOP occurs at step N . Then, by the Martingale Algorithm Theorem, $P(N) = M$. The following expression for $P(N)$ is obtained:

$$\begin{aligned}
 M = P(N) &= P(N-1) + B(N-1)\text{sgn}(N) \\
 &= P(N-2) + B(N-2)\text{sgn}(N-1) + B(N-1)\text{sgn}(N) \\
 &= \dots \\
 &= P(0) + B(0)\text{sgn}(1) + \dots + B(N-1)\text{sgn}(N) \\
 &= B(0)\text{sgn}(1) + \dots + B(N-1)\text{sgn}(N) .
 \end{aligned}$$

Observe also that, under the stated hypotheses,

$$\begin{aligned}
 y(t_i) - y(t_{i-1}) &= \text{sgn}(i)|y(t_i) - y(t_{i-1})| \\
 &= \text{sgn}(i)(PS) .
 \end{aligned}$$

Observe that

$$SH(i) = S(B(i) - B(i-1)) .$$

Now, compute $ACT(N)$:

$$\begin{aligned}
 ACT(N) &= ACT(N-1) - y(t_N) SH(N) \\
 &= ACT(N-2) - y(t_{N-1}) SH(N-1) - y(t_N) SH(N) \\
 &= \dots \\
 &= ACT(0) - [y(t_N) SH(N) + \dots + y(t_1) SH(1)] \\
 &= -B(0) S y(t_0) - [y(t_N) S(B(N) - B(N-1)) \\
 &\quad + \dots + y(t_2) S(B(2) - B(1)) + y(t_1) S(B(1) - B(0))] \\
 &= S[B(0)(y(t_1) - y(t_0)) + \dots + B(N-1)(y(t_N) - y(t_{N-1}))] \\
 &= S[B(0)\text{sgn}(1)(PS) + \dots + B(N-1)\text{sgn}(N)(PS)] \\
 &= S \cdot PS[B(0)\text{sgn}(1) + \dots + B(N-1)\text{sgn}(N)] \\
 &= (S)(PS)(M) . \blacksquare
 \end{aligned}$$

*practical
considerations*

The following observations are important:

- Typically, a trader will set two orders with a broker: one action, should a W occur, and another action, should a L occur. When the stock price changes by PS or greater, one order is filled, and the other order is cancelled.

- The algorithm need not ever STOP. For example, the action list (L, L, L, L, \dots) will cause the list $x(i)$ to continually grow in length.

A MATLAB program for computing the probability that the algorithm STOPS in less than or equal to N steps is included at the end of this section.

- Even if a STOP does occur, the broker's fees for transactions may 'overpower' the final profit.
- The account value $ACT(i)$ can get extremely negative, before reaching the final profit.
- Different values of PS can affect the algorithm dramatically. A proper determination of PS , based on historical data, is important: many values of PS should be 'tested', and an optimal value chosen.

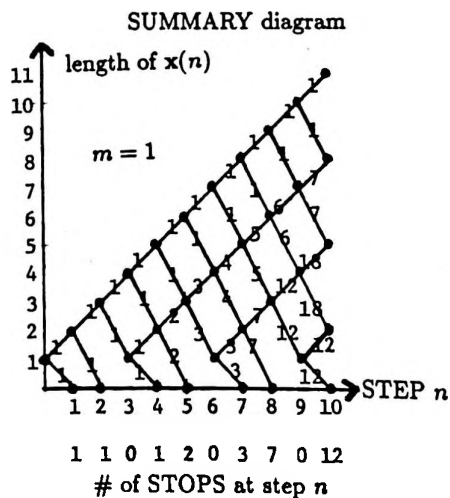
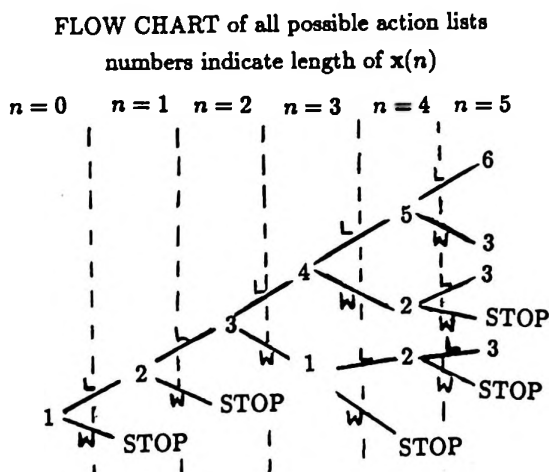
The reader interested in predictability of stock market prices is referred to [G&M] and [Rosen].

*probabilities
of STOPPING
the Martingale
Algorithm*

Let N and m be positive integers. Suppose that the Martingale Algorithm is started with a list of length m . Suppose that, at every step, there is equal probability that a W and L will occur in the action list; that is,

$$\text{probability}(W) = \text{probability}(L) = 0.5.$$

In order to determine the probability that the Martingale Algorithm STOPS in at most N steps, it is convenient to summarize the flow chart of all possible actions in a diagram such as the one shown below (where $m = 1$):



nodes;
weights

Each point $(n, \text{length of } x(n))$ in the SUMMARY diagram is called a *node*.

The numbers on the line segments between nodes in the SUMMARY diagram are called the *weights*.

analysis of
the SUMMARY
diagram

Look at the SUMMARY diagram for $m = 1$. At step 0, the list $x(0)$ has length 1, indicated by the node $(0, 1)$. A W in step 1 results in a STOP, indicated by the node $(1, 0)$. A L in step 1 results in a new list $x(1)$ of length 2, indicated by the node $(1, 2)$.

analysis of
the node $(5, 3)$

The weights give the number of paths that emanate from the previous node. For example, consider the node $(5, 3)$ in the SUMMARY diagram. There are two paths in the flow chart that result in a list of length 3 at step 5 ending with a L : (L, L, W, L, L) and (L, L, L, W, L) . These two paths are indicated by the weight 2 on the lower line segment leading into the node $(5, 3)$.

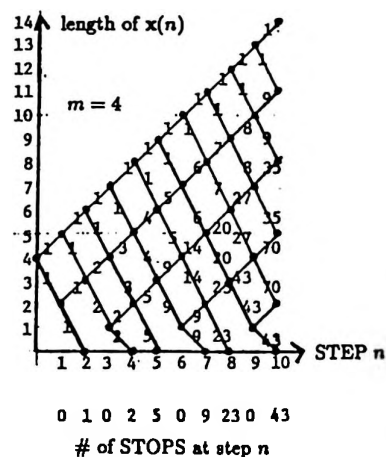
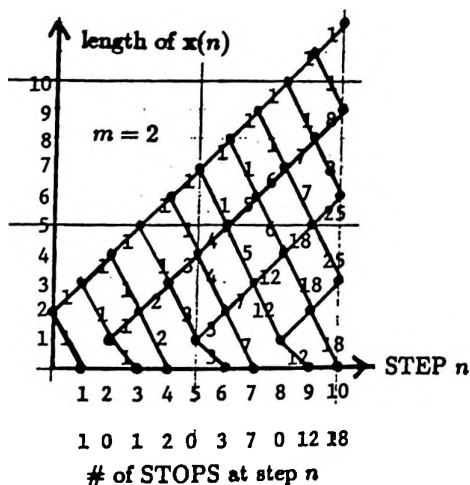
There is only one path in the flow chart that results in a list of length 3 at step 5 ending with a W : (L, L, L, L, W) . This path is indicated by the weight 1 on the upper line segment leading into the node $(5, 3)$.

Observe that each line segment leaving the node $(5, 3)$ has weight $2 + 1 = 3$.

STOPS in
the algorithm;
nodes of form
 $(n, 0)$

The nodes of the form $(n, 0)$ correspond to STOPS in the algorithm. The number of STOPS at step n is summarized along the bottom of the diagram, and is either zero, or equals the weight of the line segment leading into the node.

Two more SUMMARY diagrams are given below. The left diagram corresponds to starting length $m = 2$, and the right diagram corresponds to starting length $m = 4$.



The SUMMARY diagrams produced in this manner lend themselves nicely to an algorithm for computing the desired stopping probabilities, as follows. There are 'upward diagonals' in each SUMMARY diagram (that is, the lines that have slope 1). The weights on each of these diagonals can be associated with a list: the topmost diagonal corresponds to the list $(1, 1, 1, 1, 1, \dots)$ (for any m). The next diagonal down corresponds to the list $(1, 2, 3, 4, 5, \dots)$ (for any m). These lists are arranged in matrix form; every list (except the first) is prefaced with a 0 and started in the appropriate column, as illustrated below for the case $m = 4$:

1	1	1	1	1	1	1
0	1	2	3	4	5	6
		0	2	5	9	14
				0	9	

Let $M(i, j)$ denote the entry (or blank) in row i and column j of the arrangement above. For example, $M(2, 3) = 2$ and $M(3, 6) = 9$. There is no entry in $M(3, 1)$.

The following observations regarding this arrangement of the diagonal lists are important:

- Computation of the probability that the algorithm stops in at most N steps requires N columns.
- The starting positions of rows 2, 3, ... will depend on the starting length m .
- Once proper starting positions of the rows are computed, the remaining list entries are easily determined as indicated in the example below:

1	1	1	1	1	1	1
0	1	2	3	4	5	6
		0	2	5	9	14
				0	9	

- The number of STOPS at each step is determined by the lowest entry in each column.

1	1	1	1	1	1	1
0	1	2	3	4	5	6
↓	↓	↓	↓	↓	↓	↓
0	1	0	2	5	0	9

computing a
sample probability

A sample probability is now computed. Let $m = 4$, and suppose it is desired to find the probability that the algorithm stops in at most 5 steps; denote this by $\text{Prob}(N \leq 5)$. Let $\text{Prob}(N = i)$ denote the probability that the algorithm stops in step i (for $i \geq 1$). Then,

$$\begin{aligned}\text{Prob}(N \leq 5) &= \sum_{i=1}^5 \text{Prob}(N = i) \\ &= \sum_{i=1}^5 (\# \text{ of STOPS in step } i)(\text{probability of each STOP}) \\ &= (0)\left(\frac{1}{2}\right) + (1)\left(\frac{1}{2}\right)^2 + (0)\left(\frac{1}{2}\right)^3 + (2)\left(\frac{1}{2}\right)^4 + (5)\left(\frac{1}{2}\right)^5 \\ &\approx 0.5313 .\end{aligned}$$

Thus, there is a 53% chance that the algorithm will stop in at most 5 steps, when beginning with a list of length 4.

**MATLAB
function**

`prstop(m,N)`

The following MATLAB function utilizes the previous observations to compute the desired probabilities.

The reader should store this program in an `m`-file named `prstop.m`. Then, typing

`p = prstop(m,N)`

from within MATLAB returns a number `p`, where `p` is the probability that the Martingale Algorithm stops in less than or equal to `N` steps, starting with a list of length `m`, and assuming that, at each step, $\text{Prob}(W) = \text{Prob}(L) = 0.5$.

The table following the program lists the probabilities

$$\text{prstop}(m,N), \quad m = 1, \dots, 20, \quad N = 1, \dots, 20 .$$

The entry in row `m` and column `N` is `prstop(m,N)`.

```
% copyright 1993 by Carol J.V. Fisher
%
function PROB = prstop(m,N)
%
% This function computes the probability that the Martingale Algorithm
% will stop in less than or equal to N steps, beginning with
% a series of length m.
% It is assumed that the probability of WINNING, at every step, is 0.5.
%
% Compute the starting position of row i; store this in ST(i).
% If ST(i) > N, then STOP and record the needed number of ROWS in R.
L = m;
C = 0;
i = 1;
ST(1) = 1;
```

```

for k = 1:(N+1)
    while L-2 <= 0
        L = L+1;
        C = C+1;
    end
    C = C+1;
    L = L-2;
    i = i+1;
    ST(i) = C;
    if ST(i) > N
        R = i-1;
        break
    end
end
% Initialize the matrix that will hold all the rows:
M = zeros(R,N);
% Initialize the first and second rows.
% The first row always consists entirely of 1s.
M(1,:) = ones(1,N);
% The second row always looks like 0 1 2 3 ..., starting in column ST(2)
if R > 1
M(2,(ST(2):N)) = [0:(N-ST(2))];
end
% The remaining rows are computed as follows:
% Row k must begin in column ST(k) with a 0.
% Successive entries are computed as: M(k,j) = M(k,j-1) + M(k-1,j-1)
for k = 3:R
    M(k,ST(k)) = 0;
    for j = (ST(k)+1):N
        M(k,j) = M(k,j-1) + M(k-1,j-1);
    end
end
% Compute the number of STOPS for every i; store
% these in the matrix NST(i).
% First, take care of the trivial case where there is only one row:
if R == 1
    for j = 1:R
        NST(j) = M(1,j);
    end
end
% Now, the more interesting case:
for i = 1:(R-1)
    for j = ST(i):(ST(i+1)-1)
        NST(j) = M(i,j);
    end
end
for j = ST(R):N
    NST(j) = M(R,j);
end
% Finally, compute the desired probability:
i = 1:N;
P = (.5).^i;
PROB = sum(P.*NST);

```

prstop(m,N) is in row m and column N

	N = 1	N = 2	N = 3	N = 4	N = 5	N = 6	N = 7	N = 8	N = 9	N = 10
m=1	0.5000	0.7500	0.7500	0.8125	0.8750	0.8750	0.8984	0.9258	0.9258	0.9375
m=2	0.5000	0.5000	0.6250	0.7500	0.7500	0.7969	0.8516	0.8516	0.8750	0.9043
m=3	0	0.2500	0.5000	0.5000	0.5938	0.7031	0.7031	0.7500	0.8086	0.8086
m=4	0	0.2500	0.2500	0.3750	0.5313	0.5313	0.6016	0.6914	0.6914	0.7334
m=5	0	0	0.1250	0.3125	0.3125	0.4063	0.5313	0.5313	0.5918	0.6729
m=6	0	0	0.1250	0.1250	0.2188	0.3594	0.3594	0.4336	0.5371	0.5371
m=7	0	0	0	0.0625	0.1875	0.1875	0.2656	0.3828	0.3828	0.4463
m=8	0	0	0	0.0625	0.0625	0.1250	0.2344	0.2344	0.3008	0.4014
m=9	0	0	0	0	0.0313	0.1094	0.1094	0.1680	0.2656	0.2656
m=10	0	0	0	0	0.0313	0.0313	0.0703	0.1484	0.1484	0.2021
m=11	0	0	0	0	0	0.0156	0.0625	0.0625	0.1035	0.1787
m=12	0	0	0	0	0	0.0156	0.0156	0.0391	0.0918	0.0918
m=13	0	0	0	0	0	0	0.0078	0.0352	0.0352	0.0625
m=14	0	0	0	0	0	0	0.0078	0.0078	0.0215	0.0557
m=15	0	0	0	0	0	0	0	0.0039	0.0195	0.0195
m=16	0	0	0	0	0	0	0	0.0039	0.0039	0.0117
m=17	0	0	0	0	0	0	0	0	0.0020	0.0107
m=18	0	0	0	0	0	0	0	0	0.0020	0.0020
m=19	0	0	0	0	0	0	0	0	0	0.0010
m=20	0	0	0	0	0	0	0	0	0	0.0010

	N = 11	N = 12	N = 13	N = 14	N = 15	N = 16	N = 17	N = 18	N = 19	N = 20
m=1	0.9521	0.9521	0.9589	0.9676	0.9676	0.9718	0.9773	0.9773	0.9800	0.9837
m=2	0.9043	0.9177	0.9352	0.9352	0.9435	0.9546	0.9546	0.9601	0.9675	0.9675
m=3	0.8354	0.8704	0.8704	0.8870	0.9092	0.9092	0.9201	0.9349	0.9349	0.9423
m=4	0.7886	0.7886	0.8152	0.8509	0.8509	0.8685	0.8925	0.8925	0.9046	0.9212
m=5	0.6729	0.7126	0.7666	0.7666	0.7935	0.8304	0.8304	0.8491	0.8750	0.8750
m=6	0.5898	0.6628	0.6628	0.7000	0.7516	0.7516	0.7781	0.8151	0.8151	0.8342
m=7	0.5371	0.5371	0.5848	0.6524	0.6524	0.6877	0.7376	0.7376	0.7637	0.8007
m=8	0.4014	0.4570	0.5382	0.5382	0.5819	0.6448	0.6448	0.6783	0.7264	0.7264
m=9	0.3242	0.4136	0.4136	0.4637	0.5379	0.5379	0.5785	0.6377	0.6377	0.6698
m=10	0.2900	0.2900	0.3427	0.4234	0.4234	0.4693	0.5378	0.5378	0.5758	0.6318
m=11	0.1787	0.2283	0.3086	0.3086	0.3567	0.4308	0.4308	0.4733	0.5373	0.5373
m=12	0.1323	0.2036	0.2036	0.2496	0.3237	0.3237	0.3681	0.4369	0.4369	0.4766
m=13	0.1172	0.1172	0.1565	0.2240	0.2240	0.2669	0.3360	0.3360	0.3774	0.4418
m=14	0.0557	0.0847	0.1394	0.1394	0.1772	0.2412	0.2412	0.2815	0.3463	0.3463
m=15	0.0371	0.0752	0.0752	0.1049	0.1587	0.1587	0.1950	0.2558	0.2558	0.2939
m=16	0.0332	0.0332	0.0532	0.0934	0.0934	0.1230	0.1756	0.1756	0.2105	0.2684
m=17	0.0107	0.0217	0.0474	0.0474	0.0689	0.1100	0.1100	0.1394	0.1905	0.1905
m=18	0.0063	0.0195	0.0195	0.0329	0.0614	0.0614	0.0838	0.1252	0.1252	0.1541
m=19	0.0059	0.0059	0.0126	0.0294	0.0294	0.0445	0.0748	0.0748	0.0977	0.1391
m=20	0.0010	0.0010	0.0114	0.0114	0.0201	0.0397	0.0397	0.0560	0.0876	0.0876

2.2 Tests for Specific Conjectured Components: Linear Least-Squares Approximation

Introduction

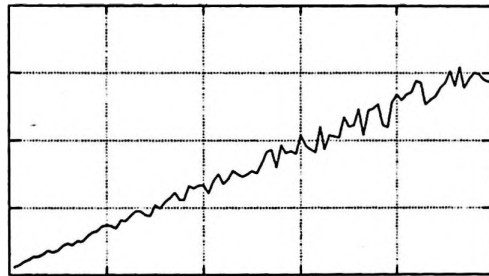
In certain instances, a researcher may conjecture that a data set $\{(t_i, y_i)\}_{i=1}^N$ will be well-modeled by a function of a specified form. This conjecture may arise from a combination of *experience*, *initial data inspection*, or *knowledge of the mechanisms generating the data*.

For example, the data set graphed below (left) shows a linear trend, and one might reasonably seek constants m and b so that the function $f(x) = mx + b$ 'fits' the data well. (The notion of 'fit' is addressed precisely in this section.)

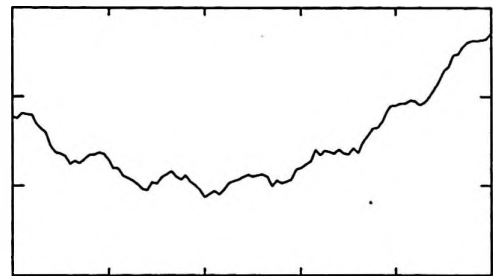
As a second example, the data set $\{(t_i, y_i)\}_{i=1}^N$ graphed below (right) appears to have a global quadratic component. There also seems to be some interesting local periodic behavior—the curve seems to oscillate about a parabola. These observations might lead one to suspect that there are constants a , b and c for which the function

$$y(t) = \overbrace{at^2 + bt + c}^{\text{quadratic component}} + \text{< some periodic component >}$$

will give a good 'fit' to the data. A researcher might attempt to identify the quadratic component, and then subtract it off, in order to isolate and study the more local periodic behavior.



LINEAR TREND



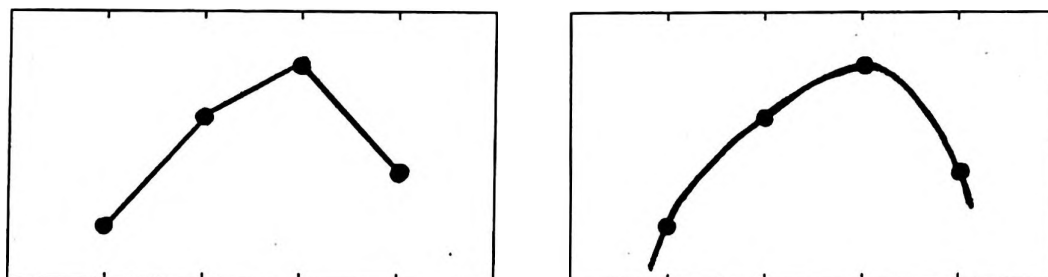
QUADRATIC TREND

'fitting' data;
interpolation
(fitting function
passes through
each data point)

There are many ways to quantify the notion of 'fitting' a data set $\{(t_i, y_i)\}_{i=1}^N$ with a function f . One very strong condition is to require that the function f pass through every data point; i.e., $f(t_i) = y_i$ for all i . The process of finding such a function is called *interpolation*; and the resulting function is called an *interpolate* (of the data set).

As illustrated below, there are many possible interpolates for a given data set. Usually, the interpolating function is selected to have some additional desirable properties. For example, the interpolate may be required to be a single polynomial (see Section 2.5). Polynomial interpolation has the disadvantage that one often gets large oscillations between the data points. To avoid such oscillations, one can alternately use cubic polynomials that are appropriately 'patched together' at the data points—the resulting interpolate is called a *cubic spline*, and is also discussed in Section 2.5. Interpolates are often used to supply missing data values.

The exact matching required in interpolation may not be possible if a function of a specific form is desired; for example, no linear function $f(x) = mx + b$ can be made to pass through three non-collinear points. More importantly, such exact matching *may not even be justified* in situations where each data value is potentially 'contaminated' by noise. In such cases, it often makes more sense to seek a function that merely 'comes close' to the data points—this leads to a method of 'fitting' a data set that is commonly called *approximation*.



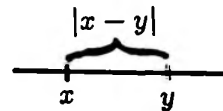
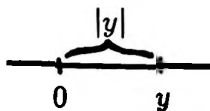
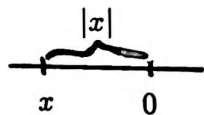
TWO INTERPOLATES OF A DATA SET

'fitting' data;
'objects being close'
is made precise
via a norm

One way to make precise the notion of 'objects being close' in mathematics is by the use of a *norm*. Roughly, a *norm* is a function that measures the size of an object; the 'size' of x is often denoted by $\|x\|$ and read as '*the norm of x*'. The norm (and underlying vector space structure) then allows one to talk about the distance between objects; the distance between x and y is $\|x - y\|$. The objects x and y are 'close' (with respect to the given norm) if $\|x - y\|$ is small. Appendix 2 gives the precise definitions of a *norm*, a *vector space*, and related results.

EXAMPLE
a norm on \mathbb{R}

For example, the absolute value function is a norm on the real numbers. For every real number x , the nonnegative number $|x|$ 'measures' x by giving its distance from zero on the real number line. Given any real numbers x and y , the distance between them is $|x - y|$; so x and y are 'close' (with respect to the absolute value norm) if $|x - y|$ is small.



EXAMPLE
a norm on \mathbb{R}^n

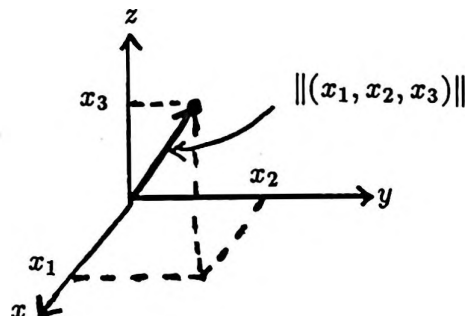
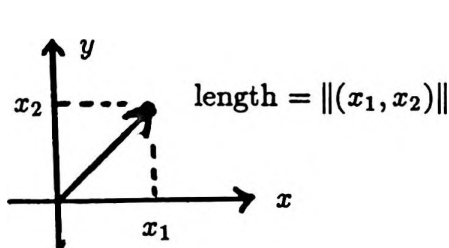
Let \mathbb{R}^n denote all n -tuples of real numbers,

$$\mathbb{R}^n := \{(x_1, \dots, x_n) \mid x_i \in \mathbb{R}, 1 \leq i \leq n\}.$$

Addition and subtraction of n -tuples is done componentwise. Elements of \mathbb{R}^n are commonly called *vectors* (since \mathbb{R}^n is a vector space). Let \mathbf{x} denote a typical element in \mathbb{R}^n . A norm on \mathbb{R}^n is given by

$$\|\mathbf{x}\| := \sqrt{x_1^2 + \dots + x_n^2}. \quad (1)$$

This norm is called the *Euclidean norm*; the Euclidean norm of an n -tuple is the square root of the sum of its squared entries. When n equals 2 or 3, the nonnegative number $\|\mathbf{x}\|$ has a nice geometric interpretation—it gives the length of the arrow from the origin to the point with coordinates \mathbf{x} . Two n -tuples \mathbf{x} and \mathbf{y} are 'close' (with respect to the given norm) if $\|\mathbf{x} - \mathbf{y}\|$ is small.



in the context of
matrix
manipulations,
elements of \mathbb{R}^n
are viewed as
column vectors

In this dissertation, whenever an element of \mathbb{R}^n is used in the context of matrix manipulations, it will be represented as a *column vector*.

Letting \mathbf{y} be the column vector

$$\mathbf{y} := \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix},$$

then \mathbf{y}^t (\mathbf{y} transpose) is the row vector

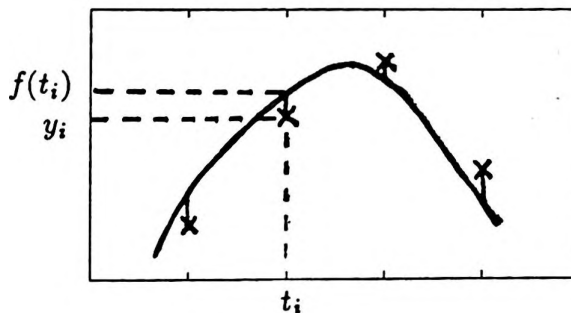
$$[y_1 \ y_2 \ \cdots \ y_n],$$

and the Euclidean norm of \mathbf{y} , squared, is displayed in matrix notation as:

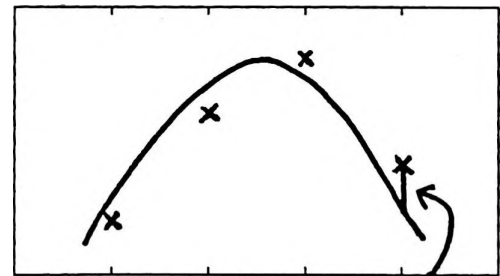
$$\begin{aligned} \|\mathbf{y}\|^2 &= y_1^2 + \cdots + y_n^2 \\ &= [y_1 \ y_2 \ \cdots \ y_n] \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} \\ &= \mathbf{y}^t \mathbf{y}. \end{aligned}$$

approximation
(fitting function
approximates the
data set
relative to a
given norm)

A function f is said to *approximate* a data set $\{(t_i, y_i)\}_{i=1}^N$ if it is 'close to' the data set, in some normed sense. The process of finding such a function is called *approximation*. The sketches below illustrate two types of approximation. *Least-squares approximation*, discussed in this and the next two sections, is so-called because it seeks a function that minimizes the sum of the squared distances between the approximating function and the data set. *Minimax* (or *Chebyshev*) approximation seeks to minimize the maximum distance between the data set and approximating function; although simply stated, the mathematics is difficult, and minimax approximation is not discussed here.



$$\underset{f}{\text{minimize}} \left(\sum_i (y_i - f(t_i))^2 \right)$$



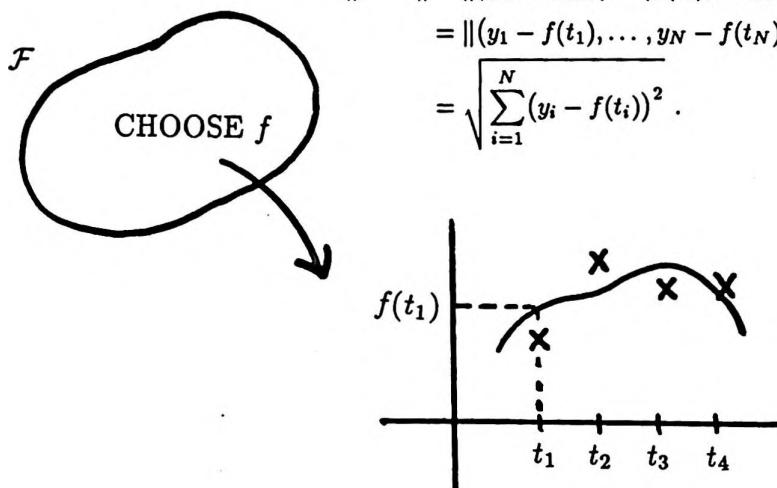
$$\underset{f}{\text{minimize}} \left(\max_i |y_i - f(t_i)| \right)$$

*least-squares
approximation*

The *method of least squares* arises when the Euclidean norm (1) is used for approximation, as follows.

Let \mathcal{F} be a set of real-valued functions of one real variable, the members of which are to serve as candidates for approximation of the data set $\{(t_i, y_i)\}_{i=1}^N$. Denote a typical function in \mathcal{F} by f , and let \mathbf{f} denote the N -tuple $(f(t_1), \dots, f(t_N))$, formed by letting f act on the N time values (t_1, \dots, t_N) . The N -tuple of data values $\mathbf{y} := (y_1, \dots, y_N)$ can be compared to the N -tuple \mathbf{f} via the Euclidean norm given in (1):

$$\begin{aligned} \|\mathbf{y} - \mathbf{f}\| &= \|(y_1, \dots, y_N) - (f(t_1), \dots, f(t_N))\| \\ &= \|(y_1 - f(t_1), \dots, y_N - f(t_N))\| \\ &= \sqrt{\sum_{i=1}^N (y_i - f(t_i))^2}. \end{aligned} \quad (2)$$



*a least-squares
approximate*

The sum in (2) is nicely interpreted as the *error between the function f and the data set*. Allowing f to vary over all the members in \mathcal{F} , one can investigate the corresponding errors. If a function \hat{f} can be found that minimizes this error, then it is called a *least-squares approximate, from \mathcal{F} , to the data set*. For such a function \hat{f} (with corresponding N -tuple $\hat{\mathbf{f}} := (\hat{f}(t_1), \dots, \hat{f}(t_N))$),

$$\|\mathbf{y} - \hat{\mathbf{f}}\| = \min_{f \in \mathcal{F}} \|\mathbf{y} - \mathbf{f}\|,$$

so that, for all $f \in \mathcal{F}$,

$$\|\mathbf{y} - \hat{\mathbf{f}}\| \leq \|\mathbf{y} - \mathbf{f}\|.$$

Since \hat{f} minimizes the error, it is a best approximation to the data set from \mathcal{F} . If \hat{f} is unique, then it is *the* best approximation from \mathcal{F} .

existence and uniqueness of the least-squares approximate

If the set \mathcal{F} contains a finite number of functions, then such a minimizing function \hat{f} can always be found (but may not be unique). In most interesting applications of least-squares approximation, however, the set \mathcal{F} contains an infinite number of functions, and existence and uniqueness of a best approximation is more delicate.

$\min_{x \in X} \|x\|$
is equivalent to
 $\min_{x \in X} \|x\|^2$

The appearance of the square root in (2) is bothersome, and can be avoided by a simple observation, which is:

LEMMA

Let X be any set on which a norm is defined. If there exists $\hat{x} \in X$ for which

$$\|\hat{x}\| = \min_{x \in X} \|x\| ,$$

then

$$\|\hat{x}\|^2 = \min_{x \in X} \|x\|^2 .$$

Conversely, if $\|\hat{x}\|^2 = \min_{x \in X} \|x\|^2$, then $\|\hat{x}\| = \min_{x \in X} \|x\|$.

PROOF

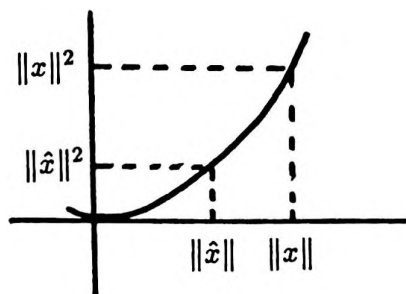
Suppose there exists $\hat{x} \in X$ for which $\|\hat{x}\| = \min_{x \in X} \|x\|$. Then,

$$\|\hat{x}\| \leq \|x\| \text{ for all } x \in X .$$

Since the squaring function $f(x) = x^2$ increases on $[0, \infty)$, it follows that

$$\|\hat{x}\|^2 \leq \|x\|^2 \text{ for all } x \in X ,$$

from which $\|\hat{x}\|^2 = \min_{x \in X} \|x\|^2$. The converse uses the fact that the square root function increases on $[0, \infty)$. ■



With this observation, the approximation problem

$$\min_{f \in \mathcal{F}} \|y - f\|$$

can always be replaced by the equivalent problem

$$\min_{f \in \mathcal{F}} \|y - f\|^2 ,$$

whenever it is convenient to do so. Such a replacement eliminates the bothersome square root sign in (2).

A frequently-occurring case of least-squares approximation is discussed next, together with its MATLAB implementation.

*linear
least-squares
approximation*

Suppose that a data set $\{(t_i, y_i)\}_{i=1}^N$ and m real-valued functions of one real variable, f_1, \dots, f_m , are given. Let

$$\mathcal{F} := \{c_1 f_1 + \dots + c_m f_m \mid c_i \in \mathbb{R}, 1 \leq i \leq m\}$$

be the class of functions to be used in approximating the data set. Note that each function in \mathcal{F} is a linear combination of the functions f_1, \dots, f_m ; hence the name *linear least-squares approximation*. However, the functions f_i certainly need not be linear functions; in many practical applications, the f_i will be polynomials or sinusoids. The function f_1 is usually taken to be $f_1(t) \equiv 1$, to allow for a constant term in the approximating function.

The goal of least-squares approximation is to find real constants b_1, \dots, b_m so that the function f given by

$$f(t) := b_1 f_1(t) + b_2 f_2(t) + \dots + b_m f_m(t)$$

minimizes the sum

$$\sum_{i=1}^N (y_i - f(t_i))^2 .$$

Evaluation of the function f at the N time values t_1, \dots, t_N yields N equations

$$\begin{aligned} f(t_1) &= b_1 f_1(t_1) + b_2 f_2(t_1) + \dots + b_m f_m(t_1) \\ f(t_2) &= b_1 f_1(t_2) + b_2 f_2(t_2) + \dots + b_m f_m(t_2) \\ &\vdots \\ f(t_N) &= b_1 f_1(t_N) + b_2 f_2(t_N) + \dots + b_m f_m(t_N) . \end{aligned} \tag{3}$$

By defining

$$\mathbf{f} := \begin{bmatrix} f(t_1) \\ \vdots \\ f(t_N) \end{bmatrix}, \quad \mathbf{X} := \begin{bmatrix} f_1(t_1) & \cdots & f_m(t_1) \\ f_1(t_2) & \cdots & f_m(t_2) \\ \vdots & \vdots & \vdots \\ f_1(t_N) & \cdots & f_m(t_N) \end{bmatrix}, \quad \text{and} \quad \mathbf{b} := \begin{bmatrix} b_1 \\ \vdots \\ b_m \end{bmatrix},$$

the system in (3) can be written as

$$\mathbf{f} = \mathbf{X}\mathbf{b}.$$

The sizes of the matrices \mathbf{f} , \mathbf{X} , and \mathbf{b} are, respectively, $N \times 1$, $N \times m$, and $m \times 1$.

Define \mathbf{y} to be the column vector of data values,

$$\mathbf{y} := \begin{bmatrix} y_1 \\ \vdots \\ y_N \end{bmatrix}.$$

Then, \mathbf{y} has size $N \times 1$, as does $\mathbf{y} - \mathbf{X}\mathbf{b}$.

*minimizing
the error*

The least-squares problem is to minimize

$$\text{SSE} := \|\mathbf{y} - \mathbf{f}\|^2 = \|\mathbf{y} - \mathbf{X}\mathbf{b}\|^2 = (\mathbf{y} - \mathbf{X}\mathbf{b})'(\mathbf{y} - \mathbf{X}\mathbf{b}),$$

where SSE stands for 'sum of the squared errors'. SSE is to be viewed as a function from \mathbb{R}^m into \mathbb{R} ; it takes an m -tuple \mathbf{b} and maps it to the real number $(\mathbf{y} - \mathbf{X}\mathbf{b})'(\mathbf{y} - \mathbf{X}\mathbf{b})$. Indeed, SSE is a differentiable function of \mathbf{b} [G&VL, 221ff]. Therefore, a necessary condition to minimize SSE is that its partial derivatives with respect to the components b_i of \mathbf{b} equal zero. Appendix 3 shows that the partial derivatives of SSE are contained in the m -column vector

$$2(\mathbf{X}'\mathbf{X}\mathbf{b} - \mathbf{X}'\mathbf{y}).$$

Therefore, a minimizing vector \mathbf{b} must satisfy

$$\mathbf{0} = 2(\mathbf{X}'\mathbf{X}\mathbf{b} - \mathbf{X}'\mathbf{y}),$$

or, equivalently,

$$\mathbf{X}'\mathbf{X}\mathbf{b} = \mathbf{X}'\mathbf{y}.$$

the least-squares
parameters \mathbf{b}

If $\mathbf{X}'\mathbf{X}$ is invertible, then

$$\mathbf{b} = (\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\mathbf{y}$$

gives the unique least-squares parameters. Observe that the matrix $\mathbf{X}'\mathbf{X}$ is an $m \times m$ real symmetric matrix; the invertibility of $\mathbf{X}'\mathbf{X}$ is discussed in Section 2.3 and Appendix 2.

A MATLAB procedure for finding the least-squares parameters \mathbf{b} is discussed next.

MATLAB IMPLEMENTATION

Linear Least-Squares Approximation

PURPOSE and NOTATION

Let $\{(t_i, y_i)\}_{i=1}^N$ be a given data set, and let f_1, \dots, f_m be m real-valued functions of a real variable.

Define:

$$\begin{aligned} \mathbf{t} &:= (t_1, \dots, t_N), \\ \mathbf{y} &:= (y_1, \dots, y_N), \\ f(t) &:= b_1 f_1(t) + \dots + b_m f_m(t) \text{ for } b_i \in \mathbb{R}, 1 \leq i \leq m, \text{ and} \\ \mathbf{f} &:= (f(t_1), \dots, f(t_N)). \end{aligned}$$

The linear least-squares approximation problem is to find specific choices for b_1, \dots, b_m so that the error

$$\|\mathbf{y} - \mathbf{f}\|^2 = \sum_{i=1}^N (y_i - f(t_i))^2$$

is minimized.

REQUIRED INPUTS

- the N time values t_i are stored in an N -column vector called \mathbf{t} ;
- the N data values y_i are stored in an N -column vector called \mathbf{y} .

**MATLAB
COMMANDS**

The following list of MATLAB commands will produce (under suitable conditions) a column vector \mathbf{b} with entries b_1, \dots, b_m so that $f(t) = b_1 f_1(t) + \dots + b_m f_m(t)$ gives the least-squares approximate to the data set. The actual data set is plotted on the same graph with the best approximating function, and the least-squares error is computed.

The example following these commands explains and illustrates and procedure.

```
f1 = < insert definition of f1 here >
f2 = < insert definition of f2 here >
:
fm = < insert definition of fm here >
X = [f1 f2 ... fm];
b = (X'*X) \ (X'*y)
f = X*b;
plot(t,y,'x',t,f,'.')
ERROR = (y - f)'*(y - f)
```

EXAMPLE

The following list of MATLAB commands produces a set of 'noisy' data of the form

$$y(t) = 3.2 - .7t + .05t^2 + \sin \frac{2\pi t}{3} + \text{noise} ,$$

which is the 'known unknown'. A least-squares approximate of the form

$$f(t) = b_1 + b_2 t + b_3 t^2 + b_4 t^3 + b_5 t^4$$

is computed. There are 5 approximating component functions:

$$f_1(t) = 1 , \quad f_2(t) = t , \quad f_3(t) = t^2 , \quad f_4(t) = t^3 , \quad f_5(t) = t^4 .$$

The actual data set is graphed together with its least-squares approximate, and the error is computed.

Based on an analysis of the difference between the actual data and the approximate, the form of the approximating function is updated, and a second least-squares approximate is obtained, which gives a beautiful fit to the actual data.

The lines are numbered for easy reference in the discussion that follows.


```

1) t = [0:.1:20]';
2) N = .5*(2*rand(t) - 1);
3) y = 3.2 - .7*t + .05*t.^2 + 2*sin(2*pi*t/3) + N;
4) plot(t,y)
5) f1 = ones(t);
6) f2 = t;
7) f3 = t.^2;
8) f4 = t.^3;
9) f5 = t.^4;
10) X = [f1 f2 f3 f4 f5];
11) b = (X'*X) \ (X'*y)

```

b =

```

4.1736
-1.2540
0.1486
-0.0069
0.0002

```

```

12) f = X*b;
13) plot(t,y,'x',t,f,'.')
14) ERROR = (y - f)'*(y - f)

```

ERROR =

```
403.4238
```

```

15) DIFF = y - f;
16) plot(t,DIFF)
17) f6 = sin(2*pi*t/3);
18) X = [X f6];
19) b = (X'*X) \ (X'*y)

```

b =

```

3.1465
-0.6713
0.0501
-0.0004
0.0000
1.9988

```

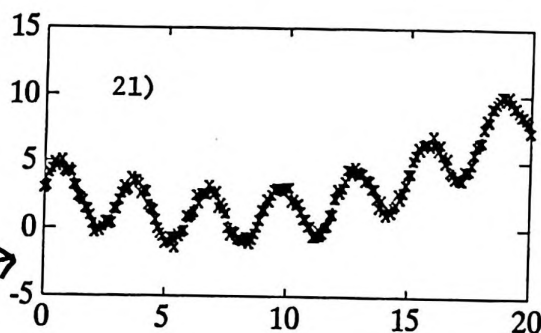
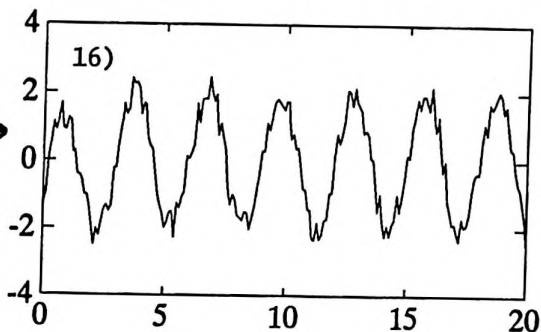
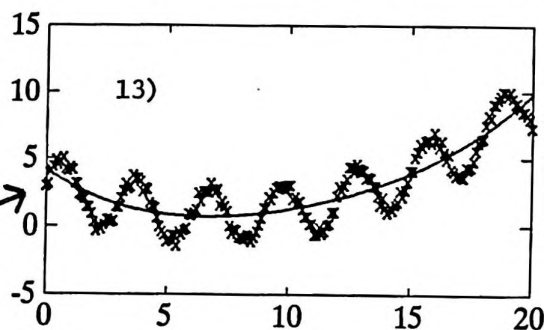
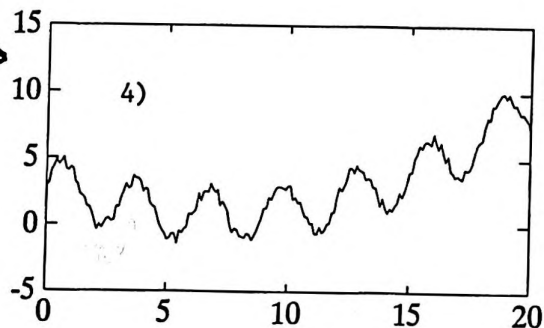
```

20) f = X*b;
21) plot(t,y,'x',t,f,'.')
22) ERROR = (y - f)'*(y - f)

```

ERROR =

```
16.0330
```



analysis of
each line

- 1) A time vector is created; entries begin at 0, are evenly spaced with spacing 0.1, and end at 20. The prime mark ' denotes the transpose operation, and makes \mathbf{t} a column vector. The semicolon (;) at the end of the line suppresses MATLAB echoing.
- 2) \mathbf{r} is a column vector the same size as \mathbf{t} with entries uniformly distributed on $[-.5, .5]$.
- 3) The data values \mathbf{y} are computed. Recall that $\mathbf{t}.^2$ gives a MATLAB array operation; each element in \mathbf{t} is squared.
- 4) The data set is plotted as a line graph. That is, MATLAB 'connects the points' with a curve.
- 5) The first function is $f_1(t) \equiv 1$. The MATLAB command `ones(t)` produces a vector the same size as \mathbf{t} , with all entries equal to 1.
- 6-9) The functions $f_2(t) = t$, $f_3(t) = t^2$, $f_4(t) = t^3$ and $f_5(t) = t^4$ are evaluated at the appropriate time values.
- 10) The matrix \mathbf{X} is formed by laying the column vectors \mathbf{f}_1 through \mathbf{f}_5 next to each other.
- 11) The least-squares parameters \mathbf{b} are computed. The backslash operator `\` provides a way of doing 'matrix division' that is more efficient than calculating the inverse of a matrix. More precisely, solving $\mathbf{A}\mathbf{x} = \mathbf{b}$ via the MATLAB command

$$\mathbf{x} = \text{inv}(\mathbf{A}) * \mathbf{b}$$

would cause MATLAB to calculate the inverse of \mathbf{A} , and then multiply by \mathbf{b} . However, solving $\mathbf{A}\mathbf{x} = \mathbf{b}$ via the MATLAB command

$$\mathbf{x} = \mathbf{A} \setminus \mathbf{b}$$

(to understand this, read it from right to left as ' \mathbf{b} divided by \mathbf{A} ') solves the system using Gaussian elimination, without ever calculating the inverse of \mathbf{A} . This method is greatly preferred from a numerical accuracy and efficiency standpoint.

Since there is no semicolon at the end of the line to suppress MATLAB echoing, the column vector \mathbf{b} is printed.

- 12) The least-squares approximate is computed at the time values of the data set, and called \mathbf{f} .
- 13) The actual data is point-plotted with the symbol ' \mathbf{x} '; on the same graph, the least-squares approximate is point plotted with the symbol ' $\mathbf{.}$ '.

- 14) The least-squares error is computed and printed.
- 15–16) The difference between the actual data and approximating function is computed as DIFF, and plotted.
- 17–18) Visual inspection shows that the difference seems to be a sinusoid of period 3. (Alternately, techniques discussed later on in this dissertation can be used to reach this conclusion; see Section 2.6). Thus, least-squares is applied again with an additional function $f_6(t) = \sin \frac{2\pi t}{3}$. The new column vector \mathbf{f}_6 is computed, and \mathbf{x} is updated by concatenating this new column vector.
- 19–21) The new least-squares parameters and function are computed, and a graph showing the ‘fit’ is plotted.
- 22) The new ERROR is computed, confirming that a beautiful fit has been achieved.

If desired, the procedure described here can be automated into a MATLAB M-file.

$X^t X$ is often
‘close to’
noninvertible

As the number of component functions f_1, f_2, \dots, f_m increases, the matrix $X^t X$ often approaches singularity; that is, it looks more and more like a noninvertible matrix. This can cause the procedure just described to break down. In the next section, the matrix $X^t X$ is studied in more detail, and the concept of *orthogonal functions* is introduced as a method of overcoming the ‘near singularity’ of $X^t X$.

2.3 Computer Application Considerations for Linear Least-Squares

Condition Numbers

*theory
versus
application*

In the previous section, it was shown that the problem

$$\min_{b \in \mathbb{R}^n} \|y - Xb\|^2$$

has a unique solution, provided that the square matrix $X'X$ is invertible. Theoretically (using exact arithmetic), the invertibility of a square matrix A is easily characterized in a variety of ways: for example, A is invertible if and only if the determinant of A (denoted by $\det(A)$) is nonzero, and in this case, the unique solution to $Ax = b$ is given by $x = A^{-1}b$. However, when the entries of a matrix are represented only to the degree of accuracy of a digital computer, the question of invertibility is more subtle: there are invertible matrices A for which correct numerical solutions to $Ax = b$ are difficult to obtain, stemming from the fact that small changes in the matrices A and b can lead to large changes in the solution x .

EXAMPLE
*MATLAB casts
suspicion on
a solution vector*

To illustrate this point, reconsider the MATLAB example of the previous section, where a 4th-order polynomial least-squares approximate of the data

$$y(t) = 3.2 - .7t + .05t^2 + \sin \frac{2\pi t}{3} + \text{< noise >}$$

was found. If, alternately, a 7th-order polynomial approximate was sought, the MATLAB user would have been faced with this warning, casting suspicion on the resulting vector b :

```
b = (X'*X) \ (X'*y)
Warning: Matrix is close to singular or badly scaled.
Results may be inaccurate. RCOND = 2.866189e-020
```

The size of $X'X$ is only 8×8 ; however, the entries of $X'X$ are such that the matrix does not lend itself nicely to numerical computation. This problem is the subject of the current section.

*the contents of
this section*

More precisely, this section gives an analysis of the system $Ax = b$ in order to understand the *sensitivity of solutions x to small changes in the matrices A and b* . A number $\text{cond}(A)$ will be defined that measures this sensitivity.

For a 'sensitive' system, correct numerical solution can be challenging, but the problem can often be overcome by replacing the system $Ax = b$ with an equivalent system $\bar{A}x = b$, where the new matrix \bar{A} is more suitable for numerical computations. This replacement requires a knowledge of discrete orthogonal functions, which are also discussed in this section.

$\phi: X \rightarrow Y$
is a map between
normed spaces

Let $\phi: X \rightarrow Y$ be a function between normed spaces. The reader is referred to Appendix 2 for a review of normed spaces. The symbol $\|\cdot\|$ will be used to denote both the X and Y norms, with context determining the correct interpretation.

relative error

Let $x \in X$, and let \bar{x} be an approximation to x . For $x \neq 0$, the *relative error of \bar{x}* (as an approximation to x) is defined to be the number

$$\frac{\|\bar{x} - x\|}{\|x\|};$$

similarly, for $\phi(x) \neq 0$, the *relative error of $\phi(\bar{x})$* (as an approximation to $\phi(x)$) is the number

$$\frac{\|\phi(\bar{x}) - \phi(x)\|}{\|\phi(x)\|}.$$

condition number
for a function
between
normed spaces

A desirable situation for the function ϕ is this: when the relative error in \bar{x} is small, so is the relative error in $\phi(\bar{x})$. In other words, small input errors lead to small output errors. To quantify this input/output relationship, the idea of a *condition number* is introduced:

DEFINITION
condition number
for $\phi: X \rightarrow Y$,
at $x \in X$

Let $\phi: X \rightarrow Y$ be a function between normed spaces, and let $x \in X$. If a real number $c > 0$ exists for which

$$\frac{\|\phi(\bar{x}) - \phi(x)\|}{\|\phi(x)\|} \leq c \frac{\|\bar{x} - x\|}{\|x\|}$$

for all \bar{x} sufficiently close to x , then c is called a *condition number* for ϕ at x .

interpreting the
condition number

This definition shows that c gives information about the am-plification of relative input errors. If c is small, then small input errors must produce small output errors. However, if c is large, or if no such positive number c exists, then small input errors *may* produce large output errors. In numerical problems, where numbers often inherently contain error (due to a forced finite computer representation), such considerations become important.

Note that the definition of *condition number* is dependent on both the norm in the domain X , and the norm in the codomain Y .

What are to
be viewed as
the 'input'
and 'output'
for the system
 $Ax = b$?

For an invertible matrix A , the system $Ax = b$ will be analyzed as follows: the matrices A and b are taken as the 'inputs' to the system, and a solution vector x is the 'output'. When the inputs A and b change slightly (say, by replacing theoretical matrices A and b by their computer representations), then it is desired to study how the solution x changes. The Euclidean norm $\|\cdot\|$ will be used to measure 'changes' in the $n \times 1$ matrix b . It is also necessary to have a means by which to measure changes in the matrix A ; this means will be provided by a matrix norm, defined next. In this next definition, \sup denotes the *supremum*, i.e., the *least upper bound*.

DEFINITION
the 2-norm
for matrices

Let A be any $n \times n$ matrix, and let $\|\cdot\|$ be the Euclidean norm on \mathbb{R}^n . The 2-norm of the matrix A is given by

$$\|A\|_2 := \sup_{\substack{x \in \mathbb{R}^n, \\ x \neq 0}} \frac{\|Ax\|}{\|x\|}.$$

It can be shown that the 2-norm is indeed a norm on the vector space of all $n \times n$ matrices. Consequently, if $\|A\|_2 = 0$, then $A = O$, where O denotes the $n \times n$ zero matrix. Thus, if $\|A\|_2 = 0$, then A is not invertible. Equivalently, if A is invertible, then $\|A\|_2 \neq 0$.

MATLAB
approximation to
 $\|A\|_2$

The number $\|A\|_2$ is approximated in MATLAB via the command

`norm(A)` .

There are other matrix norms (see, e.g., [S&B, p. 178]), many of which are more easily calculated than $\|\cdot\|_2$, but it will be seen that the 2-norm is particularly well suited to the current situation. The reader versed in functional analysis will recognize the 2-norm as the norm of the bounded linear operator $A: \mathbb{R}^n \rightarrow \mathbb{R}^n$, when the Euclidean norm is used in \mathbb{R}^n .

*an important
consequence of
the definition*

It is immediate from the definition of $\|A\|_2$ that for any $x \neq 0$,

$$\|A\|_2 \geq \frac{\|Ax\|}{\|x\|} ,$$

so that for all x ,

$$\|Ax\| \leq \|A\|_2 \cdot \|x\| . \quad (1)$$

Thus, the number $\|A\|_2$ gives the greatest magnification that a vector may attain under the mapping determined by A . That is, the norm of an output $\|Ax\|$ cannot exceed $\|A\|_2$ times the norm of the corresponding input $\|x\|$.

computing $\|A\|_2$

The 2-norm is not easy to compute. (★ When A has real number entries, $\|A\|_2$ is the square root of the largest eigenvalue of the matrix $A^t A$.) However, it is easy to obtain an upper bound for $\|A\|_2$ as the square root of the sum of the squared entries from A , as follows:

*getting an
upper bound
for $\|A\|_2$*

Let $x = (x_1, \dots, x_n)^t$, so that x is a column vector, and

$$\|x\| = \left(\sum_{i=1}^n x_i^2 \right)^{1/2} .$$

Let α_{ij} denote the element in row i and column j of the matrix A . The i^{th} component of the column vector Ax is denoted by $(Ax)_i$, and is the inner product of the i^{th} row of A with x :

$$(Ax)_i = \sum_{j=1}^n \alpha_{ij} x_j .$$

Then,

$$\begin{aligned}
 \|Ax\|^2 &= \sum_{i=1}^n (Ax)_i^2 = \sum_{i=1}^n \left(\sum_{j=1}^n \alpha_{ij} x_j \right)^2 \\
 &\leq \sum_{i=1}^n \left[\left(\sum_{j=1}^n \alpha_{ij}^2 \right)^{1/2} \left(\sum_{j=1}^n x_j^2 \right)^{1/2} \right]^2 \quad (\text{Cauchy-Schwarz}) \\
 &= \|x\|^2 \sum_{i=1}^n \sum_{j=1}^n \alpha_{ij}^2 .
 \end{aligned}$$

(The Cauchy-Schwarz Inequality is reviewed in Appendix 2.)

the Frobenius
norm, $\|A\|_F$,
is an
upper bound for
 $\|A\|_2$

Thus, for any $x \neq 0$, division by $\|x\|^2$ yields

$$\frac{\|Ax\|^2}{\|x\|^2} \leq \sum_{i=1}^n \sum_{j=1}^n \alpha_{ij}^2 ,$$

so that taking square roots gives

$$\|A\|_2 := \sup_{\substack{x \in \mathbb{R}^n \\ x \neq 0}} \frac{\|Ax\|}{\|x\|} \leq \left(\sum_{i=1}^n \sum_{j=1}^n \alpha_{ij}^2 \right)^{1/2} := \|A\|_F .$$

The square root of the sum of the squared entries from A gives another norm of A , called the *Frobenius norm*, and denoted by $\|A\|_F$ [G&VL, p. 56].

★
singular values
of A

Many of the ideas discussed in this section are conveniently expressed in terms of the *singular values* of A . In particular, the norms $\|A\|_2$ and $\|A\|_F$ have simple characterizations in terms of the singular values. The singular value decomposition of a matrix is discussed in Appendix 2.

With the matrix 2-norm in hand, the analysis of the system $Ax = b$ can now begin. It is assumed that A is an *invertible* $n \times n$ matrix with real number entries. It is desired to study the sensitivity of solutions x of this system to small changes in A and b .

*motivation for
the definition of
the
condition number
of a matrix*

[S&B, 178–180] Begin by supposing that only the column vector b changes slightly; A remains fixed. What is the corresponding effect on the solution x ? The analysis of this situation will motivate the definition of the *condition number of a matrix*.

Let

$$Ax = b$$

$$\text{and } A\bar{x} = \bar{b}$$

be true statements; that is, x denotes the system solution when column vector b is used, and \bar{x} denotes the system solution when column vector \bar{b} is used. Define

$$\Delta x := \bar{x} - x$$

$$\text{and } \Delta b := \bar{b} - b,$$

so that

$$A\Delta x = \Delta b.$$

Since A is invertible (by hypothesis),

$$\Delta x = A^{-1}\Delta b.$$

Taking norms and using property (1) gives

$$\|\Delta x\| = \|A^{-1}\Delta b\| \leq \|A^{-1}\|_2 \cdot \|\Delta b\|.$$

Let $x \neq 0$; division by the positive number $\|x\|$ gives

$$\frac{\|\Delta x\|}{\|x\|} \leq \frac{\|A^{-1}\|_2 \cdot \|\Delta b\|}{\|x\|}.$$

Since $b = Ax$, it follows that $\|b\| \leq \|A\|_2 \cdot \|x\|$ and hence $\frac{1}{\|x\|} \leq \frac{\|A\|_2}{\|b\|}$; substitution into the previous display yields

$$\frac{\|\Delta x\|}{\|x\|} \leq \|A^{-1}\|_2 \|A\|_2 \frac{\|\Delta b\|}{\|b\|}.$$

This inequality shows that the constant $\|A^{-1}\|_2 \|A\|_2$ gives information about how an error in b is potentially ‘amplified’ to produce an error in the solution x . Thus, the number $\|A^{-1}\|_2 \|A\|_2$ is a condition number for the function that ‘inputs’ a column vector b , and ‘outputs’ a solution to the system $Ax = b$. With this motivation, the next definition should come as no surprise:

DEFINITION Let A be an invertible matrix. The *condition number of A is condition number of a matrix*

$$\text{cond}(A) := \|A^{-1}\|_2 \cdot \|A\|_2 .$$

MATLAB
approximation to
`cond(A)`

The condition number $\text{cond}(A)$ is approximated in MATLAB via the command

$$\text{cond}(\mathbf{A}) .$$

properties of
`cond(A)`

When $\text{cond}(A)$ is close to 1, A is said to be *well-conditioned*, and when $\text{cond}(A)$ is large, A is said to be *ill-conditioned*. It is always true that $\text{cond}(A) \geq 1$, as follows. Since $\|A\|_2 \geq \frac{\|Ay\|}{\|y\|}$ for all $y \neq 0$, take y to be $A^{-1}x$, thus obtaining

$$\begin{aligned} \|A\|_2 &\geq \frac{\|A(A^{-1}x)\|}{\|A^{-1}x\|} \\ &= \frac{\|x\|}{\|A^{-1}x\|} \\ &\geq \frac{\|x\|}{\|A^{-1}\|_2 \|x\|} , \end{aligned}$$

from which

$$\begin{aligned} \|A\|_2 \|A^{-1}\|_2 &\geq \frac{\|x\|}{\|A^{-1}\|_2 \|x\|} \cdot \|A^{-1}\|_2 \\ &= 1 . \end{aligned}$$

orthogonal
matrix

If A is an orthogonal matrix (that is, $A^t A = I$, or equivalently, $A^{-1} = A^t$), then it can be shown that $\text{cond}(A) = 1$.

`rcond(A)`

When MATLAB issues warnings regarding ill-conditioned matrices, it reports the information in terms of the reciprocal of the condition number,

$$\text{rcond}(\mathbf{A}) := \frac{1}{\text{cond}(\mathbf{A})} .$$

If A is well-conditioned, then $\text{rcond}(\mathbf{A})$ is near 1.0; if A is ill-conditioned, then $\text{rcond}(\mathbf{A})$ is near 0.

It is important to note that, in the system $Ax = b$, the sensitivity of solutions x to small changes in b *clearly has more to do with A than with b* .

Since A is invertible if and only if $\det(A) \neq 0$, one might conjecture that when the determinant of A is close to 0, then solutions to $Ax = b$ are sensitive to small changes in A . However, this is not the case. The examples below show that $\det(A)$ can be made as small as desired, while keeping $\text{cond}(A) = 1$.

$X = \begin{pmatrix} 0.1000 & 0 \\ 0 & 0.1000 \end{pmatrix}$ $D = \det(X)$ $D = 0.0100$ $C = \text{cond}(X)$ $C = 1$	$X = \begin{pmatrix} 0.0100 & 0 \\ 0 & 0.0100 \end{pmatrix}$ $D = \det(X)$ $D = 1.0000e-004$ $C = \text{cond}(X)$ $C = 1$
--	---

*cond(A)
does indeed
measure
sensitivity to
changes in
BOTH A and b*

The definition of the condition number of a matrix was motivated by studying the sensitivity of solutions to $Ax = b$ to small changes in b . Does the number $\text{cond}(A)$, as defined, also measure the sensitivity of solutions to changes in *both* A and b ? The following discussion shows that the answer is 'Yes'.

[G&VL, 79-80] Allow BOTH A and b to vary in the system $Ax = b$; recall that A is assumed to be invertible.

Denote a change in A by ϵF , where F is an $n \times n$ constant matrix and $\epsilon > 0$. Similarly, denote a change in b by ϵf , where f is a fixed $n \times 1$ matrix. Different values of ϵ lead to different approximations $\tilde{A} := A + \epsilon F$ and $\tilde{b} := b + \epsilon f$ of A and b , respectively. Let $x(\epsilon)$ denote the solution of the 'perturbed' system $\tilde{A}x = \tilde{b}$; the situation is then summarized by

$$(A + \epsilon F)x(\epsilon) = b + \epsilon f, \quad x(0) = x.$$

Differentiating both sides with respect to ϵ yields

$$(A + \epsilon F)\dot{x}(\epsilon) + Fx(\epsilon) = f;$$

evaluation at $\epsilon = 0$ and a slight rearrangement yields

$$A\dot{x}(0) = f - Fx,$$

or, since A is invertible,

$$\dot{x}(0) = A^{-1}(f - Fx). \quad (2)$$

The Taylor expansion for x (as a function of the single variable ϵ) is

$$x(\epsilon) = x + \epsilon \dot{x}(0) + \overbrace{\frac{\ddot{x}(0)}{2!} \epsilon^2 + \frac{\ddot{x}(0)}{3!} \epsilon^3 + \dots}^{o(\epsilon)}.$$

The notation $o(\epsilon)$ is used to represent the fact that the indicated terms converge to 0 faster than ϵ converges to 0; that is,

$$\lim_{\epsilon \rightarrow 0} \frac{\frac{\ddot{x}(0)}{2!} \epsilon^2 + \frac{\ddot{x}(0)}{3!} \epsilon^3 + \dots}{\epsilon} = 0.$$

Thus, when ϵ is close to 0, the power series is well-approximated by its first two terms.

Substitution of formula (2) for $\dot{x}(0)$ into the approximate equation $x(\epsilon) = x + \epsilon \dot{x}(0)$ gives the approximate equation

$$x(\epsilon) = x + \epsilon A^{-1}(f - Fx);$$

rearranging and taking norms yields

$$\begin{aligned} \|x(\epsilon) - x\| &= \|\epsilon A^{-1}(f - Fx)\| \\ &\leq \epsilon \|A^{-1}\|_2 \|f - Fx\|. \end{aligned}$$

Division by $\|x\|$ and rearrangement gives

$$\begin{aligned} \frac{\|x(\epsilon) - x\|}{\|x\|} &\leq \epsilon \|A^{-1}\|_2 \frac{\|f - Fx\|}{\|x\|} \\ &\leq \epsilon \|A^{-1}\|_2 \frac{\|f\| + \|F\|_2 \|x\|}{\|x\|} \\ &= \|A^{-1}\|_2 \left(\epsilon \frac{\|f\|}{\|x\|} + \epsilon \|F\|_2 \right) \\ &= \|A^{-1}\|_2 \|A\|_2 \left(\epsilon \frac{\|f\|}{\|A\|_2 \|x\|} + \epsilon \frac{\|F\|_2}{\|A\|_2} \right) \\ &= \text{cond}(A) \left(\frac{\|\epsilon f\|}{\|A\|_2 \|x\|} + \frac{\|\epsilon F\|_2}{\|A\|_2} \right) \\ &\leq \text{cond}(A) \left(\frac{\|\epsilon f\|}{\|b\|} + \frac{\|\epsilon F\|_2}{\|A\|_2} \right). \end{aligned}$$

Recalling that ϵf and ϵF denote changes in b and A , respectively, this final form shows that the condition number of A does indeed measure the sensitivity of solutions to changes in the initial data A and b .

Discrete Orthogonal Functions

Introduction

Next, the concept of *discrete orthogonal functions*, and MATLAB implementation of these ideas, is introduced as a way of dealing with sensitive systems that may arise from linear least-squares approximation. The reader is referred to Appendix 2 for a review of the linear algebra concepts (e.g., inner products, orthogonal vectors, Gram-Schmidt process) that appear in this section.

replacing the modeling functions

If MATLAB balks in a solution attempt of the the linear least-squares problem

$$\min_{\mathbf{b} \in \mathbf{R}^n} \|\mathbf{y} - \mathbf{X}\mathbf{b}\|^2, \quad (3)$$

because the condition number of $\mathbf{X}'\mathbf{X}$ is too large, then what can be done? One solution is to transform the set of modeling functions f_1, f_2, \dots, f_m into a replacement set F_1, F_2, \dots, F_m , that has the following properties:

- Any function representable as a linear combination of the f_i is also representable as a linear combination of the F_i (and vice versa): that is,

$$\{c_1 f_1 + \dots + c_m f_m \mid c_i \in \mathbf{R}\} = \{c_1 F_1 + \dots + c_m F_m \mid c_i \in \mathbf{R}\}.$$

- The matrix $\bar{\mathbf{X}}'\bar{\mathbf{X}}$ that results from using the functions F_i is much better suited to numerical computations than the matrix $\mathbf{X}'\mathbf{X}$ that uses the original f_i .

The new functions F_i will be linear combinations of the original functions f_i , and consequently will usually be considerably more complicated than the original f_i .

NOTATION \mathbf{A}_{ij}	For a matrix \mathbf{A} , the notation \mathbf{A}_{ij} is used to denote the entry in row i and column j of \mathbf{A} .
--------------------------------------	--

analysis of $\mathbf{X}'\mathbf{X}$

Recall that the matrix \mathbf{X} in the linear least-squares problem (3) is constructed by setting

$$\mathbf{X}_{ij} = f_j(t_i),$$

where f_i ($i = 1, \dots, m$) are the modeling functions, and t_i ($i = 1, \dots, N$) are the time values of the data points (see Section 2.2). Therefore, $X_{ij}^t = X_{ji} = f_i(t_j)$. The ij^{th} entry of $X^t X$ is found by taking the inner product of the i^{th} row of X^t with the j^{th} column of X , yielding

$$\begin{aligned} (X^t X)_{ij} &= \sum_{k=1}^N X_{ik}^t X_{kj} \\ &= \sum_{k=1}^N f_i(t_k) f_j(t_k) . \end{aligned} \quad (4)$$

If the matrix $X^t X$ is diagonal, then the system $X^t X b = X^t y$ is trivial to solve for b . From (4), it is clear that $X^t X$ is diagonal if and only if

$$\sum_{k=1}^N f_i(t_k) f_j(t_k) = 0 \quad \text{for all } i \neq j, \quad i, j = 1, \dots, m .$$

Functions f_i and time values t_i that satisfy such a condition are therefore particularly desirable. This idea is captured in the following definition:

DEFINITION Let f_1, \dots, f_m be real-valued functions of one real variable, and let t_1, \dots, t_N be N distinct real numbers.
discrete orthogonality The functions f_1, \dots, f_m are *mutually orthogonal with respect to the time values* t_1, \dots, t_N if

$$\sum_{k=1}^N f_i(t_k) f_j(t_k) = 0 \quad \text{for all } i \neq j, \quad i, j = 1, \dots, m .$$

Functions $\{f_i\}_{i=1}^m$ satisfying this property are informally referred to as *discrete orthogonal functions*.

NOTATION For vectors $\mathbf{u} := (u_1, \dots, u_N)$ and $\mathbf{v} := (v_1, \dots, v_N)$ in \mathbb{R}^N , the notation $\langle \mathbf{u}, \mathbf{v} \rangle$ denotes the Euclidean inner product of \mathbf{u} and \mathbf{v} :

$$\langle \mathbf{u}, \mathbf{v} \rangle = \sum_{k=1}^N u_k v_k .$$

Vectors \mathbf{u} and \mathbf{v} are *orthogonal* if and only if $\langle \mathbf{u}, \mathbf{v} \rangle = 0$.

NOTATION Let $\mathbf{T} = (t_1, \dots, t_N)$ be an N -tuple of distinct time values, and let $f(\mathbf{T})$,
 f be a real-valued function of one real variable. The notation $f(\mathbf{T})$ denotes the N -tuple

$$f(\mathbf{T}) := (f(t_1), \dots, f(t_N)) .$$

If \mathbf{T} is understood from context, then $f(\mathbf{T})$ is more simply written as f .

In particular, a function f_i has associated N -tuple

$$\mathbf{f}_i := (f_i(t_1), \dots, f_i(t_N)) .$$

With the notation above, functions f_1, \dots, f_m are mutually orthogonal with respect to time values $\mathbf{T} = (t_1, \dots, t_N)$ if and only if \mathbf{f}_i is orthogonal to \mathbf{f}_j , whenever $i \neq j$.

In a typical linear least-squares application, the number of approximating functions (m) is much less than the number of time values (N). In this case, the m vectors $\mathbf{f}_1, \dots, \mathbf{f}_m$ span a finite-dimensional subspace of \mathbb{R}^N .

If the vectors $\mathbf{f}_1, \dots, \mathbf{f}_m$ are linearly independent, then they span an m -dimensional subspace of \mathbb{R}^N . In this case, the Gram-Schmidt orthogonalization procedure can be used to replace the original vectors \mathbf{f}_i by new vectors \mathbf{F}_i that are mutually orthogonal in \mathbb{R}^N , and that span precisely the same subspace of \mathbb{R}^N . Correspondingly, the original functions f_i are replaced by new functions F_i that are mutually orthogonal with respect to \mathbf{T} . This procedure is illustrated in the next example.

EXAMPLE

Let

$$f_1(t) = 1, \quad f_2(t) = t, \quad f_3(t) = t^2 ,$$

and let

$$\mathbf{T} = (1, 2, 3) .$$

Then,

$$\mathbf{f}_1 = (1, 1, 1) , \quad \mathbf{f}_2 = (1, 2, 3) , \quad \text{and} \quad \mathbf{f}_3 = (1, 4, 9) .$$

The Gram-Schmidt process yields:

$$\begin{aligned} \mathbf{F}_1 &= \frac{\mathbf{f}_1}{\|\mathbf{f}_1\|} = \frac{1}{\sqrt{3}}(1, 1, 1), \\ \mathbf{F}_2 &= \frac{\mathbf{f}_2 - \langle \mathbf{f}_2, \mathbf{F}_1 \rangle \mathbf{F}_1}{\|\mathbf{f}_2 - \langle \mathbf{f}_2, \mathbf{F}_1 \rangle \mathbf{F}_1\|} = \dots = \frac{1}{\sqrt{2}}(-1, 0, 1), \text{ and} \\ \mathbf{F}_3 &= \frac{\mathbf{f}_3 - \langle \mathbf{f}_3, \mathbf{F}_1 \rangle \mathbf{F}_1 - \langle \mathbf{f}_3, \mathbf{F}_2 \rangle \mathbf{F}_2}{\|\mathbf{f}_3 - \langle \mathbf{f}_3, \mathbf{F}_1 \rangle \mathbf{F}_1 - \langle \mathbf{f}_3, \mathbf{F}_2 \rangle \mathbf{F}_2\|} = \dots = \frac{1}{\sqrt{6}}(1, -2, 1). \end{aligned}$$

It is routine to check that the \mathbf{F}_i are mutually orthogonal in \mathbb{R}^3 .

the corresponding
functions F_i

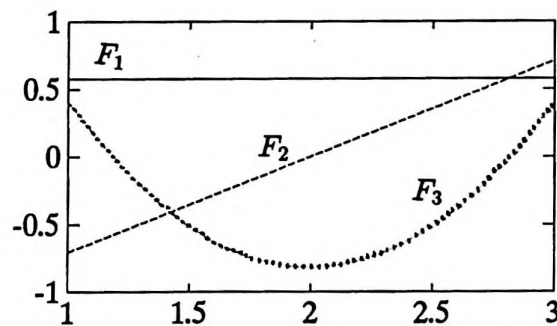
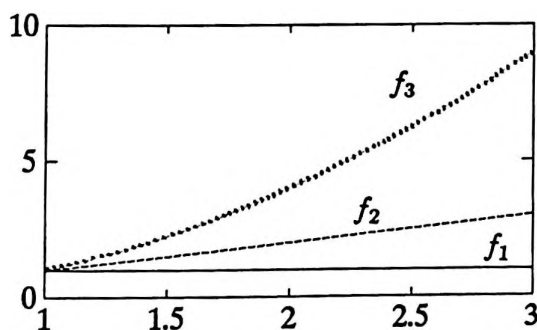
By writing the \mathbf{F}_i as linear combinations of the \mathbf{f}_i , one obtains the corresponding continuous functions F_i , which are mutually orthogonal with respect to \mathbf{T} :

$$\mathbf{F}_1 = \frac{\mathbf{f}_1}{\|\mathbf{f}_1\|} \Rightarrow F_1(t) = \frac{1}{\sqrt{3}}f_1(t) = \frac{1}{\sqrt{3}};$$

$$\begin{aligned} \mathbf{F}_2 &= \frac{\mathbf{f}_2 - \langle \mathbf{f}_2, \mathbf{F}_1 \rangle \mathbf{F}_1}{\|\mathbf{f}_2 - \langle \mathbf{f}_2, \mathbf{F}_1 \rangle \mathbf{F}_1\|} = \frac{1}{\sqrt{2}}(\mathbf{f}_2 - \langle \mathbf{f}_2, \mathbf{f}_1 \rangle \frac{1}{\|\mathbf{f}_1\|^2} \mathbf{f}_1) = \frac{1}{\sqrt{2}}(\mathbf{f}_2 - 2\mathbf{f}_1) \\ \Rightarrow F_2(t) &= \frac{1}{\sqrt{2}}(t - 2); \end{aligned}$$

$$\begin{aligned} \mathbf{F}_3 &= \frac{\mathbf{f}_3 - \langle \mathbf{f}_3, \mathbf{F}_1 \rangle \mathbf{F}_1 - \langle \mathbf{f}_3, \mathbf{F}_2 \rangle \mathbf{F}_2}{\|\mathbf{f}_3 - \langle \mathbf{f}_3, \mathbf{F}_1 \rangle \mathbf{F}_1 - \langle \mathbf{f}_3, \mathbf{F}_2 \rangle \mathbf{F}_2\|} = \dots = \frac{1}{\sqrt{6}}(3\mathbf{f}_3 - 12\mathbf{f}_2 + 10\mathbf{f}_1) \\ \Rightarrow F_3(t) &= \frac{1}{\sqrt{6}}(3t^2 - 12t + 10). \end{aligned}$$

The functions f_i and F_i are graphed below.



Next, MATLAB implementation is provided to transform a set of functions $\{f_i\}_{i=1}^m$ into a set $\{F_i\}_{i=1}^m$ of mutually orthogonal functions with respect to time values $\mathbf{T} := (t_1, \dots, t_n)$, providing that the vectors \mathbf{f}_i obtained from f_i and \mathbf{T} are linearly independent.

The algorithm used in the MATLAB function exploits the relationship between the functions f_i and the corresponding functions F_i . To illustrate this relationship, first use the Gram-Schmidt procedure on the vectors \mathbf{f}_i to obtain the corresponding mutually orthogonal vectors \mathbf{F}_i :

$$\begin{aligned}\mathbf{F}_1 &= \frac{\mathbf{f}_1}{\|\mathbf{f}_1\|}, \\ \mathbf{F}_2 &= \frac{\mathbf{f}_2 - \langle \mathbf{f}_2, \mathbf{F}_1 \rangle \mathbf{F}_1}{\|\mathbf{f}_2 - \langle \mathbf{f}_2, \mathbf{F}_1 \rangle \mathbf{F}_1\|}, \\ \mathbf{F}_3 &= \frac{\mathbf{f}_3 - \langle \mathbf{f}_3, \mathbf{F}_1 \rangle \mathbf{F}_1 - \langle \mathbf{f}_3, \mathbf{F}_2 \rangle \mathbf{F}_2}{\|\mathbf{f}_3 - \langle \mathbf{f}_3, \mathbf{F}_1 \rangle \mathbf{F}_1 - \langle \mathbf{f}_3, \mathbf{F}_2 \rangle \mathbf{F}_2\|}, \\ &\vdots\end{aligned}$$

Note that each \mathbf{F}_i is of the form $\frac{\mathbf{v}_i}{\|\mathbf{v}_i\|}$; define $K_i := \frac{1}{\|\mathbf{v}_i\|}$. Then, write each \mathbf{F}_i as a linear combination of the \mathbf{f}_i :

$$\begin{aligned}\mathbf{F}_1 &= K_1 \mathbf{f}_1, \\ \mathbf{F}_2 &= K_2 (\mathbf{f}_2 - \langle \mathbf{f}_2, \mathbf{F}_1 \rangle K_1 \mathbf{f}_1) \\ &= K_2 \mathbf{f}_2 - K_2 K_1 \langle \mathbf{f}_2, \mathbf{F}_1 \rangle \mathbf{f}_1, \\ &\vdots\end{aligned}$$

In this way, each vector \mathbf{F}_i is a linear combination of the original vectors $\{\mathbf{f}_i\}_{i=1}^m$, which determines the relationship between the functions f_i and F_i . The coefficients relating f_i and F_i are recorded in a matrix \mathbf{M} : the rows are labeled as F_i and the columns as f_i , so that

$$F_i = \mathbf{M}_{i1} f_1 + \mathbf{M}_{i2} f_2 + \dots + \mathbf{M}_{im} f_m.$$

The first few entries of \mathbf{M} are shown in the following diagram:

	f_1	f_2	f_3	f_4	\dots
F_1	K_1	0	0	0	\dots
F_2	$-K_2 M_{11}(f_2, F_1)$	K_2	0	0	\dots
F_3	$-K_3 M_{11}(f_3, F_1)$ $-K_3 M_{21}(f_3, F_2)$	$-K_3 M_{22}(f_3, F_2)$	K_3	0	\dots
F_4	$-K_4 M_{11}(f_4, F_1)$ $-K_4 M_{21}(f_4, F_2)$ $-K_4 M_{31}(f_4, F_3)$	$-K_4 M_{22}(f_4, F_2)$ $-K_4 M_{32}(f_4, F_3)$	$-K_4 M_{33}(f_4, F_3)$	K_4	\dots
\vdots					

The MATLAB algorithm exploits the following observations regarding the structure of M :

- The numbers K_i form the main diagonal.
- The element M_{i1} is easily determined from $M_{(i-1),1}$, for $i \geq 3$.
- Once element M_{i1} is known, all the entries on the sub-diagonal beginning with M_{i1} are easily determined.

MATLAB IMPLEMENTATION

Converting Functions to Discrete Orthogonal Functions

**PURPOSE
and
NOTATION**

Let f_1, \dots, f_m be m real-valued functions of a real variable. Let $\mathbf{T} = (t_1, \dots, t_N)$ be a vector of N distinct real numbers, where $m \leq N$.

Suppose that the vectors \mathbf{f}_i , defined by

$$\mathbf{f}_i = (f_i(t_1), \dots, f_i(t_N)) ,$$

are linearly independent in \mathbb{R}^N .

The following MATLAB function produces a matrix M that contains the coefficients relating the functions f_i to functions F_i that are mutually orthogonal with respect to the times values in \mathbf{T} . Precisely, for every $i = 1, \dots, m$,

$$F_i = M_{i1}f_1 + \dots + M_{im}f_m .$$

Optionally, the function will return a matrix \mathbf{F} that contains the mutually orthogonal vectors \mathbf{F}_i , with \mathbf{F}_i in column i :

$$\mathbf{F} = \begin{bmatrix} \mathbf{F}_1 & \mathbf{F}_2 & \cdots & \mathbf{F}_m \end{bmatrix}.$$

using \mathbf{F}
and \mathbf{M}
to solve a
sensitive
least-squares
problem

The matrices \mathbf{M} and \mathbf{F} can then be used to obtain an improved solution to a sensitive least-squares problem

$$\min_{\mathbf{b} \in \mathbb{R}^m} \|\mathbf{y} - \mathbf{X}\mathbf{b}\|^2,$$

where $\mathbf{X}_{ij} = f_j(t_i)$, and where $\mathbf{X}'\mathbf{X}$ has a large condition number, as follows:

- Replace \mathbf{X} by \mathbf{F} , and use the MATLAB implementation from Section 2.2 to find a vector $\mathbf{bnew} = (b_1, \dots, b_m)$ so that

$$f(t) = b_1 F_1(t) + \cdots + b_m F_m(t)$$

gives the least-squares approximate to the data (t_i, y_i) .

- Then, use the relationship between the functions f_i and F_i (contained in \mathbf{M}) to express $f(t)$ as a linear combination of the original functions f_i :

$$\begin{aligned} f(t) &= b_1 F_1(t) + \cdots + b_m F_m(t) \\ &= b_1 (\mathbf{M}_{11} f_1 + \mathbf{M}_{12} f_2 + \cdots + \mathbf{M}_{1m} f_m) \\ &\quad + b_2 (\mathbf{M}_{21} f_1 + \mathbf{M}_{22} f_2 + \cdots + \mathbf{M}_{2m} f_m) \\ &\quad + \cdots \\ &\quad + b_m (\mathbf{M}_{m1} f_1 + \mathbf{M}_{m2} f_2 + \cdots + \mathbf{M}_{mm} f_m) \\ &= f_1 (b_1 \mathbf{M}_{11} + b_2 \mathbf{M}_{21} + \cdots + b_m \mathbf{M}_{m1}) \\ &\quad + f_2 (b_1 \mathbf{M}_{12} + b_2 \mathbf{M}_{22} + \cdots + b_m \mathbf{M}_{m2}) \\ &\quad + \cdots \\ &\quad + f_m (b_1 \mathbf{M}_{1m} + b_2 \mathbf{M}_{2m} + \cdots + b_m \mathbf{M}_{mm}) \\ &:= \tilde{b}_1 f_1(t) + \cdots + \tilde{b}_m f_m(t). \end{aligned}$$

The coefficients $\tilde{\mathbf{b}} := (\tilde{b}_1, \dots, \tilde{b}_m)$ are obtained by:

$$\tilde{\mathbf{b}} = \mathbf{M}'(\mathbf{bnew}).$$

```

function [M,F] = dscorth(T,f,m)
% T is the input column vector of times
% f is the N x m matrix where column i is the vector fi
% m is the number of functions
% initialize the matrix to hold the orthogonal vectors Fi
F = zeros(length(T),m);
% initialize the matrix to hold the coefficients relating fi to Fi
M = zeros(m,m);
% initialize the vector to hold the norms of the Fi
K = zeros(1,m);
% compute the K(i) and Fi
for i = 1:m
    j = 1;
    NUM = f(:,i);
    while j < i
        NUM = NUM - (f(:,i)'*F(:,j))*F(:,j);
        j = j+1;
    end
    K(i) = 1/norm(NUM);
    F(:,i) = NUM*K(i);
end
% Compute the coefficients relating fi and Fi
% First compute the elements on the main diagonal
for i = 1:m
    M(i,i) = K(i);
end
% Compute the remaining elements
k = 2;
while k <= m
    i = k;
    j = 1;
    while i <= m
        M(i,j) = (f(:,i)'*F(:,j:i-1))*(-K(i)*M(j:i-1,j));
        j = j+1;
        i = i+1;
    end
    k = k+1;
end

```

EXAMPLE

In the following diary of an actual MATLAB session, a data set is fitted with a polynomial of degree 14, in order to illustrate how large condition numbers can cause degraded results, and how the use of discrete orthogonal functions can improve the situation.

The 'traditional' least-squares solution results in a matrix $X'X$ with an extremely large condition number, yielding a 'best fit' that has error 443.6303.

By using the mutually orthogonal function approach, the error is decreased significantly, to 273.5300.

```

t = [0:.1:20]';
N = .5*(2*rand(t) - 1);
y = 3.2 - .7*t + .05*t.^2 + 2*sin(2*pi*t/3) + N;
f1 = ones(t);
f2 = t;
f3 = t.^2;
f4 = t.^3;
f5 = t.^4;
f6 = t.^5;
f7 = t.^6;
f8 = t.^7;
f9 = t.^8;
f10 = t.^9;
f11 = t.^(10);
f12 = t.^(11);
f13 = t.^(12);
f14 = t.^(13);
f15 = t.^(14);
X = [f1 f2 f3 f4 f5 f6 f7 f8 f9 f10 f11 f12 f13 f14 f15];
b = (X'*X) \ (X'*y)
Warning: Matrix is close to singular or badly scaled.
Results may be inaccurate. RCOND = 6.208041e-040

```

```

b =
    4.4985
   -7.4163
   22.4853
  -26.4539
   14.9247
   -4.7515
    0.9246
   -0.1128
    0.0083
   -0.0003
   -0.0000
    0.0000
   -0.0000
    0.0000
   -0.0000

[M,F] = dscorth(t,X,15);
bnew = (F'*F) \ (F'*y);
btilde = M'*bnew

btilde =

    2.7347
    6.1579
    3.9543
   -22.4910
   21.5315
   -10.2060
    2.9203
   -0.5486
    0.0706
   -0.0063
    0.0004
   -0.0000
    0.0000
   -0.0000
    0.0000

fold = X*b;
fnew = X*btilde;
error1 = (y - fold)'*(y - fold)

error1 =

    443.6303

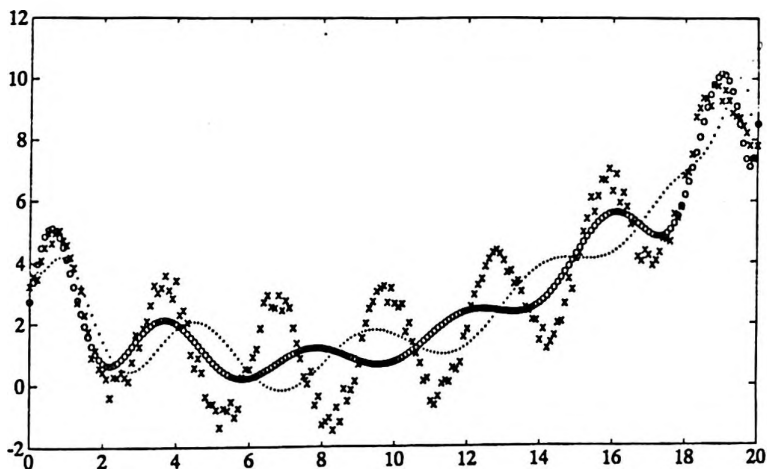
error2 = (y - fnew)'*(y - fnew)

error2 =

    273.5300

plot(t,y,'x',t,fold,'.',t,fnew,'o')

```



x x x x x data points

. 'traditional' least-squares solution

o o o o o improved fit, obtained from the discrete orthogonal function approach

2.4 Nonlinear Least Squares Approximation

A Linearizing Technique and Gradient Methods

Introduction

Let $\{(t_i, y_i)\}_{i=1}^N$ be a given data set. Consider the following minimization problem:

Find a sinusoid that best approximates the data, in the least-squares sense. That is, find real constants $A \neq 0$, ϕ , and $\omega \neq 0$ so that the function f given by

$$f(t) = A \sin(\omega t + \phi)$$

minimizes the sum of the squared errors between $f(t_i)$ and y_i :

$$\begin{aligned} SSE &:= \sum_{i=1}^N (y_i - f(t_i))^2 \\ &= \sum_{i=1}^N (y_i - A \sin(\omega t_i + \phi))^2. \end{aligned}$$

a nonlinear least-squares problem

This problem differs from the least-squares problem considered in Section 2.2, in that the parameters to be ‘adjusted’ (A , ϕ and ω) do not all appear linearly in the formula for the approximate f . (This notion of *linear dependence on a parameter* is made precise below.) Consequently, the resulting problem is called *nonlinear least-squares approximation*, and is exceedingly more difficult to solve, because it does not lend itself to exact solution by matrix methods. Indeed, nonlinear least-squares problems must generally be solved by iterative techniques. Nonlinear least-squares approximation is the subject of this section.

First, a definition:

DEFINITION *linear dependence on a parameter*

Suppose f is a function that depends on time, a fixed parameter b , and possibly other parameters. This dependence is denoted by $f = f(t, b, x)$, where x represents all other parameters.

The function f is said to be *linear with respect to the parameter b* if and only if the function f has the form

$$f(t, b, x) = b \cdot f_1(t, x) + f_2(t, x)$$

for functions f_1 and f_2 that may depend on t and x , but not on b .

*generalizing
the definition*

This definition is generalized in the obvious way. For example, a function $f = f(t, b_1, b_2, x)$ is linear with respect to the parameters b_1 and b_2 if and only if f has the form

$$f(t, b_1, b_2, x) = b_1 \cdot f_1(t, x) + b_2 \cdot f_2(t, x) + f_3(t, x)$$

for functions f_1, f_2 and f_3 that may depend on t and x , but not on b_1 and b_2 .

EXAMPLE

The function $f(t, A, \phi, \omega) = A \sin(\omega t + \phi)$ is linear with respect to the parameter A , but not linear with respect to ϕ or ω .

*when f depends
linearly on b ,
so does $\frac{\partial SSE}{\partial b}$*

If f depends linearly on a parameter b , then $\frac{\partial SSE}{\partial b}$ also depends linearly on b , as follows:

$$\begin{aligned} SSE &= \sum_{i=1}^N (y_i - b \cdot f_1(t_i, x) - f_2(t_i, x))^2 ; \\ \frac{\partial SSE}{\partial b} &= \sum_{i=1}^N 2(y_i - b \cdot f_1(t_i, x) - f_2(t_i, x)) \cdot (-f_1(t_i, x)) \\ &= b \cdot \sum_{i=1}^N 2(f_1(t_i, x))^2 - 2 \sum_{i=1}^N y_i f_1(t_i, x) - f_1(t_i, x) f_2(t_i, x) . \end{aligned}$$

Similarly, if f depends linearly on parameters b_1, \dots, b_m , then so do $\frac{\partial SSE}{\partial b_i}$, for all $i = 1, \dots, m$. Consequently, as seen in Section 2.2, matrix methods can be used to find parameters that make the partials equal to zero.

Unfortunately, when f does NOT depend linearly on a parameter b , the situation is considerably more difficult, as illustrated next:

EXAMPLE

Let $f(t) = A \sin(\omega t + \phi)$, so that, in particular, f does not depend linearly on ω . In this case, differentiation of

$$SSE = \sum_{i=1}^N (y_i - A \sin(\omega t_i + \phi))^2$$

with respect to ω yields

$$\frac{\partial SSE}{\partial \omega} = \sum_{i=1}^N 2(y_i - A \sin(\omega t_i + \phi)) (-A t_i \cos(\omega t_i + \phi)) .$$

Even if A and ϕ are held constant, the equation $\frac{\partial SSE}{\partial \omega} = 0$ is not easy to solve for ω .

*a linearizing
technique*

Since a linear dependence on parameters is desirable, it is reasonable to try and replace the class of approximating functions,

$$\mathcal{F} = \{f \mid f(t) = A \sin(\omega t + \phi), A \neq 0, \phi \in \mathbf{R}, \omega \neq 0\} \quad (1)$$

with another class $\tilde{\mathcal{F}}$, that satisfies the following properties:

- $\mathcal{F} = \tilde{\mathcal{F}}$, and
- The new class $\tilde{\mathcal{F}}$ involves functions that have more linear dependence on the approximating parameters than the original class.

If it is possible to find such a class $\tilde{\mathcal{F}}$, then the process of replacing \mathcal{F} by $\tilde{\mathcal{F}}$ is called a *linearizing technique*.

EXAMPLE

For example, the next lemma shows that the set

$$\tilde{\mathcal{F}} := \{f \mid f(t) = C \sin \omega t + D \cos \omega t, \omega \neq 0, C \text{ and } D \text{ not both zero}\}$$

is equal to the set \mathcal{F} given in (1). Thus, any function representable in the form $A \sin(\omega t + \phi)$ is also representable in the form $C \sin \omega t + D \cos \omega t$, and vice versa.

Both classes \mathcal{F} and $\tilde{\mathcal{F}}$ involve three parameters: \mathcal{F} involves A , ϕ and ω ; and $\tilde{\mathcal{F}}$ involves C , D and ω . However, a function $f(t) = C \sin \omega t + D \cos \omega t$ from $\tilde{\mathcal{F}}$ depends linearly on *both* C and D , whereas $f(t) = A \sin(\omega t + \phi)$ only depends linearly on A . Thus, the class $\tilde{\mathcal{F}}$ is more desirable.

LEMMA

*a linearizing
transformation*

Let \mathcal{F} and $\tilde{\mathcal{F}}$ be classes of functions of one variable, defined by

$$\mathcal{F} = \{f \mid f(t) = A \sin(\omega t + \phi), A \neq 0, \phi \in \mathbf{R}, \omega \neq 0\}, \text{ and}$$

$$\tilde{\mathcal{F}} = \{f \mid f(t) = C \sin \omega t + D \cos \omega t, \omega \neq 0, C \text{ and } D \text{ not both zero}\}.$$

Then, $\mathcal{F} = \tilde{\mathcal{F}}$.

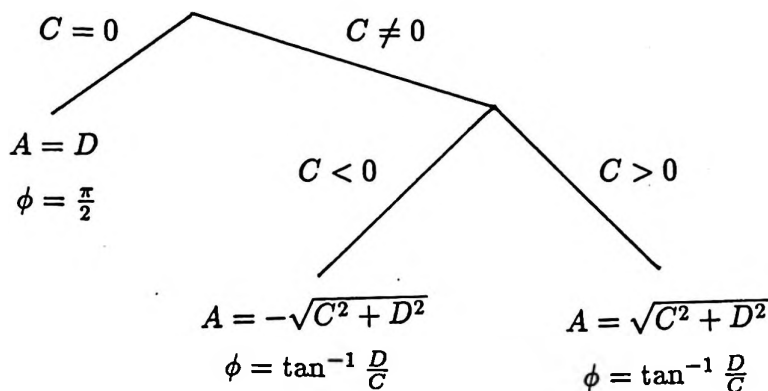
PROOF

Let $f \in \mathcal{F}$. For all t , A , ϕ and ω ,

$$\begin{aligned} f(t) &= A \sin(\omega t + \phi) \\ &= A(\sin \omega t \cos \phi + \cos \omega t \sin \phi) \\ &= \underbrace{(A \cos \phi)}_C \sin \omega t + \underbrace{(A \sin \phi)}_D \cos \omega t. \end{aligned} \quad (*)$$

Define $C := A \cos \phi$ and $D := A \sin \phi$. Since $f \in \mathcal{F}$, one has $A \neq 0$. Therefore, the only way that both C and D can equal zero is if $\cos \phi = \sin \phi = 0$, which is impossible. Thus, $f \in \tilde{\mathcal{F}}$.

Now, let $f \in \tilde{\mathcal{F}}$, so that $f(t) = C \sin \omega t + D \cos \omega t$ for $\omega \neq 0$, and for numbers C and D that are not both zero. It may be useful to refer to the flow chart below while reading the remainder of the proof.



If $C = 0$, then $D \neq 0$. In this case,

$$C \sin \omega t + D \cos \omega t = D \cos \omega t = D \sin\left(\omega t + \frac{\pi}{2}\right).$$

Take $A := D$, and $\phi = \frac{\pi}{2}$ to see that $f \in \mathcal{F}$.

Now, let $C \neq 0$. If there exist constants $A \neq 0$ and ϕ for which $C = A \cos \phi$ and $D = A \sin \phi$, then A and ϕ must satisfy certain requirements, as follows:

Since $C \neq 0$, it follows that $\cos \phi \neq 0$. Thus,

$$\frac{D}{C} = \frac{A \sin \phi}{A \cos \phi} = \tan \phi.$$

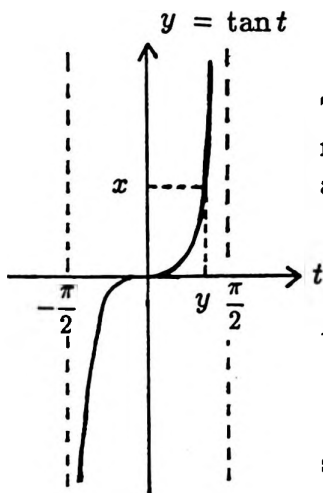
There are an infinite number of choices for ϕ that satisfy this requirement. Choose $\phi := \tan^{-1} \frac{D}{C}$, where \tan^{-1} denotes the arctangent function, defined by

$$y = \tan^{-1} x \iff (\tan y = x \text{ and } y \in (-\frac{\pi}{2}, \frac{\pi}{2})).$$

Also note that

$$C^2 + D^2 = (A \cos \phi)^2 + (A \sin \phi)^2 = A^2,$$

so that A must equal either $\sqrt{C^2 + D^2}$ or $-\sqrt{C^2 + D^2}$.



The table below gives the correct formulas for

$$\cos(\tan^{-1} \frac{D}{C}) \quad \text{and} \quad \sin(\tan^{-1} \frac{D}{C}) ,$$

for all possible sign choices of C and D . In this table,

$$K := \sqrt{C^2 + D^2} .$$

formulas for
 $\sin(\tan^{-1} \frac{D}{C})$
 and
 $\cos(\tan^{-1} \frac{D}{C})$

D	C	$\sin(\tan^{-1} \frac{D}{C})$	$\cos(\tan^{-1} \frac{D}{C})$
+	+	$\frac{D}{K}$	$\frac{C}{K}$
+	-	$-\frac{D}{K}$	$-\frac{C}{K}$
-	+	$\frac{D}{K}$	$\frac{C}{K}$
-	-	$-\frac{D}{K}$	$-\frac{C}{K}$

If $C > 0$, then take $A := \sqrt{C^2 + D^2}$. Using (*),

$$\begin{aligned} A \sin(\omega t + \phi) &= \sqrt{C^2 + D^2} \cos(\tan^{-1} \frac{D}{C}) \sin \omega t \\ &\quad + \sqrt{C^2 + D^2} \sin(\tan^{-1} \frac{D}{C}) \cos \omega t \\ &= \sqrt{C^2 + D^2} \frac{C}{\sqrt{C^2 + D^2}} \sin \omega t \\ &\quad + \sqrt{C^2 + D^2} \frac{D}{\sqrt{C^2 + D^2}} \cos \omega t \\ &= C \sin \omega t + D \cos \omega t . \end{aligned}$$

If $C < 0$, take $A := -\sqrt{C^2 + D^2}$. Then,

$$\begin{aligned} A \sin(\omega t + \phi) &= -\sqrt{C^2 + D^2} \cos(\tan^{-1} \frac{D}{C}) \sin \omega t \\ &\quad - \sqrt{C^2 + D^2} \sin(\tan^{-1} \frac{D}{C}) \cos \omega t \\ &= -\sqrt{C^2 + D^2} \frac{-C}{\sqrt{C^2 + D^2}} \sin \omega t \\ &\quad - \sqrt{C^2 + D^2} \frac{-D}{\sqrt{C^2 + D^2}} \cos \omega t \\ &= C \sin \omega t + D \cos \omega t . \end{aligned}$$

In both cases, it has been shown that $f \in \mathcal{F}$. ■

general
approach

The following iterative approach can be used to find a choice of parameters C , D and ω so that the function $f(t) = C \sin \omega t + D \cos \omega t$ best approximates the data, in the least-squares sense. Observe that, once a value of ω is fixed, the remaining problem is just linear least-squares approximation, treated in Section 2.2.

- Estimate a value of ω , based perhaps on initial data analysis.
- Use the methods from Section 2.2 to find optimal parameters C and D corresponding to ω .
- Compute the sum of squared errors, $SSE(C, D, \omega)$, corresponding to C , D and ω .
- Try to decrease SSE by adjusting ω , and repeating the process.

One algorithm for 'adjusting' ω is based on the following result from multivariable calculus:

THEOREM
a differentiable
function h
changes
most rapidly
in the direction of
the gradient;
 ∇h is
the gradient of h

Let h be a function from \mathbb{R}^m into \mathbb{R} , and let $\mathbf{x} = (x_1, \dots, x_m)$ be an element of \mathbb{R}^m . If h has continuous first partials in a neighborhood of \mathbf{x} , then the function $\nabla h : \mathbb{R}^m \rightarrow \mathbb{R}^m$ defined by

$$\nabla h(x_1, \dots, x_m) := \left(\frac{\partial h}{\partial x_1}(\mathbf{x}), \dots, \frac{\partial h}{\partial x_m}(\mathbf{x}) \right)$$

has the following property:

at \mathbf{x} , the function h increases most rapidly in the direction of the vector $\nabla h(\mathbf{x})$ (the rate of change is then $\|\nabla h(\mathbf{x})\|$), and decreases most rapidly in the direction of $-\nabla h(\mathbf{x})$ (the rate of change is then $-\|\nabla h(\mathbf{x})\|$).

The function ∇h is called the *gradient* of h .

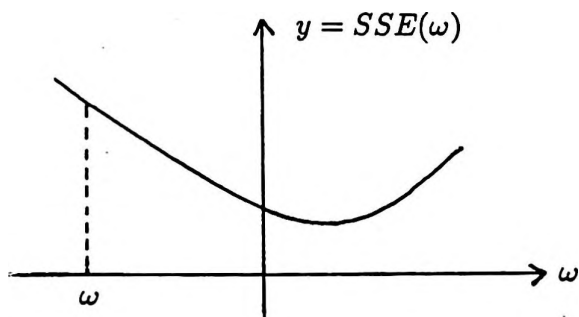
This theorem will be used to 'adjust' parameter(s) so that the function SSE will decrease most rapidly.

- Let ω be a current parameter value.
- Compute $\nabla SSE(\omega)$. One must move in the direction of $-\nabla SSE(\omega)$ to decrease SSE most rapidly. Thus, let ϵ be a small positive number, and compute

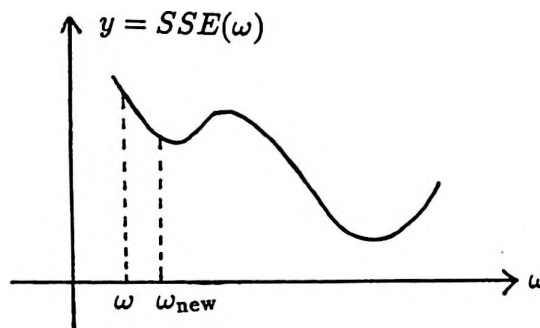
$$\omega_{\text{new}} = \omega - \epsilon \cdot \frac{\nabla SSE(\omega)}{\|\nabla SSE(\omega)\|}.$$

When $\|\nabla SSE(\omega)\|$ is large, the function SSE is changing rapidly at ω ; when $\|\nabla SSE(\omega)\|$ is small, the function SSE is changing slowly at ω .

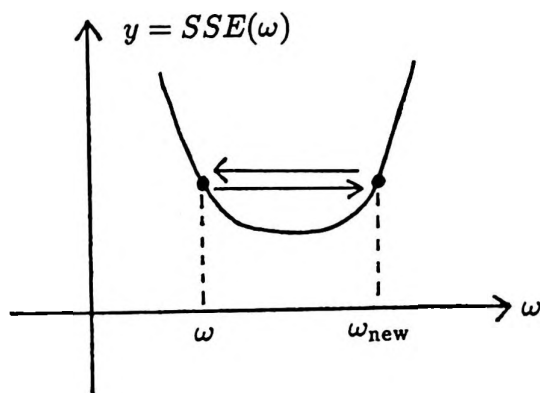
The MATLAB implementation given in this section allows the user to choose the value of ϵ . 'Optimal' values of ϵ will depend on both the function SSE , and the current value of ω , as suggested by the sketches below (where, for convenience, SSE is a function of one variable only). These sketches also illustrate how 'bouncing' can occur, and how ω can converge to a non-optimal minimum. In general, experience and experimentation by the analyst will play a role in the success of this technique.



Far away from minimum;
large ϵ will speed
process initially



convergence to
'wrong' minimum



'bouncing'

*a general
nonlinear
least-squares
problem*

The previous theorem is now applied to a general nonlinear least-squares problem. MATLAB implementation, for a special case of the situation discussed here, will follow.

Let f_1, f_2, \dots, f_m be functions that depend on time, and on parameters $\mathbf{c} := (c_1, \dots, c_p)$. That is, $f_i = f_i(t, \mathbf{c})$ for $i = 1, \dots, m$. The dependence on any of the inputs may be nonlinear.

Define a function f that depends on time, the parameters \mathbf{c} , and parameters $\mathbf{b} = (b_1, \dots, b_m)$, via

$$f(t, \mathbf{c}, \mathbf{b}) = b_1 f_1(t, \mathbf{c}) + \dots + b_m f_m(t, \mathbf{c}) .$$

Thus, f is linear with respect to the parameters in \mathbf{b} , but not in general with respect to the parameters in \mathbf{c} .

Given a data set $\{(t_i, y_i)\}_{i=1}^N$, let $\mathbf{t} = (t_1, \dots, t_N)$ and $\mathbf{y} = (y_1, \dots, y_N)$. The sum of the squared errors between $f(t_i, \mathbf{c}, \mathbf{b})$ and y_i is

$$SSE = \sum_{i=1}^N (y_i - f(t_i, \mathbf{c}, \mathbf{b}))^2 ,$$

which has partials

$$\frac{\partial SSE}{\partial c_j} = \sum_{i=1}^N 2((y_i - f(t_i, \mathbf{c}, \mathbf{b}))(-\frac{\partial f}{\partial c_j}(t_i, \mathbf{c}, \mathbf{b})) ,$$

for $j = 1, \dots, p$. Note that

$$\frac{\partial f}{\partial c_j}(t_i, \mathbf{c}, \mathbf{b}) = b_1 \frac{\partial f_1}{\partial c_j}(t_i, \mathbf{c}) + \dots + b_m \frac{\partial f_m}{\partial c_j}(t_i, \mathbf{c}) .$$

Then, $\nabla SSE = (\frac{\partial SSE}{\partial c_1}, \dots, \frac{\partial SSE}{\partial c_p})$.

the algorithm

The algorithm:

- Choose an initial value of (c_1, \dots, c_p) .
- Use the methods in Section 2.2 to find corresponding optimal values of (b_1, \dots, b_m) .
- Adjust:

$$(c_1, \dots, c_p)_{\text{new}} = (c_1, \dots, c_p) - \epsilon \cdot \frac{\nabla SSE}{\|\nabla SSE\|} .$$

The analyst may need to adjust ϵ based on an analysis of the values of SSE . If SSE is large, but decreasing slowly, make ϵ larger. If SSE is 'bouncing', make ϵ smaller.

- Repeat with the new values in \mathbf{c} .

MATLAB IMPLEMENTATION

Nonlinear Least-Squares, Gradient Method

using
the function
nonlin

The following MATLAB function uses the gradient method discussed in this section to fit data with a sum of the form

$$f(t) = A + Bt + \sum_{i=1}^m C_i \sin \omega_i t + D_i \cos \omega_i t .$$

The fundamental period of $\sin \omega_i t$ is $P_i := \frac{2\pi}{\omega_i}$. To use the function, type

```
[E,f] = nonlin(P,D,epsil,tol);
```

required
inputs

The required inputs are:

- $P = [P_1 \cdots P_m]$ is a matrix containing the initial estimates for the unknown periods. P may be a row or a column matrix.
- D is an $N \times 2$ matrix containing the data set $\{(t_i, y_i)\}_{i=1}^N$. The time values are in the first column; the corresponding data values are in the second column.
- `epsil` is used to control the search in parameter space. Indeed,

$$(P_1, \dots, P_m)_{\text{new}} = (P_1, \dots, P_m) - \epsilon \frac{\nabla SSE}{\|\nabla SSE\|} .$$

- `tol` is used to help determine when the algorithm stops. The algorithm will stop when either 25 iterations have been made, or when $SSE < \text{tol}$.

outputs from
the function

The function returns the following information:

- E is a matrix containing each iteration of 'best' parameters and the associated error. Each row of E is of the form:

$$A \quad B \quad P_1 \quad C_1 \quad D_1 \quad P_2 \quad C_2 \quad D_2 \quad \cdots \quad P_m \quad C_m \quad D_m \quad SSE$$

- f contains the values of the approximation to the data, using the parameter values in the last row of the matrix E , and using the time values in t . Then, the command `plot(t,y,'x',t,f,'.')` can be used to compare the actual data set with the approximate.

The MATLAB function `nonlin` is given on the next page.

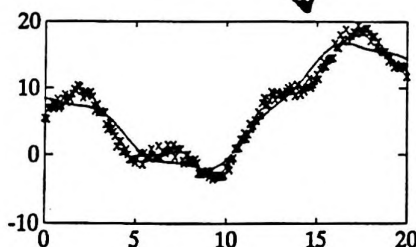
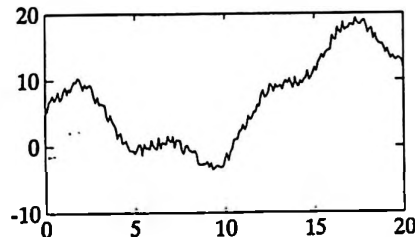
% copyright 1993 Carol J.V. Fisher

function [E,f] = nonlin(P,D,epsil,tol)

```
% P=[P1 P2 ... PM] is an Mx1 matrix containing the initial
% period estimates, where  $w = (2\pi)/P$ 
% D (for 'data') is a two-column matrix, size N x 2;
% the first column contains the time values ti
% the second column contains the data values yi
% epsil is used to determine the movement in
% in parameter space:  $w_{new} = w_{old} + \text{epsil} * (\text{gradient} / |\text{gradient}|)$ 
% The algorithm stops if SSE < tol, or if 25 iterations have been completed
% (adjust this number where indicated below, if desired).
M = length(P);
t = D(:,1);
N = length(t);
y = D(:,2);
w = 2*pi ./ P;
% initialize matrices
% E=[A B P1 C1 D1 P2 C2 D2...SSE] holds the parameters and associated error
E = zeros(2,3*M+3);
% gradient will hold (partial of SSE with respect to w)
gradient = zeros(1,M);
% ngradient will hold (gradient)/(|gradient|), which has unit length
% approximate is of form:
%  $f(t) = A + Bt + (C1)\sin(w1)t + (D1)\cos(w1)t + \dots + (CM)\sin(wM)t + (DM)\cos(wM)t$ 
X = zeros(N,2*M+2);
X(:,1) = ones(t);
X(:,2) = t;
% initially set SSE greater than tol so it loops through at least once
SSE = tol + 1;
i = 0;
% Change the number of iterations from 25 in the next line, if desired.
while ((SSE > tol) & (i < 25))
    i = i+1;
    % complete the X matrix with the appropriate w values
    k=1;
    for j = 1:2:2*M
        X(:,j+2) = sin(w(k)*t);
        X(:,j+3) = cos(w(k)*t);
        k=k+1;
    end
    b = (X'*X) \ (X'*y);
    f = X*b;
    SSE = (y - f)'*(y - f);
    % store the results in the matrix E
    c = 1;
    k = 1;
    E(i,1) = b(1);
    E(i,2) = b(2);
    for j = 1:3:3*M
        E(i,j+2) = P(c);
        E(i,j+3) = b(k+2);
        E(i,j+4) = b(k+3);
        c = c+1;
        k = k+2;
    end
    E(i,3*M+3) = SSE;
    % next, compute the gradient and move to a new w in parameter space
    for k = 1:M
        gradient(k) = sum(-2*(y-f).*(b(2*k+1)*t.*cos(w(k)*t) - b(2*k+2)*t.*sin(w(k)*t)));
    end
    ngradient = gradient ./ norm(gradient);
    w = w - epsil*ngradient;
    P = 2*pi ./ w;
    end
    E = E(1:i,:);
```

The following diary of an actual MATLAB session illustrates the use of the function `nonlin`.

```
t = [0:.1:20]';
y = 1+.5*t + sin((2*pi/5)*t) - 2*cos((2*pi/5)*t) + 7*cos((2*pi/17)*t);
noise = 2*(rand(t) - .5);
y = y+noise;
subplot(221)
plot(t,y)
P = [4 20];
D = [t y];
[E f] = nonlin(P,D,.01,1);
plot(t,y,'x',t,f,'.')
```



E

E =

A	B	P ₁	C ₁	D ₁	P ₂	C ₂	D ₂	SSE
.6735	0.2179	4.0000	-0.1875	-0.1483	20.0000	-4.9345	6.1258	692.6973
.9394	0.2727	3.9991	-0.1788	-0.1316	19.3834	-4.0371	6.5072	655.2642
.2665	0.3230	3.9983	-0.1696	-0.1135	18.8036	-3.1435	6.7961	620.6948
.6485	0.3691	3.9976	-0.1598	-0.0938	18.2574	-2.2567	6.9966	590.1082
.0804	0.4117	3.9971	-0.1492	-0.0726	17.7420	-1.3802	7.1114	564.8433
.5579	0.4509	3.9968	-0.1380	-0.0500	17.2548	-0.5180	7.1427	546.4859
.0778	0.4873	3.9969	-0.1262	-0.0258	16.7936	0.3248	7.0921	536.8874
.6390	0.5209	3.9988	-0.1154	0.0023	16.3576	1.1402	6.9618	538.1223
.0788	0.4872	4.0010	-0.1304	-0.0173	16.7932	0.3247	7.0927	536.8176
.6416	0.5206	4.0037	-0.1210	0.0118	16.3586	1.1374	6.9631	537.9642
.0808	0.4870	4.0065	-0.1367	-0.0057	16.7932	0.3235	7.0937	536.6824
.6466	0.5202	4.0106	-0.1295	0.0247	16.3614	1.1311	6.9656	537.6635
.0845	0.4866	4.0143	-0.1466	0.0104	16.7940	0.3204	7.0953	536.4148
.6561	0.5194	4.0201	-0.1429	0.0424	16.3678	1.1176	6.9700	537.0875
.0910	0.4860	4.0253	-0.1625	0.0326	16.7962	0.3141	7.0980	535.8669
.6737	0.5179	4.0335	-0.1645	0.0663	16.3808	1.0911	6.9781	535.9781
.1005	0.4852	4.0411	-0.1893	0.0633	16.7995	0.3051	7.1023	534.6910
.7048	0.5154	4.0525	-0.2005	0.0983	16.4052	1.0425	6.9921	533.8175
.1085	0.4844	4.0641	-0.2363	0.1050	16.7983	0.3037	7.1079	532.0192
.7583	0.5112	4.0801	-0.2638	0.1394	16.4483	0.9584	7.0153	529.4687
.0964	0.4853	4.0980	-0.3210	0.1565	16.7733	0.3465	7.1128	525.8067
.8531	0.5039	4.1203	-0.3773	0.1839	16.5272	0.8073	7.0543	520.3607
.0528	0.4887	4.1449	-0.4638	0.1994	16.7134	0.4569	7.1149	512.9167
.9972	0.4932	4.1722	-0.5552	0.2051	16.6501	0.5768	7.1082	503.3758
.0633	0.4884	4.1998	-0.6593	0.1956	16.7057	0.4751	7.1325	491.7921

% repeat, using the parameters from the last line

```
[E f] = nonlin([4.1998 16.7057],D,.01,1);
E
```


E =

1.0633	0.4884	4.1998	-0.6592	0.1956	16.7057	0.4750	7.1325	491.800
1.1066	0.4855	4.2280	-0.7670	0.1685	16.7401	0.4146	7.1509	478.170
1.1579	0.4821	4.2565	-0.8766	0.1226	16.7834	0.3391	7.1707	462.482
1.2139	0.4784	4.2853	-0.9844	0.0570	16.8329	0.2536	7.1907	444.774
1.2742	0.4745	4.3145	-1.0870	-0.0289	16.8882	0.1588	7.2103	425.145
1.3378	0.4704	4.3441	-1.1803	-0.1347	16.9491	0.0555	7.2288	403.749
1.4041	0.4662	4.3740	-1.2608	-0.2592	17.0150	-0.0554	7.2454	380.799
1.4722	0.4620	4.4043	-1.3247	-0.4009	17.0853	-0.1727	7.2596	356.561
1.5408	0.4577	4.4350	-1.3689	-0.5571	17.1593	-0.2951	7.2706	331.345
1.6090	0.4535	4.4661	-1.3904	-0.7248	17.2362	-0.4208	7.2779	305.498
1.6754	0.4494	4.4977	-1.3872	-0.9004	17.3148	-0.5479	7.2809	279.390
1.7385	0.4456	4.5296	-1.3576	-1.0800	17.3938	-0.6741	7.2794	253.409
1.7966	0.4421	4.5621	-1.3008	-1.2593	17.4717	-0.7965	7.2731	227.943
1.8478	0.4392	4.5951	-1.2167	-1.4339	17.5467	-0.9121	7.2621	203.376
1.8899	0.4368	4.6286	-1.1059	-1.5995	17.6163	-1.0169	7.2467	180.073
1.9202	0.4353	4.6627	-0.9699	-1.7517	17.6777	-1.1064	7.2273	158.370
1.9356	0.4347	4.6973	-0.8106	-1.8866	17.7275	-1.1751	7.2048	138.567
1.9321	0.4355	4.7327	-0.6309	-2.0002	17.7612	-1.2163	7.1801	120.910
1.9053	0.4378	4.7686	-0.4344	-2.0893	17.7735	-1.2217	7.1545	105.581
1.8498	0.4422	4.8050	-0.2253	-2.1510	17.7581	-1.1816	7.1291	92.687
1.7602	0.4488	4.8419	-0.0090	-2.1836	17.7083	-1.0854	7.1046	82.244
1.6331	0.4581	4.8788	0.2086	-2.1866	17.6190	-0.9247	7.0806	74.192
1.4712	0.4698	4.9157	0.4216	-2.1616	17.4907	-0.7003	7.0547	68.425
1.2887	0.4828	4.9524	0.6262	-2.1124	17.3362	-0.4329	7.0236	64.841
1.1127	0.4953	4.9896	0.8225	-2.0426	17.1824	-0.1681	6.9862	63.348

[E f] = nonlin([4.9896 17.1824],D,.01,1);

E

E =

1.1127	0.4953	4.9896	0.8226	-2.0426	17.1824	-0.1682	6.9862	63.348
0.8913	0.5112	5.0253	1.0004	-1.9545	16.9748	0.1898	6.9353	63.919
1.2988	0.4813	5.0047	0.8931	-2.0277	17.3774	-0.5068	6.9887	64.625
0.8716	0.5133	4.9910	0.8359	-2.0207	16.9376	0.2607	6.9496	64.493

:

:

% 'Bouncing' has occurred. Go back and decrease epsilon.
E(25,:)

ans =

Columns 1 through 7

1.3020	0.4810	5.0040	0.8898	-2.0293	17.3803	-0.5117
--------	--------	--------	--------	---------	---------	---------

Columns 8 through 9

6.9892	64.6252
--------	---------

[E f] = nonlin([5.004 17.3803],D,.001,1);

A Genetic Algorithm

What is a genetic algorithm?

A *genetic algorithm* is a search technique based on the mechanics of natural selection and genetics. Genetic algorithms may be used whenever there is a real-valued function f that is to be maximized over a finite set of parameters, S :

$$\max_{p \in S} f(p) .$$

In practice, an infinite parameter space is 'coded' to obtain the finite set S .

Genetic algorithms use random choice as a *tool* to guide the search through the set S .

NOTATION

objective

function;

fitness;

parameter set;

strings

In the context of genetic algorithms, the function f is called the *objective function*, and is said to measure the *fitness* of elements in S . The set S is called the *parameter set*, and its elements are called *strings*. It will be seen that the elements of S take the form (p_1, p_2, \dots, p_m) .

advantages of genetic algorithms over other optimization methods

One advantage of genetic algorithms over other optimization methods is that *only the objective function f and set S are needed for implementation*. In particular, there are no continuity or differentiability requirements on the objective function. Also, genetic algorithms work from a wide selection of points simultaneously (instead of a single point), making it far less likely to hit a 'wrong' optimal point.

Genetic algorithms can be particularly useful for determining a 'starting point' for the gradient method, when an analyst has no natural candidate.

basic steps in a genetic algorithm:
Initialization,
Reproduction,
Crossover,
Mutation

Here are the basic steps in a simple genetic algorithm:

- (Initialization) An initial population is randomly selected from S . The objective function f is used to determine the fitness of each choice from the initial population.
- (Reproduction) A new population is produced, based on the fitness of elements in the original population. Strings with a high level of 'fitness' are more likely to appear in the new population.

- (Crossover) This new population is adjusted, by randomly 'matching up and mixing' the strings. This information exchange between the 'fittest' strings is hoped to produce 'offspring' with a fitness greater than their 'parents'.
- (Mutation) *Mutation* is the occasional random alteration of a particular value in a parameter string. In practice, mutation rates are on the order of one alteration per thousand position changes [Gold, p.14]. In the MATLAB implementation discussed in this section, the mutation rate is zero.

generation

- The steps above are repeated as necessary. Each new population, formed by reproduction, crossover and (possibly) mutation, is called a *generation*.

*revisiting an
earlier problem*

An earlier problem is revisited:

**PROBLEM
(P)**

Given a data set $\{(t_i, y_i)\}_{i=1}^N$, find parameters A, B, C_i, D_i , and $P_i := \frac{2\pi}{\omega_i}$ ($i = 1, \dots, m$) so that the function

$$f(t) = A + Bt + \sum_{i=1}^m C_i \sin \omega_i t + D_i \cos \omega_i t$$

minimizes the sum of the squared errors between $f(t_i)$ and the data values y_i :

$$\min \sum_{i=1}^N (f(t_i) - y_i)^2.$$

*a genetic
algorithm
will be used*

A genetic algorithm will be used to search among potential choices for the periods of the sinusoids. For each string (P_1, \dots, P_m) in the parameter set, linear least-squares techniques (Section 2.2) will be used to find the corresponding optimal choices for A, B, C_i and D_i ($i = 1, \dots, m$), which are then used to find the mean-square error associated with that string:

$$\text{error}(P_1, \dots, P_m) = \sum_{i=1}^N (f(t_i) - y_i)^2.$$

The objective function will be defined so that strings with minimum error have maximum fitness, and strings with maximum error have minimum fitness.

The reader is now guided through an implementation of a genetic algorithm, using MATLAB. A wealth of additional information can be found in [Gold]. It may be helpful to look ahead to the EXAMPLE while reading through the following discussion of the function `genetic`.

MATLAB FUNCTION `genetic`

The following MATLAB function uses a genetic algorithm to search for periods (P_1, \dots, P_m) that approximate a solution to (P).

To use the function, type:

```
best = genetic(D,a,b,dT,popsize,strlen,numgen);
```

Important sections of code are analyzed. The entire code is given at the end of this section.

REQUIRED INPUTS

The required inputs are:

- **D** is an $N \times 2$ matrix containing the data set $\{(t_i, y_i)\}_{i=1}^N$. The first column of **D** must contain the time values. The second column contains the corresponding data values.
- **a, b, dT** Let a and b be positive real numbers with $a < b$, and let $\Delta T > 0$. The MATLAB variables **a**, **b** and **dT** correspond, respectively, to a , b and ΔT .

Periods are chosen from the interval $[a, b]$, with increment ΔT , as follows. Let k be the greatest nonnegative integer for which $a + k\Delta T \leq b$. Then, every number in the set $\tilde{S} := \{a, a + \Delta T, a + 2\Delta T, \dots, a + k\Delta T\}$ is in the interval $[a, b]$.



Define

$$S := \{(P_1, \dots, P_i, \dots, P_m) \mid P_i \in \tilde{S}, 1 \leq i \leq m\}.$$

The initial population is chosen from this parameter set S .

popsize

- **popsize** is a positive even integer, that gives the size of the population chosen from S . It is assumed that the population size is constant over all generations.

strlength • **strlength** gives the length of each string in the parameter set S . For example, if each string is of the form (P_1, \dots, P_m) , then **strlength** = m .

numgen • **numgen** gives the number of generations of the genetic algorithm to be implemented.

OUTPUT FROM THE FUNCTION

The function scrolls information about each generation as it is running. This information is described later on.

The function returns a matrix **best** that contains the best string of periods, and related information, from each generation of the genetic algorithm. If two strings have equal fitness, then the first such string is returned in **best**.

Each row of **best** is of the form:

(generation #) (best string of periods) (error) $A \ B \ C_1 \ D_1 \ C_2 \ D_2 \ \dots \ C_m \ D_m$

INITIAL- IZATION

(INITIALIZATION) The genetic algorithm begins with a random selection of the initial population from the parameter set S .

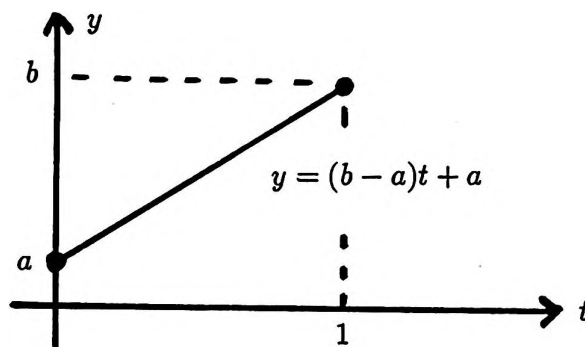
Each line below is numbered for easy reference in the explanations that follow.

```
(1) index = [1:popsizel]';
(2) ran = rand(popsizel, strlength);
(3) oldpop = (b-a)*ran + a;
(4) oldpop = dT*round((1/dT)*(oldpop-a))+a;
(5) oldpop = chkfdup(oldpop, a, b, dT);
(6) error = linlstsqr(oldpop, M, t, y);
(7) averror = sum(error) / popsizel;
(8) [minerror, besti] = min(error);
(9) maxerror = max(error);
(10) M = minerror + maxerror;
(11) fitness = M - error;
(12) sumfit = sum(fitness);
(13) pselect = fitness / sumfit;
(14) expcnt = popsizel * pselect;
```

an explanation of each line

- (1) The column vector **index** is used to number the strings in each population, for easy reference.
- (2) The command **rand(popsizel, strlength)** creates a matrix of size **popsizel** \times **strlength** with entries randomly chosen (using a uniform distribution) from the interval $[0, 1]$.

- (3) The entries in `ran` are mapped to entries in the interval $[a, b]$, via the mapping given below. The resulting matrix is named `oldpop`, for 'old population'.



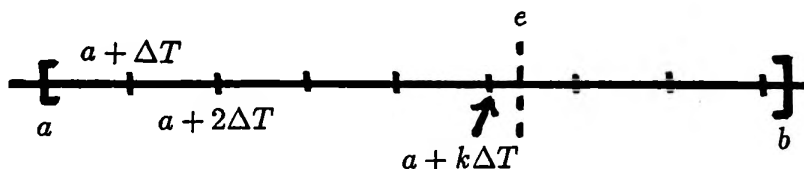
- (4) Each entry in `oldpop` must be replaced by a best approximation from the allowable parameter set, S . Let e denote an entry in `oldpop`. A nonnegative integer k is sought, so that e is closest to $a + k\Delta T$. Since $\Delta T > 0$, one has

$$\begin{aligned} |e - (a + k\Delta T)| &= \left| \Delta T \left(\frac{e}{\Delta T} - \frac{a}{\Delta T} - k \right) \right| \\ &= \Delta T \left| \frac{e - a}{\Delta T} - k \right|. \end{aligned}$$

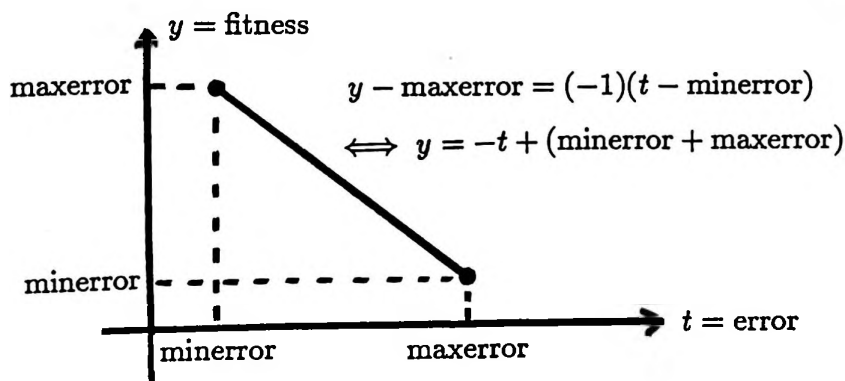
Thus, it suffices to choose an integer k that is closest to $\frac{e-a}{\Delta T}$. This is accomplished for each entry of `oldpop` simultaneously, by use of the `round` command:

```
k = round((1/dT)*(oldpop - a))
```

The desired element in S is then $\Delta T \cdot k + a$.



- chkfdup** (5) The periods in each string (P_1, \dots, P_m) must be distinct. Otherwise, when linear least-squares techniques are applied, the matrix X will have duplicate columns, and hence be singular (Section 2.2). The MATLAB function **chkfdup** checks each string for duplicate periods. Entries are (randomly) adjusted by $\pm\Delta T$ to correct any duplicate values. The code for the function **chkfdup** is included at the end of this section.
- linlstsqr** (6) The function **linlstsqr** finds the parameters A, B, C_i and D_i ($i = 1, \dots, m$) corresponding to each string (P_1, \dots, P_m) in **oldpop**, and computes the associated mean-square error. These errors are returned in the matrix **error**. The code for the function **linlstsqr** is included at the end of this section.
- averror** (7) The average error (**averror**) for the initial population is computed by summing the individual errors, and dividing by the number of strings.
- minerror** (8) The least error, over all the strings, is returned as **minerror**. The row number of the corresponding least-error string is returned in **besti**. If more than one row has the same least error, then the first such row is returned.
- maxerror** (9) The greatest error, over all the strings, is returned as **maxerror**.
- (10-11) The fitness of each string is computed via the linear function below. Observe that minimum error is mapped to maximum fitness, and maximum error is mapped to minimum fitness.



- (12) The 'population fitness' `sumfit` is the sum of the fitnesses of each string in the population.
- (13) The probability that a given string will be selected for the next generation is given by its fitness, divided by the population fitness.
- (14) The (theoretical) expected number of each string in the subsequent population is found by multiplying the probability of selection by the population size.

EXAMPLE

Let `D` be the data set formed by these commands:

```
t = [0:.1:20]';
y = 1 + .5*t + sin(2*pi*t/5) - 2*cos(2*pi*t/5) + 7*cos(2*pi*t/17);
noise = 2*(rand(t) - .5);
y = y + noise;
D = [t y];
```

The reader will recognize this data set from the previous MATLAB example of the gradient method. The sinusoids have periods 5 and 17.

Let `a = 1`, `b = 20`, `dT = 0.5`, `popsiz = 10`, `strlength = 2`, and `numgen = 3`. An application of

```
best = genetic(D,a,b,dT,popsiz,strlength,numgen)
```

(modified to print out more information than usual), yielded the following initial population, associated mean-square errors, fitness, probability of selection, and expected count in the next population:

index	P_1	P_2	error	fitness	pselect	expcnt
1	12.5	14.0	496.1	4535.2	0.1355	1.3545
2	4.5	16.0	341.5	4689.8	0.1401	1.4007
3	16.5	14.0	532.6	4498.7	0.1344	1.3436
4	4.0	15.5	588.3	4443.1	0.1327	1.3270
5	20.0	13.5	553.5	4477.9	0.1337	1.3374
6	6.0	13.0	1181.0	3850.4	0.1150	1.1500
7	5.5	2.0	4689.8	341.5	0.0102	0.1020
8	3.0	12.5	2199.8	2831.6	0.0846	0.8457
9	5.0	5.5	4592.9	438.5	0.0131	0.1310
10	13.0	7.0	1655.5	3375.9	0.1008	1.0083

Observe that the numbers P_1 and P_2 are randomly distributed between 1 and 20, with increment 0.5. The string with the least error is [4.5 16.0]; not surprisingly, since these periods are close to the actual periods 5 and 17.

The average error for this initial population is

$$\text{averror} = \frac{496.1 + 341.5 + \dots + 1655.5}{10} = 1683.1 .$$

Observe that small error corresponds to high fitness, and strings with high fitness have a greater probability of selection (*select*) and expected count (*expcnt*) in the next generation.

The sum of the fitnesses for this first generation is

$$\text{sumfit} = 4535.2 + 4689.8 + \dots + 3375.9 = 33482.6 .$$

Notice that most of the 'high-fitness' strings seem to have a period close to 17. This is because the amplitude of the period-17 sinusoid in the 'known unknown' is 7, whereas the amplitude of the period-5 sinusoid is only $\sqrt{1^2 + (-2)^2} = \sqrt{5}$. In general, periods corresponding to larger amplitude sinusoids will have a greater influence on the string fitness.

information
scrolled
while genetic
is running

As written, the function *genetic* does *not* scroll all the information given in the previous example. It scrolls *only* the information:

```
E = [index oldpop]
error
averror
```

while it is running. This information can be captured by typing *diary* before running *genetic*.

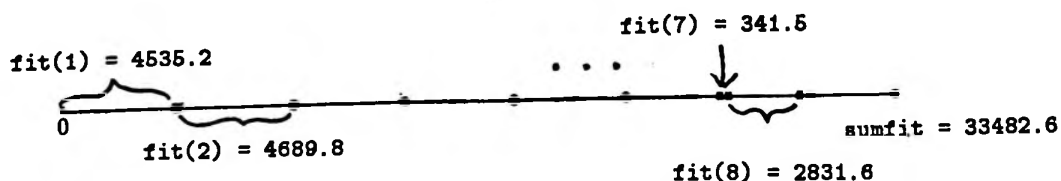
After producing the initial population and statistics, *reproduction* takes place.

REPRO- DUCTION

(REPRODUCTION) The initial population is reproduced, based on the fitness of its strings.

The following reproduction scheme is adapted from [Gold, p. 63].

The interval $[0, \text{sumfit}]$ is partitioned, according to the fitness of each string. Let $\text{fit}(i)$ denote the fitness of string i , for $i = 1, \dots, \text{popsize}$. The sketch below shows the partitioning of $[0, \text{sumfit}] = [0, 33482.6]$ corresponding to the initial population in the previous example:



With this partition in hand, a number is selected randomly (using a uniform distribution) from the interval $[0, \text{sumfit}]$. If the number lands in the subinterval corresponding to string i , then a copy of string i is placed in the next population. This process is repeated `popsiz` times to complete reproduction.

The following MATLAB code implements the ideas just discussed:

```
% initialize 'newpop'
newpop = zeros(oldpop);
for i = 1:popsiz
    partsum = 0;
    j = 0;
    ran = rand(1)*sumfit;
    while ((partsum < ran) & (j <= popsiz))
        j = j+1;
        partsum = partsum + fitness(j);
    end
    newpop(i,:) = oldpop(j,:);
end
```

EXAMPLE (continued)

Shown below is the initial population, `oldpop`, and the reproduced population, `newpop`. Also shown is the expected count, `expcnt`, of each string in `oldpop`, and the actual count, as observed from `newpop`:

oldpop		newpop		expcnt	actualcount
12.5	14.0	6.0	13.0	1.3545	1
4.5	16.0	12.5	14.0	1.4007	2
16.5	14.0	6.0	13.0	1.3436	1
4.0	15.5	4.0	15.5	1.3270	1
20.0	13.5	20.0	13.5	1.3374	2
6.0	13.0	20.0	13.5	1.1500	3
5.5	2.0	16.5	14.0	0.1020	0
3.0	12.5	6.0	13.0	0.8457	0
5.0	5.5	4.5	16.0	0.1310	0
13.0	7.0	4.5	16.0	1.0083	0

CROSSOVER (CROSSOVER) The final step in this simple genetic algorithm is *crossover*, where the strings in `newpop` are 'matched up and mixed' to try and achieve an optimal string, as follows:

EXAMPLE
(continued)

Shown below is `newpop` before crossover, and after crossover. The arrows indicate the strings that were mated. Observe that when `stringlength = 2`, the crossover site is always 1.

before crossover		after crossover	
6.0	13.0	6.0	13.5
12.5	14.0	20.0	13.0
6.0	13.0	6.0	16.0
4.0	15.5	4.5	13.0
20.0	13.5	20.0	15.5
20.0	13.5	4.0	13.5
16.5	14.0	16.5	13.0
6.0	13.0	6.0	14.0
4.5	16.0	12.5	16.0
4.5	16.0	4.5	14.0

Since it is possible to have duplicate values after crossover, `newpop` is checked for duplicates before continuing with the algorithm.

The average error in this reproduced, crossed-over population is 792.2, as compared to 1683.1 for the old population. Thus, the 'fitness' of the overall population has indeed improved!

*diary of
an actual
MATLAB
session*

The following diary of an actual MATLAB session begins by constructing a 'known unknown' consisting of sinusoids with periods 5, 33 and 87, and with close amplitudes. The search for periods is done on the interval [1,100] with increment 1. A population size of 30 is used. Five generations are computed.

By the fifth generation, components with periods 5, 35 and 87 were 'found' with an associated error of only ≈ 140 . What is the probability that this good a result would have been obtained by random choice alone?

The parameter space has $(100)^3$ elements, since there are 100 choices for periods in the interval [1,100] with interval 1. Each generation uses 30 points in this space, so 5 generations yield $5 \cdot 30 = 150$ points in parameter space.

Consider a 'box' around the actual solution point (5,33,87), where each coordinate lies within 2 of the actual solution point: $\{3, 4, 5, 6, 7\} \times \{31, 32, 33, 34, 35\} \times \{85, 86, 87, 88, 89\}$. There are $5^3 = 125$ such points. By merely *randomly* choosing points in space, the probability of hitting a point in this 'box' on a single draw is $\frac{125}{(100)^3} = 0.000125$.

On N draws, what is the probability of hitting a point within this box?

$$\begin{aligned}
&P[(\text{point in box on draw \#1}) \text{ OR } (\text{point in box on draw \#2}) \\
&\quad \text{OR } \dots \text{ OR } (\text{point in box on draw \#N})] \\
&= 1 - P(\text{point is NOT in box on all } N \text{ draws}) \\
&= 1 - (1 - .000125)^N \\
&\approx 0.0186, \text{ when } N = 150.
\end{aligned}$$

*an illustration of
the power of the
genetic algorithm*

Thus, the probability of hitting a point in the box on 150 draws, by random search alone, is less than 2% ! This example illustrates the power of the genetic algorithm.

It is again noted that if one sinusoid has an amplitude considerably larger than others in the sum, then that sinusoid gets 'favored' in the search process.

The summary of the 'best' choices from each generation, together with the first generation, are shown.

```

t = [1:100]';
y = 5 + .1*t + 3.9*cos((2*pi/5)*t) - 4.1*sin((2*pi/33)*t);
y = y + 2.3*cos((2*pi/87)*t) - 3.3*sin((2*pi/87)*t);
noise = 3*(rand(t) - .5);
y = y + noise;
D = [t y];
best = genetic(D,1,100,1,30,3,5)

```

best =

Columns 1 through 7

generation #	P_1	P_2	P_3	error	A	B
1.0000	52.0000	25.0000	5.0000	721.3155	-0.4123	0.2061
2.0000	40.0000	43.0000	5.0000	754.4211	0.9280	0.1775
3.0000	33.0000	47.0000	5.0000	422.0458	1.3175	0.1721
4.0000	33.0000	19.0000	5.0000	531.5720	2.0746	0.1570
5.0000	87.0000	35.0000	5.0000	140.4562	5.0390	0.0964

Columns 8 through 13

C_1	D_1	C_2	D_2	C_3	D_3
2.4387	0.2409	1.1601	0.0237	0.0507	3.7751
3.2776	-1.6336	-2.0687	-1.9841	0.0880	3.7534
-2.8692	-0.4977	1.5142	-0.7472	0.0310	3.7887
-3.2706	-0.4404	0.2535	-0.1147	-0.0036	3.8149
-3.5699	2.3719	-3.3266	-2.3252	-0.0883	3.8785

E			
1	13	53	34
2	53	67	80
3	17	87	64
4	52	25	5
5	11	71	5
6	84	27	16
7	37	39	77
8	43	9	7
9	33	47	52
10	72	24	93
11	58	80	23
12	87	50	21
13	76	1	10
14	84	27	70
15	70	67	71
16	40	43	50
17	94	19	27
18	61	96	71
19	37	14	38
20	54	80	21
21	46	95	24
22	41	18	44
23	25	61	47
24	9	7	86
25	43	35	98
26	48	30	45
27	30	51	4
28	11	83	53
29	22	84	4
30	80	73	76

FIRST GENERATION

bfitgen

The function **bfitgen** ('best fit from the genetic algorithm') is convenient for plotting the best approximation obtained from an application of the genetic algorithm,

```
G = genetic(D,a,b,dT,popsize,strings,numgen);
```

To use the function **bfitgen**, type:

```
[yb,rowofG,per,coef] = bfitgen(t,G);
```

INPUTS

The required inputs are:

- the matrix **G** from an application of the genetic algorithm;
- a column vector **t** used to compute the data values of the best approximation. Often, one uses $t = D(:,1)$.

OUTPUTS

- The column vector **yb** contains the values of the best approximation. To plot this best approximation, type either: `plot(t,yb)` or `plot(t,yb,'x')`.
- The optional outputs **rowofG**, **per**, and **coef** contain, respectively, the generation number in which the best approximation was obtained, the periods $P_1 \dots P_m$ of the best approximation, and the corresponding coefficients $A \ B \ C_1 \ D_1 \dots C_m \ D_m$ of the best approximation.

the programs The code for *genetic* and the necessary auxiliary functions is given next:

```
% copyright 1993 Carol J.V. Fisher
function best = genetic(D,a,b,dT,popsize,stringlength,numgen)
% D is the data set:
%   the first column contains the time values
%   the second column contains the corresponding data values
% a,b,dT: the (unknown) periods are chosen from the interval [a,b],
%   with positive increment dT;
% 'popsize' is a positive even integer, giving the (constant) population size
% 'stringlength' is a positive integer, giving the string length;
%   that is, the number of periods used for approximation
% 'numgen' is a positive integer, giving the maximum number of generations
%
t = D(:,1);
N = length(t);
y = D(:,2);
% gen contains the generation number
gen = 0;
best = zeros(numgen,4+3*stringlength);
%
% INITIALIZATION: select the initial population
index = [1:popsize]';
% Randomly choose the initial population of strings
ran = rand(popsize,stringlength);
% oldpop is the 'old' population; before reproduction and crossover
% This command maps the numbers in [0,1] to numbers in [a,b]
oldpop = (b-a)*ran + a;
% discretize oldpop; let 'n' denote an element of oldpop
% Want integer k such that n is closest to a + kdT;
% Equivalently, (n-a)/dT is closest to k.
oldpop = dT*round((1/dT)*(oldpop-a))+a;
% check 'oldpop' for duplicate period values;
% if so, change the duplicate values as little as possible
oldpop = chkfdup(oldpop,a,b,dT);
%
while gen < numgen
    % find the corresponding parameters A,B,Ci and Di, and
    % compute the mean-square error corresponding to the strings in oldpop
    error = linlstsqr(oldpop,N,t,y);
    % compute the fitness; note that small error gives large fitness
    % Here, min(error) is mapped to max(fitness), and max(error) is mapped
    % to min(fitness). The index of the 'best' string is stored in 'besti'
    [minerror,besti] = min(error);
    maxerror = max(error);
    M = minerror + maxerror;
    fitness = M - error;
    % STATISTICS:
    sumfit = sum(fitness);
    averror = sum(error) / popsize;
    pselect = fitness / sumfit;
    expcnt = popsize * pselect;
    % print out the results of each generation
    E = [index oldpop]
    error
    averror
    % start a new generation
    gen = gen+1;
    % Find the parameters corresponding to the best in this generation
    [besterror,bestpar] = linlstsqr(oldpop(besti,:),N,t,y);
    % The elements of the matrix 'best' are of the form:
    % (generation #) (string of periods) (mean-square error) A B C1 D1 ...
```



```

best(gen,:) = [gen oldpop(best1,:) besterror bestpar'];
%
% REPRODUCTION
% initialize 'newpop'
newpop = zeros(oldpop);
% (based on [Gold, p. 63])
% A fitness value is randomly chosen from [0,sumfit]. This interval
% is partitioned according to each string's fitness, and an entry is
% selected accordingly (see diagram at right).
for i = 1:popsize
    partsum = 0;
    j = 0;
    ran = rand(1)*sumfit;
    while ((partsum < ran) & (j <= popsize))
        j = j+1;
        partsum = partsum + fitness(j);
    end
    newpop(i,:) = oldpop(j,:);
end
%
% CROSSOVER
for j = 1:2:(popsize-1)
    % if stringlength = 1, then there is no crossover
    if stringlength==1
        break
    end
    mateindex = selint(j+1,popsize);
    xsite = selint(1,(stringlength-1));
    ind = (xsite+1):stringlength;
    temp = newpop(j,:);
    newpop(j,ind) = newpop(mateindex,ind);
    newpop(mateindex,ind) = temp(ind);
    % switch rows so that 'mates' are together at beginning of matrix
    temp = newpop(j+1,:);
    newpop(j+1,:) = newpop(mateindex,:);
    newpop(mateindex,:) = temp;
end
% check the new population for duplicates
newpop = chkfdup(newpop,a,b,dT);
oldpop = newpop;
end

```

```

%This computes the function of 'best fit' from the genetic algorithm
function [yb,rowofG,per,coef] = bfitgen(t,G)
%First, find the entry with the least mean-square error
[m,n] = size(G);
% Let P be the number of periods being sought
% The number of columns of G is 1+P+1+2+2P = 3P+4
P = (n-4)/3;
[minerror,rowofG] = min(G(:,(P+2)));
G = G(rowofG,:);
per = G(2:(1+P));
coef = G(P+3:n);
yb = coef(1) + t*coef(2);
for j = 1:P
    yb = yb + coef(2*j+1)*sin(2*pi*t/per(j)) + coef(2*j+2)*cos(2*pi*t/per(j));
end

```

```

% copyright 1993 Carol J.V. Fisher
function i = selint(j,k)
% this program returns an INTEGER selected randomly (uniform distribution)
% between integers 'j' and 'k', where j < k
i = floor(rand(1)*(k-j+1)+j);
if i > k
    i = k;
end

```

```

% copyright 1993 Carol J.V. Fisher
function [error,bestpar] = linlstsqr(oldpop,N,t,y)
[popsize,stringsh] = size(oldpop);
% This computes the error corresponding to each string in oldpop
w = 2*pi ./ oldpop;
% Initialize the error matrix
error = zeros(popsize,1);
% find the X matrix necessary for linear least squares
X = zeros(N,2*stringsh+2);
X(:,1) = ones(t);
X(:,2) = t;
for i = 1:popsize
    k = 1;
    for j = 1:2*stringsh
        X(:,j+2) = sin(w(i,k)*t);
        X(:,j+3) = cos(w(i,k)*t);
        k = k+1;
    end
    if rcond(X'*X) > 1e-15
        b = (X'*X) \ (X'*y);
    else
        [M,F] = dscorth(t,X,(2*stringsh+2));
        b = (F'*F) \ (F'*y);
        b = M'*b;
    end
    f = X*b;
    error(i) = (y-f)'*(y-f);
end
% The best parameters are optionally returned when only one
% row is being processed.
bestpar = b;

```

```

% copyright 1993 Carol J.V. Fisher
function nodup = chkfdup(oldpop,a,b,dT)
[popsize,stringlength] = size(oldpop);
nodup = oldpop;
i = 1;
% Sort each row in increasing order, put in z . Thus, any identical
% entries will be adjacent in z .
% 'count' will keep track of the number of times a string is checked
% for duplicate entries; if count > 10, go on to next string
count = 0;
while i <= popsize
    [z,k] = sort(nodup(i,:));
    flag = 0;
    j = 1;
    while (j < stringlength) & (flag==0)
        if z(j) == z(j+1)
            % If a duplicate entry is found, set 'flag' to 1. Adjust the second
            % entry by (plus or minus)dT (randomly chosen) and recheck the
            % string for other duplicates
            flag = 1;
            R = rand;
            if ((R >= .5) & (nodup(i,k(j+1)) <= (b-dT))) | (nodup(i,k(j+1)) <= (a+dT))
                sgn = 1;
            else
                sgn = -1;
            end
            nodup(i,k(j+1)) = nodup(i,k(j+1)) + sgn*dT;
        end
        j = j+1;
    end
    % If a duplicate was found and count <= 10 , recheck the string
    if flag == 1
        count = count+1;
        i = i - 1;
    end
    if (count > 10)
        count = 0;
        i = i + 1;
    end
    i = i+1;
end
end

```

2.5 Cubic Spline Interpolation

*missing values
in a data set*

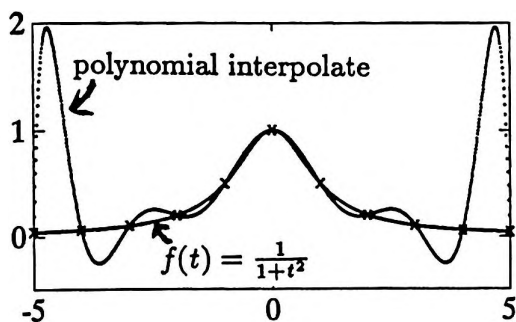
Occasionally, an analyst may choose to 'fill in' (interpolate) some missing values in a data set—perhaps to achieve a uniform time list, or to replace a data point that is clearly in error. The validity of such replacement of actual data with artificial data should always be carefully considered. When it is decided that such a replacement should take place, then *splines* are a useful tool for determining the replacement data value(s).

*What is
a 'spline'?*

In mathematical literature, a *spline* is a function formed by piecing together other functions, achieving some degree of smoothness (differentiability) in the final 'pieced together' curve. Splines have applications in graphics, where, for example, a set of points must be connected with a smooth curve or surface [Prenter, Chap. 5]. Splines also have applications in the numerical solutions of partial and differential equations [Prenter, Chaps. 7&8]. The current section discusses an important class of spline functions, called *cubic splines*, and MATLAB implementation of the ideas herein.

*polynomials
can exhibit
large oscillations*

When polynomials are used to interpolate data, it is possible to get large oscillations between data points. This phenomenon is illustrated in the graph below, where a polynomial is fitted to uniformly-spaced data points from the function $f(t) = \frac{1}{1+t^2}$. Splines offer an interpolation approach that does NOT yield such oscillations.

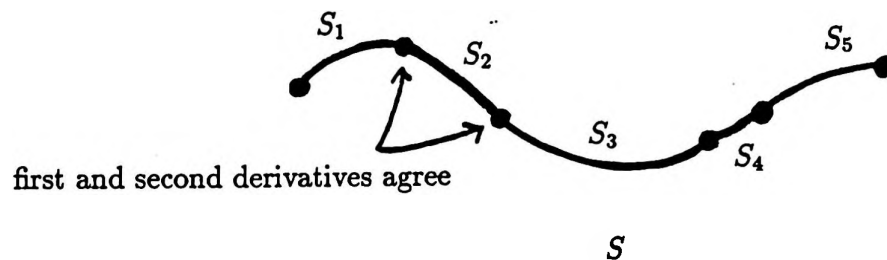


'x' = data points used for interpolation

What are
cubic splines?

Roughly, a *cubic spline* is a function formed by *patching together* cubic polynomials, forcing the function values, first and second derivatives to agree at the *patching points*. In this way, a curve is obtained that has a continuous second derivative.

The cubic polynomial on the i^{th} subinterval is called S_i . The entire spline (formed from the pieced-together cubics) is called S .

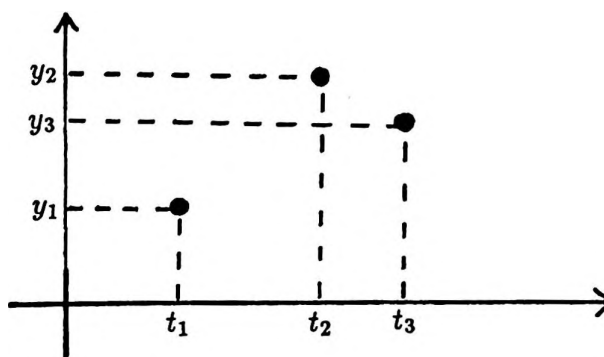


$(t_1, y_1), (t_2, y_2),$
etc.
are the 'knots'

The ideas behind cubic spline interpolation can be illustrated using just three data points; call them

$$(t_1, y_1), (t_2, y_2), (t_3, y_3), \quad t_1 < t_2 < t_3.$$

These points need not be equally spaced. In the context of splines, interpolating points are commonly referred to as 'knots'.



$S_1(t), S_2(t)$

The t -values of these three data points naturally yield two subintervals, $[t_1, t_2]$ and $[t_2, t_3]$, called the *first* and *second* subinterval, respectively. On the first subinterval, let

$$S_1(t) := c_{10} + c_{11}t + c_{12}t^2 + c_{13}t^3 .$$

Similarly, on the second subinterval, let

$$S_2(t) := c_{20} + c_{21}t + c_{22}t^2 + c_{23}t^3 .$$

The following naming convention is used for the polynomial coefficients:

- The first subscript in the coefficient c_{ij} agrees with the subscript on S_i , and gives the subinterval on which S_i is defined. Thus, S_1 is a cubic polynomial on the first subinterval, which has coefficients c_{1j} .
- The second subscript in the coefficient c_{ij} tells the power of t that the coefficient multiplies. For example, c_{10} multiplies the t^0 (constant) term, and c_{13} multiplies the t^3 term.

there are
8 unknowns

The cubic spline is formed by appropriately patching together S_1 and S_2 . Observe that there are 8 unknown coefficients: $c_{10}, \dots, c_{13}, c_{20}, \dots, c_{23}$. Thus, due to the linear nature of the problem, 8 pieces of (non-overlapping, non-contradictory) information are needed to solve uniquely for these coefficients.

Requiring that the curves pass through the appropriate data points yields 4 equations:

$$S_1(t_1) = y_1$$

$$S_1(t_2) = y_2$$

$$S_2(t_2) = y_2$$

$$S_2(t_3) = y_3$$

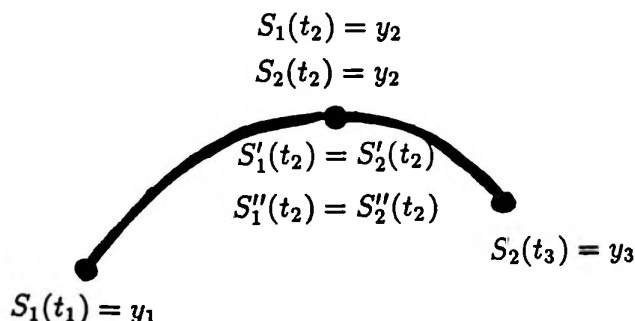
Requiring that the first and second derivatives agree at the 'patching point' yields 2 more equations:

$$S_1'(t_2) = S_2'(t_2)$$

$$S_1''(t_2) = S_2''(t_2)$$

There are two additional pieces of information needed.

Observe that the resulting spline function S will only be defined on the interval $[t_1, t_3]$. Therefore, S is useless for predictive purposes.



*more general
counting
argument*

Here is the scenario when there are $N \geq 2$ data points: in this case, there are $N - 1$ subintervals, and hence $N - 1$ cubic polynomials being patched together. Thus, there are $4(N - 1) = 4N - 4$ unknowns.

Requiring that the spline pass through the data points yields 2 equations (for the endpoints), plus $2(N - 2)$ equations (for the interior points). This gives a total of $2 + 2(N - 2) = 2N - 2$ equations.

Requiring that the first and second derivatives agree at the interior points yields another $2(N - 2) = 2N - 4$ equations.

Together, this yields $(2N - 2) + (2N - 4) = 4N - 6$ equations. Again, two additional equations are required to reach the number of unknowns, $4N - 4$.

*the natural
cubic spline*

One way to impose the remaining two conditions is to require that the second derivative be zero at the endpoints; for the situation where there are only three points, this gives

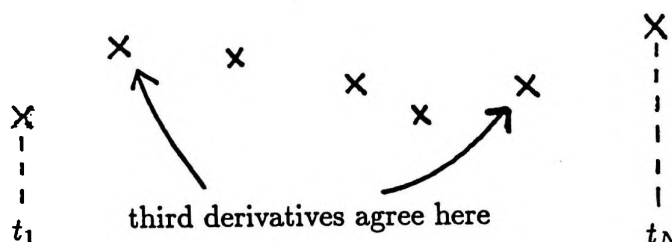
$$S_1''(t_1) = S_2''(t_3) = 0.$$

The resulting spline is called the 'natural' cubic spline function. A program for computing natural cubic splines is included at the end of this section.

There are other ways to impose the remaining two conditions. For example, the 'not-a-knot' condition requires that

$$S_1'''(t_2) = S_2'''(t_2) \quad \text{and} \quad S_{N-2}'''(t_{N-1}) = S_{N-1}'''(t_{N-1}) .$$

In this case, the functions S_1 and S_2 agree in function value, first, second and third derivatives at t_2 , from which it follows that S_1 is identical to S_2 . Similarly, the functions S_{N-2} and S_{N-1} must be identical. Hence, the points (t_2, y_2) and (t_{N-1}, y_{N-1}) are *not* knots, and the name is appropriate. The built-in MATLAB `spline` command uses the 'not-a-knot' condition.



*the natural
cubic spline
minimizes
curvature*

The natural cubic spline has the property that it minimizes the integral

$$\int_{t_1}^{t_3} (f''(t))^2 dt$$

over *all possible* functions f that have continuous second derivatives on the interval $[t_1, t_3]$ [S&B, p. 96]. What is the significance of this minimization property? The answer lies in the fact that the second derivative of a function measures how 'curvy' that function is, as follows:

*the second
derivative
of a function
measures
how 'curvy' the
function is*

Recall that if a function f is differentiable at t , then $f'(t)$ gives the *instantaneous rate of change* of the function values $f(t)$ with respect to the inputs t , at the point $(t, f(t))$.

If f is twice differentiable at t , then the second derivative $(f')'(t) := f''(t)$ gives the instantaneous rate of change of the slopes $f'(t)$ with respect to t ; that is, f'' measures how fast the curve 'turns'.

For example, if the second derivative is large and positive, then the slopes increase quickly; and if the second derivative is small and positive, the slopes increase slowly:



f'' large and positive;
turns quickly



f'' small and positive;
turns slowly

In this way, the second derivative is a measure of 'curviness'. The integral $\int_{t_1}^{t_3} (f''(t))^2 dt$ 'sums' the contributions of $(f''(t))^2$ on the interval $[t_1, t_3]$. (The quantity $f''(t)$ is squared, since one is only interested in the magnitude of f'' , and not its sign.)

Thus, the number $\int_{t_1}^{t_3} (f''(t))^2 dt$ is a measure of the 'curviness' of the function f on the interval $[t_1, t_3]$. With respect to this measure of 'curviness', the natural cubic spline is the least curvy interpolate of the data points, over all possible interpolates that have a continuous second derivative.

In practice, the natural cubic spline is very similar to the 'not-a-knot' spline. One example, illustrating how closely they agree, is included at the end of this section.

*rewriting the
constraints
in terms of
the coefficients c_{ij}*

Returning to the simple case of three data points, the 8 imposed conditions for the natural cubic spline can be rewritten in terms of the coefficients c_{ij} , yielding a system of 8 equations in 8 unknowns.

Observe that with

$$S_i(t) = c_{i0} + c_{i1}t + c_{i2}t^2 + c_{i3}t^3 ,$$

one has

$$S'_i(t) = c_{i1} + 2c_{i2}t + 3c_{i3}t^2$$

and

$$S''_i(t) = 2c_{i2} + 6c_{i3}t .$$

The 8 constraints for the natural cubic spline are therefore

$$\begin{aligned}
 c_{10} + c_{11}t_1 + c_{12}t_1^2 + c_{13}t_1^3 &= y_1 \\
 c_{10} + c_{11}t_2 + c_{12}t_2^2 + c_{13}t_2^3 &= y_2 \\
 c_{20} + c_{21}t_2 + c_{22}t_2^2 + c_{23}t_2^3 &= y_2 \\
 c_{20} + c_{21}t_3 + c_{22}t_3^2 + c_{23}t_3^3 &= y_3 \\
 c_{11} + 2c_{12}t_2 + 3c_{13}t_2^2 &= c_{21} + 2c_{22}t_2 + 3c_{23}t_2^2 \\
 2c_{12} + 6c_{13}t_2 &= 2c_{22} + 6c_{23}t_2 \\
 2c_{12} + 6c_{13}t_1 &= 0 \\
 2c_{22} + 6c_{23}t_3 &= 0 ,
 \end{aligned}$$

or, in matrix form,

$$\begin{array}{c} \overbrace{\begin{bmatrix} 1 & t_1 & t_1^2 & t_1^3 & 0 & 0 & 0 & 0 \\ 1 & t_2 & t_2^2 & t_2^3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & t_2 & t_2^2 & t_2^3 \\ 0 & 0 & 0 & 0 & 1 & t_3 & t_3^2 & t_3^3 \\ 0 & 1 & 2t_2 & 3t_2^2 & 0 & -1 & -2t_2 & -3t_2^2 \\ 0 & 0 & 2 & 6t_2 & 0 & 0 & -2 & -6t_2 \\ 0 & 0 & 2 & 6t_1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 2 & 6t_3 \end{bmatrix}}^A \end{array} \begin{array}{c} \overbrace{\begin{bmatrix} c_{10} \\ c_{11} \\ c_{12} \\ c_{13} \\ c_{20} \\ c_{21} \\ c_{22} \\ c_{23} \end{bmatrix}}^c \end{array} = \begin{array}{c} \overbrace{\begin{bmatrix} y_1 \\ y_2 \\ y_2 \\ y_3 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}}^y \end{array} \quad (*)$$

For convenience, denote this in the form $Ac = y$, with appropriate definitions for A , c and y . It can be shown that the matrix A is always invertible [S&B, p. 101], so the coefficients are given (at least theoretically) by

$$c = A^{-1}y .$$

EXAMPLE

As a simple example, the natural cubic spline through the three data points

$$(0,1), (1,3), (2,2)$$

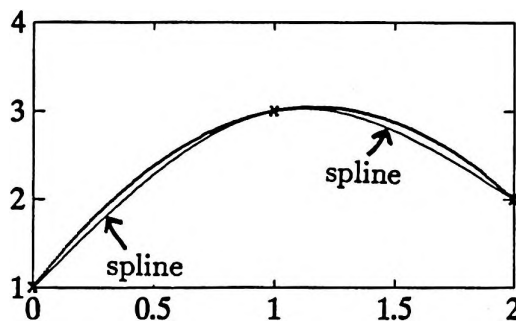
is computed. The matrix (*) becomes:

$$\begin{array}{c} \begin{bmatrix} 1 & 0 & 0^2 & 0^3 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1^2 & 1^3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1^2 & 1^3 \\ 0 & 0 & 0 & 0 & 1 & 2 & 2^2 & 2^3 \\ 0 & 1 & 2 & 3 & 0 & -1 & -2 & -3 \\ 0 & 0 & 2 & 6 & 0 & 0 & -2 & -6 \\ 0 & 0 & 2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 2 & 12 \end{bmatrix} \end{array} \begin{array}{c} \begin{bmatrix} c_{10} \\ c_{11} \\ c_{12} \\ c_{13} \\ c_{20} \\ c_{21} \\ c_{22} \\ c_{23} \end{bmatrix} \end{array} = \begin{array}{c} \begin{bmatrix} 1 \\ 3 \\ 3 \\ 2 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \end{array} .$$

Solution of this system (here, using MATLAB) yields the c column vector:

```
c =
    1.0000
    2.7500
     0
   -0.7500
   -0.5000
    7.2500
   -4.5000
    0.7500
```

The spline is graphed below. On the same graph is shown the unique parabola (degree 2 polynomial) that passes through these three points. Observe that the spline is 'less curvy' than the parabola. (To see this, note that the 'least curvy' path between, say, the points (0,1) and (1,3), is the line segment connecting these two points—and the spline is closer to this line segment.)



*the matrices get
very large,
very fast*

One has probably noted that the matrix A in the system (*) gets very large, very fast, as the number of data points increases. Numerical methods to compute splines take a dramatically different approach to the problem, which results in smaller matrix sizes. The interested reader is referred to [S&B, 97–102] for details.

MATLAB IMPLEMENTATION

Polynomial and Spline Interpolation

- PURPOSE** MATLAB provides built-in commands for polynomial approximation and interpolation, and for cubic spline interpolation (with the 'not-a-knot' condition). In what follows, let \mathbf{t} be a vector containing the time values of a data set $\{(t_i, y_i)\}_{i=1}^N$, and let \mathbf{y} be a vector containing the corresponding data values. The entries of \mathbf{t} should all be distinct. There are N data points.
- MATLAB COMMANDS:**
- polyfit** The command
 $\mathbf{p} = \text{polyfit}(\mathbf{t}, \mathbf{y}, k)$
 fits the data set with a polynomial of degree k . The returned vector \mathbf{p} contains the coefficients of the fitting polynomial, $p(t)$, in descending powers of t .
 If $k \geq N - 1$, then the resulting polynomial will pass through all the data points; in this case, the polynomial is an *interpolate* of the data set. If $k < N - 1$, then the resulting polynomial fits the data in a least-squares sense.
- polyval** The output \mathbf{p} from the previous command can be input into the **polyval** ('polynomial values') command, in order to plot the 'fitting' polynomial. Whenever \mathbf{p} is a vector whose elements are the coefficients of a polynomial in descending powers, then the command
 $\mathbf{yfit} = \text{polyval}(\mathbf{p}, \mathbf{tfine})$
 returns a vector \mathbf{yfit} that contains the polynomial \mathbf{p} evaluated at each element in \mathbf{tfine} . The command
 $\text{plot}(\mathbf{tfine}, \mathbf{yfit}, '.')$
 can then be used to view the fitting polynomial.
- MATLAB COMMAND:**
- spline** Let S be the 'not-a-knot' cubic spline that passes through the data points $\{(t_i, y_i)\}_{i=1}^N$. Let \mathbf{t} and \mathbf{y} be vectors containing the time and data values, respectively, and let \mathbf{tfine} be a vector containing numbers from the interval $[t_1, t_N]$. The command
 $\mathbf{yspl} = \text{spline}(\mathbf{t}, \mathbf{y}, \mathbf{tfine})$
 returns, in the vector \mathbf{yspl} , the values $S(\mathbf{tfine})$.
- EXAMPLE** The following diary of an actual MATLAB session illustrates the use of the MATLAB commands discussed here.
- diary of
 an actual
 MATLAB session* A noisy data set, with a missing value, is generated. Both the **polyfit** and **spline** commands are used to fill in the missing value.

```

% Note that t = 3 is 'missing'
t = [0 1 2 4 5 6 7 8 9 10];
noise = .3*(rand(t) - .5);
y = sin(t) + noise;
subplot(221)
plot(t,y,'x')
hold

Current plot held

tfine = [0:.01:10];
yfine = sin(tfine);
plot(tfine,yfine)

% 3 data points on each side of t = 3 will be used
i = [1 2 3 4 5 6];

% First, polynomial approximation with a degree 2 polynomial
p = polyfit(t(i),y(i),2);
yintpol = polyval(p,3)

yintpol =

    0.1420

plot(3,yintpol,'o')
% Next, spline interpolation
yspl = spline(t(i),y(i),3)

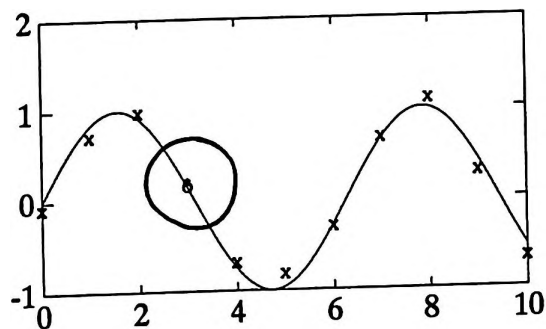
yspl =

    0.2051

plot(3,yspl,'*')

% The two interpolation points are so close that they are hard to
% distinguish from one another. These points are circled in
% the graph below.

```



The MATLAB function given next computes the *natural* cubic spline. An example illustrating its similarity to the 'not-a-knot' spline follows.

```
% copyright 1993 Carol J.V. Fisher
% This function computes the unique CUBIC SPLINE S(x)
% that passes through the N data points (x_i,y_i), i = 1,...,N ,
% and that satisfies S''(x_1) = S''(x_N) = 0 .
%
% This spline is then used to interpolate:
% let xfine = (xf_1,...,xf_M) be an input vector whose corresponding
% outputs on the spline are desired:
% the components must be listed in increasing order; i.e.,
% xf_1 < xf_2 < ... < xf_M .
% Also, it is required that xf_1 >= x_1 and xf_M <= x_N . (Splines
% can only be used for INTERPOLATION, not EXTRAPOLATION.)
%
% To use this MATLAB function, type:
% v = cfspline(x,y,xfine)
% where v = <vector to contain output>
% OR
% [v,C] = cfspline(x,y,xfine)
% where
% C = <matrix to contain coefficients of cubic polynomials
%      on each subinterval>
% x is a vector containing the inputs x_i, i = 1,...,N ;
% y is a vector containing the outputs y_i, i = 1,...,N ;
% xfine is a vector containing the x-values xf_i of desired
% interpolation points; xfine may be of any size. See above
% comments for additional requirements on xfine.
%
% The vectors may be row or column vectors; the output is returned
% in the same form as the input data.
%
% The function returns the spline values S(xf_i) in v .
% The function optionally returns the coefficients
% c_(10) c_(11) c_(12) c_(13)
% c_(20) c_(21) c_(22) c_(23) etc.
% in C .
% The algorithm used is described on pp. 97--102 of:
% Introduction to Numerical Analysis, by J. Stoer and R. Bulirsch
%
function [output,C] = cfspline(x,y,xfine)
N = length(x);
Nx = size(x); if Nx(1) ~= 1, x = x'; end
Ny = size(y); if Ny(1) ~= 1, y = y'; end
Nxfine = size(xfine); if Nxfine(1) ~= 1, xfine = xfine'; end
h = diff(x); hdf = h(2:N-1); hdl = h(1:N-2); hp = hdf + hdl;
L = hdf ./ hp;
MU = 1 - L;
MU = [MU 0];
L = [0 L];

dy = diff(y); dydf = dy(2:N-1); dydl = dy(1:N-2);
D = (6 ./ hp) .* ( (dydf ./ hdf) - (dydl ./ hdl) ) ;
D = [0 D 0];
A = diag(2*ones(N,1)) + diag(L,1) + diag(MU,-1);
M = inv(A) * D'; M = M';
Mdf = M(2:N); Mdl = M(1:N-1);
C0 = y(1:N-1);
C1 = (dy ./ h) - (1/6)*(2*Mdl + Mdf) .* h;
C2 = (.5) * Mdl;
C3 = (Mdf - Mdl) ./ (6*h);
C = [C0;C1;C2;C3];
k = 1;
sxf = length(xfine);
xfine = [xfine 0];
```

```

for i = 1:(N-1),
    while k <= sxf & xfine(k) <= x(i+1),
        dx = xfine(k) - x(i);
        output(k) = C0(i) + C1(i)*dx + C2(i)*(dx)^2 + C3(i)*(dx)^3;
        k = k+1;
    end
end
if Nx(1) ~= 1, output = output'; end

```

EXAMPLE

```

t = [0:10];
y = sin(t);
subplot(221)
plot(t,y,'x')
hold

```

Current plot held

```

tfine = [0:.1:10];
yspl = spline(t,y,tfine);
plot(tfine,yspl)
natspl = cfspline(t,y,tfine);
plot(tfine,natspl,'.')
% The two curves are so close that they are hard to distinguish.
% The 'difference curve' is plotted next.
hold

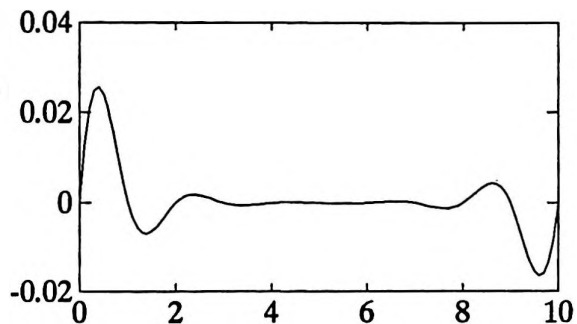
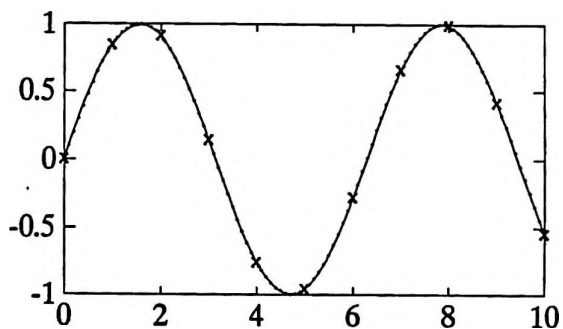
```

Current plot released

```

diff = yspl - natspl;
plot(tfine,diff)

```



2.6 Discrete Fourier Series and the Periodogram

Introduction

Suppose $\{(t_i, y_i)\}_{i=1}^N$ is a data set with a *uniform* time list; let $\Delta T > 0$ be such that $t_{k+1} - t_k = \Delta T$ for $k = 1, \dots, N-1$. If it is suspected that this data set has at least one periodic component, but there is no specific conjecture as to its period or form, then the *discrete Fourier series* corresponding to the data may give useful information regarding the periodic component.

The reader should compare the results in this section with the corresponding results regarding *continuous* Fourier series, which were reviewed in Section 1.6.

The next few results will lead to the definition of the *discrete Fourier series*.

THEOREM 1 Consider the constant function $f(t) \equiv 1$, and the functions

$$\sin \frac{2\pi kt}{P} \quad \text{and} \quad \cos \frac{2\pi kt}{P}$$

for $k = 1, \dots, K$, where P is a positive real number, and K is a positive integer. This collection of $2K + 1$ functions is mutually orthogonal with respect to the time values t_1, \dots, t_N , where $N > 1$ is a positive integer, if the following conditions are met:

- The list (t_1, t_2, \dots, t_N) is uniform with $\Delta T = \frac{P}{N}$, and
- $N \geq 2K + 1$.

PROOF of Theorem 1

Recalling the definition of *mutually orthogonal functions* from page 165, it must be shown that

$$\sum_{n=1}^N \cos \frac{2\pi jt_n}{P} \cos \frac{2\pi kt_n}{P} = 0, \quad j, k = 1, \dots, K, \quad j \neq k, \quad (1)$$

$$\sum_{n=1}^N \sin \frac{2\pi jt_n}{P} \sin \frac{2\pi kt_n}{P} = 0, \quad j, k = 1, \dots, K, \quad j \neq k, \quad (2)$$

$$\sum_{n=1}^N \sin \frac{2\pi jt_n}{P} \cos \frac{2\pi kt_n}{P} = 0, \quad j, k = 1, \dots, K, \quad \text{and} \quad (3)$$

$$\sum_{n=1}^N \cos \frac{2\pi kt_n}{P} = \sum_{n=1}^N \sin \frac{2\pi kt_n}{P} = 0, \quad k = 1, \dots, K. \quad (4)$$

The proof makes use of the identities

$$\cos t = \frac{1}{2}(e^{it} + e^{-it}) \quad \text{and} \quad \sin t = \frac{1}{2i}(e^{it} - e^{-it}), \quad \forall t \in \mathbb{R},$$

and Proposition 2 from Section 1.6, which states that

$$\sum_{n=0}^{N-1} e^{2\pi k i t_n / P} = 0 \quad \text{and} \quad \sum_{n=0}^{N-1} e^{-2\pi k i t_n / P} = 0$$

whenever N and k are positive integers with $N > 1$, and $1 \leq k \leq N-1$. Since the time list is uniform with spacing $\frac{P}{N}$, one can write $t_n = t_1 + (n-1)\frac{P}{N}$ for $n = 1, \dots, N$. Let $j, k = 1, \dots, K$ with $j \neq k$. Then,

$$\begin{aligned} & \sum_{n=1}^N \cos \frac{2\pi j t_n}{P} \cos \frac{2\pi k t_n}{P} \\ &= \sum_{n=1}^N \cos \left(\frac{2\pi j t_1}{P} + \frac{2\pi j(n-1)}{N} \right) \cos \left(\frac{2\pi k t_1}{P} + \frac{2\pi k(n-1)}{N} \right) \\ &= \sum_{n=0}^{N-1} \cos \left(\frac{2\pi j t_1}{P} + \frac{2\pi j n}{N} \right) \cos \left(\frac{2\pi k t_1}{P} + \frac{2\pi k n}{N} \right) \\ &= \frac{1}{4} \sum_{n=0}^{N-1} \left(e^{i(\frac{2\pi j t_1}{P} + \frac{2\pi j n}{N})} + e^{-i(\frac{2\pi j t_1}{P} + \frac{2\pi j n}{N})} \right) \left(e^{i(\frac{2\pi k t_1}{P} + \frac{2\pi k n}{N})} + e^{-i(\frac{2\pi k t_1}{P} + \frac{2\pi k n}{N})} \right). \end{aligned}$$

To simplify notation, let $C_j := \frac{2\pi j t_1}{P}$ and $C_k := \frac{2\pi k t_1}{P}$. Observe that both C_j and C_k are independent of the index of summation, n . With this notation,

$$\begin{aligned} & \frac{1}{4} \sum_{n=0}^{N-1} \left(e^{i(\frac{2\pi j t_1}{P} + \frac{2\pi j n}{N})} + e^{-i(\frac{2\pi j t_1}{P} + \frac{2\pi j n}{N})} \right) \left(e^{i(\frac{2\pi k t_1}{P} + \frac{2\pi k n}{N})} + e^{-i(\frac{2\pi k t_1}{P} + \frac{2\pi k n}{N})} \right) \\ &= \frac{1}{4} \sum_{n=0}^{N-1} \left(e^{i(C_j + \frac{2\pi j n}{N})} + e^{-i(C_j + \frac{2\pi j n}{N})} \right) \left(e^{i(C_k + \frac{2\pi k n}{N})} + e^{-i(C_k + \frac{2\pi k n}{N})} \right) \\ &= \frac{1}{4} \sum_{n=0}^{N-1} \left(e^{i C_j} e^{i \frac{2\pi j n}{N}} + e^{-i C_j} e^{-i \frac{2\pi j n}{N}} \right) \left(e^{i C_k} e^{i \frac{2\pi k n}{N}} + e^{-i C_k} e^{-i \frac{2\pi k n}{N}} \right) \\ &= \frac{1}{4} e^{i(C_j + C_k)} \overbrace{\sum_{n=0}^{N-1} e^{2\pi(j+k)i(\frac{n}{N})}}^{\text{first sum}} + \frac{1}{4} e^{-i(C_j + C_k)} \overbrace{\sum_{n=0}^{N-1} e^{-2\pi(j+k)i(\frac{n}{N})}}^{\text{second sum}} \\ &\quad + \frac{1}{4} e^{i(C_j - C_k)} \overbrace{\sum_{n=0}^{N-1} e^{2\pi(j-k)i(\frac{n}{N})}}^{\text{third sum}} + \frac{1}{4} e^{-i(C_j - C_k)} \overbrace{\sum_{n=0}^{N-1} e^{-2\pi(j-k)i(\frac{n}{N})}}^{\text{fourth sum}}. \end{aligned} \quad (5)$$

Since j and k take on positive integer values between 1 and K with $j \neq k$, and since $N \geq 2K + 1$, it follows that

$$3 \leq j + k \leq K + (K - 1) = 2K - 1 \leq (N - 1) - 1 = N - 2.$$

In particular, $1 \leq j + k \leq N - 1$, so by Proposition 2, the first and second sums in equation (5) vanish.

Without loss of generality, suppose that $j > k$. Then,

$$1 \leq j - k \leq K - 1 \leq \frac{N - 1}{2} - 1 < N - 1.$$

In particular, $1 \leq j - k \leq N - 1$, so the third and fourth sums also vanish. This completes the proof of (1). The same technique applies, with obvious changes, to prove (2).

To verify (3), drop the restriction that j and k be nonequal, and compute

$$\begin{aligned} & \sum_{n=1}^N \sin \frac{2\pi j t_n}{P} \cos \frac{2\pi k t_n}{P} \\ &= \sum_{n=0}^{N-1} \frac{1}{2i} \left(e^{i(C_j + \frac{2\pi j n}{N})} - e^{-i(C_j + \frac{2\pi j n}{N})} \right) \frac{1}{2} \left(e^{i(C_k + \frac{2\pi k n}{N})} + e^{-i(C_k + \frac{2\pi k n}{N})} \right) \\ &= \underbrace{\frac{1}{4i} e^{i(C_j + C_k)} \sum_{n=0}^{N-1} e^{2\pi(j+k)i(\frac{n}{N})}}_{\text{first sum}} - \underbrace{\frac{1}{4i} e^{-i(C_j + C_k)} \sum_{n=0}^{N-1} e^{-2\pi(j+k)i(\frac{n}{N})}}_{\text{second sum}} \\ &\quad + \underbrace{\frac{1}{4i} e^{i(C_j - C_k)} \sum_{n=0}^{N-1} e^{2\pi(j-k)i(\frac{n}{N})}}_{\text{third sum}} - \underbrace{\frac{1}{4i} e^{-i(C_j - C_k)} \sum_{n=0}^{N-1} e^{-2\pi(j-k)i(\frac{n}{N})}}_{\text{fourth sum}}. \end{aligned} \quad (6)$$

If $j \neq k$, then previous arguments show that all four sums vanish. If $j = k$, then the last two sums cancel, since $C_j = C_k$ and $j - k = 0$. Also, if $j = k$, then $j + k = 2j \leq 2K \leq N - 1$, so the first two sums vanish.

Similar arguments show that equations (4) are true. ■

COROLLARY The restriction $N \geq 2K + 1$ in Theorem 1 is as good as possible; that is, no smaller value of N works. In particular, if $N = 2K$ and $j = k = K$, then the condition

$$\sum_{n=1}^N \sin \frac{2\pi j t_n}{P} \cos \frac{2\pi k t_n}{P} = 0, \quad j, k = 1, \dots, K$$

holds if and only if $\frac{4Kt_1}{P}$ is an integer.

PROOF

It is first shown that when $N = 2K$, Theorem 1 is not true. Let $K = 2$, $N = 4$, $P = 2\pi$, and $(t_1, t_2, t_3, t_4) = (\frac{\pi}{8}, \frac{5\pi}{8}, \frac{9\pi}{8}, \frac{13\pi}{8})$. Let $j = k = 2$. Then,

$$\begin{aligned} \sum_{n=1}^N \sin \frac{2\pi j t_n}{P} \cos \frac{2\pi k t_n}{P} &= \sum_{n=1}^4 \sin 2t_n \cos 2t_n \\ &= \sin \frac{\pi}{4} \cos \frac{\pi}{4} + \sin \frac{5\pi}{4} \cos \frac{5\pi}{4} \\ &\quad + \sin \frac{9\pi}{4} \cos \frac{9\pi}{4} + \sin \frac{13\pi}{4} \cos \frac{13\pi}{4} \\ &= 4\left(\frac{1}{2}\right) \neq 0. \end{aligned}$$

Next, it is shown that Theorem 1 holds with $N \geq 2K+1$ replaced by the less restrictive requirement $N \geq 2K$, if and only if $\frac{4Kt_1}{P}$ is an integer. It is only necessary to consider the case $N = 2K$.

When $N = 2K$ and $j \neq k$, then

$$3 \leq j+k \leq K+(K-1) = 2K-1 = N-1$$

and, for $j > k$,

$$1 \leq j-k \leq K-1 = \frac{N}{2} - 1 < N-1.$$

Thus, equations (1) and (2) from Theorem 1 still hold. Also, equation (3) holds whenever $j \neq k$.

Suppose that $j = k$, and consider equation (6) from the proof of Theorem 1. The third and fourth sums cancel. If $j < K$, then $j+k < 2K$, so that $j+k \leq N-1$, and the first and second sums vanish. If $j = k = K$, then the first and second sums become

$$\begin{aligned} \frac{1}{4i} e^{i(C_j+C_k)} \sum_{n=0}^{N-1} e^{2\pi(j+k)i(\frac{n}{N})} - \frac{1}{4i} e^{-i(C_j+C_k)} \sum_{n=0}^{N-1} e^{-2\pi(j+k)i(\frac{n}{N})} \\ = \frac{1}{4i} e^{2iC_K} \sum_{n=0}^{N-1} e^{4\pi K i(\frac{n}{2K})} - \frac{1}{4i} e^{-2iC_K} \sum_{n=0}^{N-1} e^{-4\pi K i(\frac{n}{2K})} \\ = \frac{1}{4i} e^{2iC_K} \sum_{n=0}^{N-1} e^{2\pi i n} - \frac{1}{4i} e^{-2iC_K} \sum_{n=0}^{N-1} e^{-2\pi i n} \\ = \frac{N}{4i} (e^{2iC_K} - e^{-2iC_K}) \\ = \frac{N}{4i} (2i \sin 2C_K). \end{aligned} \tag{7}$$

Thus, (7) equals zero if and only if $\sin 2C_K$ equals zero, if and only if $2C_K$ is an integer multiple of π . That is, it must be that

$$2C_K := 2 \left(\frac{2\pi K t_1}{P} \right) = \pi j$$

for some integer j ; that is, $\frac{4Kt_1}{P}$ must be an integer. Note that if $\frac{t_1}{P}$ is an integer, or if $t_1 = 0$, then equation (7) equals zero.

In the counterexample given above, $\frac{4Kt_1}{P} = \frac{4(2)(\pi/8)}{2\pi} = \frac{1}{2}$ is not an integer. ■

COROLLARY Let the conditions of Theorem 1 hold. Then

$$\sum_{n=1}^N \cos^2 \frac{2\pi k t_n}{P} = \frac{N}{2}, \quad k = 1, \dots, K, \quad \text{and}$$

$$\sum_{n=1}^N \sin^2 \frac{2\pi k t_n}{P} = \frac{N}{2}, \quad k = 1, \dots, K.$$

PROOF

Let $k = 1, \dots, K$. Using (5) from Theorem 1 with $j = k$, one obtains

$$\begin{aligned} \sum_{n=1}^N \cos^2 \frac{2\pi k t_n}{P} &= \frac{1}{4} e^{i2C_k} \sum_{n=0}^{N-1} e^{2\pi(2k)i(\frac{n}{N})} + \frac{1}{4} e^{-i2C_k} \sum_{n=0}^{N-1} e^{-2\pi(2k)i(\frac{n}{N})} \\ &\quad + \frac{1}{4}(1) \sum_{n=0}^{N-1} (1) + \frac{1}{4}(1) \sum_{n=0}^{N-1} (1). \end{aligned}$$

Since $2k \leq 2K \leq N-1$, Proposition 2 (Section 1.6) implies that the first and second sums vanish, and thus

$$\sum_{n=1}^N \cos^2 \frac{2\pi k t_n}{P} = \frac{N}{4} + \frac{N}{4} = \frac{N}{2}.$$

A similar procedure verifies the second result. ■

THEOREM 2 Let P be a positive real number, and let N and K be positive integers with $N \geq 2K + 1$. Let $\{(t_i, y_i)\}_{i=1}^N$ be a data set, where the time list (t_1, \dots, t_N) is uniform with $\Delta T = \frac{P}{N}$. Define

$$f(t) := a_0 + \sum_{k=1}^K \left(a_k \cos \frac{2\pi kt}{P} + b_k \sin \frac{2\pi kt}{P} \right).$$

Then, the least-squares solution to

$$\min_{a_k, b_k \in \mathbb{R}} \sum_{n=1}^N (y_i - f(t_i))^2$$

is given by

$$\begin{aligned} a_0 &= \frac{1}{N} \sum_{n=1}^N y_n, \\ a_k &= \frac{2}{N} \sum_{n=1}^N y_n \cos \frac{2\pi kt_n}{P}, \quad k = 1, \dots, K, \quad \text{and} \\ b_k &= \frac{2}{N} \sum_{n=1}^N y_n \sin \frac{2\pi kt_n}{P}, \quad k = 1, 2, \dots, K. \end{aligned}$$

PROOF
of Theorem 2

Recalling results from Sections 2.2 and 2.3, the least-squares solution $\mathbf{b} = (a_0, a_1, \dots, a_K, b_1, \dots, b_K)$ is given by $\mathbf{b} = (\mathbf{X}^t \mathbf{X})^{-1} \mathbf{X}^t \mathbf{y}$, where \mathbf{y} is the column vector of data values,

$$f_1(t) \equiv 1,$$

$$f_{k+1}(t) = \cos \frac{2\pi kt}{P}, \quad k = 1, \dots, K,$$

$$f_{k+(K+1)}(t) = \sin \frac{2\pi kt}{P}, \quad k = 1, \dots, K,$$

$$\mathbf{X}_{ij} = f_j(t_i), \quad 1 \leq i \leq N, \quad 1 \leq j \leq 2K + 1,$$

and

$$(\mathbf{X}^t \mathbf{X})_{ij} = \sum_{n=1}^N f_i(t_n) f_j(t_n), \quad 1 \leq i, j \leq 2K + 1.$$

By Theorem 1, the functions $\{f_k\}_{k=1}^{2K+1}$ are mutually orthogonal with respect to the given time values, so the matrix $X^t X$ is diagonal, with

$$\begin{aligned}(X^t X)_{11} &= N, \\(X^t X)_{k+1,k+1} &= \sum_{n=1}^N \cos^2 \frac{2\pi k t_n}{P} = \frac{N}{2}, \quad k = 1, \dots, K, \quad \text{and} \\(X^t X)_{k+K+1,k+K+1} &= \sum_{n=1}^N \sin^2 \frac{2\pi k t_n}{P} = \frac{N}{2}, \quad k = 1, \dots, K.\end{aligned}$$

Recall that the inverse of a diagonal matrix is easily found by taking the reciprocals of each diagonal entry. Thus,

$$((X^t X)^{-1})_{kk} = \begin{cases} \frac{1}{N} & k = 1 \\ \frac{2}{N} & k = 2, \dots, 2K + 1. \end{cases}$$

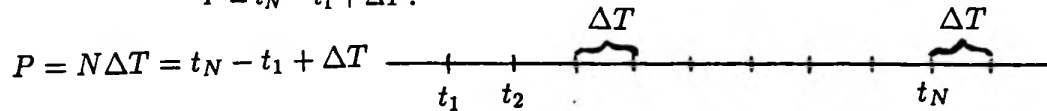
The matrix product b is now computed:

$$b = (X^t X)^{-1} X^t y$$

$$\begin{aligned}&= \begin{bmatrix} \frac{1}{N} & 0 & 0 & \cdots & 0 \\ 0 & \frac{2}{N} & 0 & \cdots & 0 \\ 0 & 0 & \frac{2}{N} & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & \frac{2}{N} \end{bmatrix} \begin{bmatrix} f_1(t_1) & f_1(t_2) & f_1(t_3) & \cdots & f_1(t_N) \\ f_2(t_1) & f_2(t_2) & f_2(t_3) & \cdots & f_2(t_N) \\ f_3(t_1) & f_3(t_2) & f_3(t_3) & \cdots & f_3(t_N) \\ \vdots & \vdots & \vdots & \cdots & \vdots \\ f_{2K+1}(t_1) & f_{2K+1}(t_2) & f_{2K+1}(t_3) & \cdots & f_{2K+1}(t_N) \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_N \end{bmatrix} \\&= \begin{bmatrix} \frac{1}{N} f_1(t_1) & \frac{1}{N} f_1(t_2) & \frac{1}{N} f_1(t_3) & \cdots & \frac{1}{N} f_1(t_N) \\ \frac{2}{N} f_2(t_1) & \frac{2}{N} f_2(t_2) & \frac{2}{N} f_2(t_3) & \cdots & \frac{2}{N} f_2(t_N) \\ \frac{2}{N} f_3(t_1) & \frac{2}{N} f_3(t_2) & \frac{2}{N} f_3(t_3) & \cdots & \frac{2}{N} f_3(t_N) \\ \vdots & \vdots & \vdots & \cdots & \vdots \\ \frac{2}{N} f_{2K+1}(t_1) & \frac{2}{N} f_{2K+1}(t_2) & \frac{2}{N} f_{2K+1}(t_3) & \cdots & \frac{2}{N} f_{2K+1}(t_N) \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_N \end{bmatrix} \\&= \begin{bmatrix} \frac{1}{N} & \frac{1}{N} & \frac{1}{N} & \cdots & \frac{1}{N} \\ \frac{2}{N} \cos \frac{2\pi t_1}{P} & \frac{2}{N} \cos \frac{2\pi t_2}{P} & \frac{2}{N} \cos \frac{2\pi t_3}{P} & \cdots & \frac{2}{N} \cos \frac{2\pi t_N}{P} \\ \frac{2}{N} \cos \frac{2\pi 2t_1}{P} & \frac{2}{N} \cos \frac{2\pi 2t_2}{P} & \frac{2}{N} \cos \frac{2\pi 2t_3}{P} & \cdots & \frac{2}{N} \cos \frac{2\pi 2t_N}{P} \\ \vdots & \vdots & \vdots & \cdots & \vdots \\ \frac{2}{N} \sin \frac{2\pi K t_1}{P} & \frac{2}{N} \sin \frac{2\pi K t_2}{P} & \frac{2}{N} \sin \frac{2\pi K t_3}{P} & \cdots & \frac{2}{N} \sin \frac{2\pi K t_N}{P} \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_N \end{bmatrix}\end{aligned}$$

It follows readily that the coefficients have the desired form. ■

Let N be a positive integer, and suppose that a data set $\{(t_i, y_i)\}_{i=1}^N$ has a time list (t_1, \dots, t_N) that is uniform with positive increment ΔT . By defining $P := N\Delta T$, one has $\frac{P}{N} = \Delta T$, so that P meets the requirements of Theorems 1 and 2. Note also that $P = t_N - t_1 + \Delta T$.



Any value of K that is less than or equal to $\frac{N-1}{2}$ will meet the requirements of Theorems 1 and 2. In the next definition, K is usually taken to be as large as possible, subject to the condition $K \leq \frac{N-1}{2}$.

DEFINITION
discrete
Fourier series
(DFS);
periodogram

Let N be a positive integer, and let $\{(t_i, y_i)\}_{i=1}^N$ be a data set with a time list (t_1, \dots, t_N) that is uniform with positive increment ΔT . Define $P := N\Delta T$. Let K be any positive integer less than or equal to $\frac{N-1}{2}$.

The *discrete Fourier series* corresponding to the data set and the chosen value K is the function DFS given by

$$\text{DFS}(t) := a_0 + \sum_{k=1}^K \left(a_k \cos \frac{2\pi kt}{P} + b_k \sin \frac{2\pi kt}{P} \right),$$

where the coefficients $a_0, a_1, \dots, a_K, b_1, \dots, b_K$ are given by

$$\begin{aligned} a_0 &= \frac{1}{N} \sum_{n=1}^N y_n, \\ a_k &= \frac{2}{N} \sum_{n=1}^N y_n \cos \frac{2\pi kt_n}{P}, \quad k = 1, \dots, K, \quad \text{and} \\ b_k &= \frac{2}{N} \sum_{n=1}^N y_n \sin \frac{2\pi kt_n}{P}, \quad k = 1, 2, \dots, K. \end{aligned}$$

The graph of the points

$$\left\{ \left(\frac{P}{k}, \frac{N}{2} \sqrt{a_k^2 + b_k^2} \right) \mid k = 1, \dots, K \right\}$$

is the *periodogram* corresponding to the discrete Fourier series.

The reason for the factor $\frac{N}{2}$ in the definition of the *periodogram* will become apparent in the next section, on the discrete Fourier transform.

*properties of
the DFS*

The periods of the sinusoids appearing in the discrete Fourier series are

$$P, \frac{P}{2}, \frac{P}{3}, \dots, \frac{P}{K};$$

the coefficients a_k and b_k multiply the sinusoids with period $\frac{P}{k}$. The coefficient a_0 gives the average of the data values. If $N = 2K + 1$, then the discrete Fourier series passes through all the data points; and if $N > 2K + 1$, then the series minimizes the mean-square error.

The periodogram gives information about 'how much' of each period is present in the data set, as the examples later on in this section will illustrate.

*comparing the
discrete and
continuous
coefficients*

Observe that if one takes the formula for the *continuous* Fourier series coefficient a_k (see p. 81),

$$a_k = \frac{2}{P} \int_0^P g(t) \cos \frac{2\pi kt}{P} dt,$$

replaces the \int_0^P by $\sum_{n=1}^N$, dt by the increment $\frac{P}{N}$, t by t_n , and $g(t)$ by y_n , one obtains

$$a_k = \frac{2}{P} \sum_{n=1}^N y_n \left(\cos \frac{2\pi kt_n}{P} \right) \cdot \frac{P}{N} = \frac{2}{N} \sum_{n=1}^N y_n \cos \frac{2\pi kt_n}{P},$$

which is precisely the discrete Fourier coefficient a_k . So there is a beautiful (and not unexpected) analogy between the continuous and discrete results.

MATLAB IMPLEMENTATION

Discrete Fourier Series and the Periodogram

**MATLAB
FUNCTION**
dfs(D,K)

The following MATLAB function computes the discrete Fourier series and periodogram corresponding to a data set. To use the program, type:

```
[per,sqrcoef,C,DFS] = dfs(D,K);
```


<i>required input, D</i>	<p>The required input is a data set $\{(t_i, y_i)\}_{i=1}^N$, with $N \geq 3$. The time values must be stored in a column vector \mathbf{t} and the corresponding data values in a column vector \mathbf{y}. Then, $\mathbf{D} = [\mathbf{t} \ \mathbf{y}]$ is the $N \times 2$ matrix containing the data set. The time list contained in \mathbf{t} must be uniform (increment $\Delta T > 0$). The program begins by checking that this requirement is met; if not, the program is halted and the message 'not a uniform time list' is displayed. (As written, increments between time values are said to 'differ' if they differ by more than 0.0000001. This value may be changed for different tolerances.)</p>
<i>optional input, K</i>	<p>The number K must satisfy $N \geq 2K + 1$, that is, $K \leq \frac{N-1}{2}$. If no value of K is supplied, then the program uses the greatest possible K satisfying the stated requirement.</p>
<i>outputs: per sqrcoef</i>	<p>The output <i>per</i> is a column vector containing the periods $P, \frac{P}{2}, \frac{P}{3}, \dots, \frac{P}{K}$, where $P = N\Delta T$.</p> <p>The output <i>sqrcoef</i> (for 'square root ... coefficients') is a column vector containing the numbers $\frac{N}{2}\sqrt{a_k^2 + b_k^2}$ for $k = 1, \dots, K$.</p> <p>The periodogram is then obtained with the command:</p> <pre>plot(per,sqrcoef)</pre> <p>If only the outputs <i>per</i> and <i>sqrcoef</i> are desired, the shorter command</p> <pre>[per,sqrcoef] = dfs(D);</pre> <p>or</p> <pre>[per,sqrcoef] = dfs(D,K);</pre> <p>may be used.</p>
<i>outputs: c DFS</i>	<p>The (optional) output <i>c</i> is a column vector containing the coefficients $a_0, a_1, b_1, a_2, b_2, \dots, a_K, b_K$ (in the order given).</p> <p>The (optional) output <i>DFS</i> is a column vector containing the values $\text{DFS}(t_n)$ for $n = 1, \dots, N$.</p>
<i>source code</i>	<p>The source code for the function <i>dfs</i> is given next.</p>

```

% copyright 1993 Carol J.V. Fisher
function [per,sqrcoef,C,DFS] = dfs(D,K)
% This function computes the discrete Fourier series and periodogram
%   for a data set D = [t y] ,
%   where the time values form a uniform time list.
% The number of data points (rows in D) must be greater than or equal to 3.
% K is an (optional) positive integer; if no value of K is specified
% then the greatest integer less than or equal to (N-1)/2 is used.
t = D(:,1);
% First, check that the time list is uniform:
d = diff(t);
% entries that differ by more than .0000001 are called 'different'
p = ( abs(ones(d)*d(1) - d) > .0000001);
err = find(p);
if max(err) ~= 0
    'not a uniform time list'
    return
end
N = length(t);
dT = t(2) - t(1);
P = N*dT;
y = D(:,2);
% 'nargin' is the 'number of arguments into the function'
if (nargin == 1)
    K = floor( (N-1)/2 );
end
% Initialize the matrices
per = zeros(K,1);
sqrcoef = zeros(K,1);
% the matrix C is of the form:
% a_0  0
% a_1  b_1
% a_2  b_2
% .....
% a_K  b_K
C = zeros(K+1,2);
DFS = zeros(1:N)';
C(1,1) = (1/N)*sum(y);
for k = 2:(K+1)
    tvec = (2*pi*(k-1)/P)*t;
    cosv = cos(tvec);
    sinv = sin(tvec);
    C(k,1) = (2/N)*sum(y .* cosv);
    C(k,2) = (2/N)*sum(y .* sinv);
    per(k-1) = P/(k-1);
    sqrcoef(k-1) = sqrt( (C(k,1))^2 + (C(k,2))^2 );
    DFS = DFS + C(k,1)*cosv + C(k,2)*sinv;
end
sqrcoef = (N/2)*sqrcoef;
% Here's the Discrete Fourier Series corresponding to the data:
DFS = C(1,1) + DFS;

```

MATLAB EXAMPLE

*sinusoidal
components,
periods in the
DFS*

In this first example, a data set is constructed having sinusoidal components of periods 25 and 10. Fifty data points are used, with $\Delta T = 1$, so that $P = (50)(1) = 50$. The largest allowable value of K is used: since $K \leq \frac{50-1}{2}$, the greatest such K equals 24. Observe that the periods $\frac{P}{5} = 10$ and $\frac{P}{2} = 25$ appear in the discrete Fourier series.

First, the 'pure' data (no noise) is analyzed. Of course, the discrete Fourier series 'recovers' the components perfectly in this case.

Then, some noise is added to the data. In this case, the periodogram clearly peaks at periods 10 and 25. However, the noise has contributed some small amplitude, high frequency (small period) components in the discrete Fourier series.

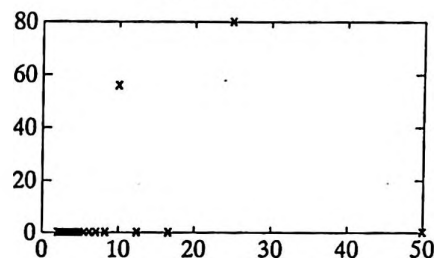
```
t = [1:50]';
y = sin(2*pi*t/10) - 2*cos(2*pi*t/10) + 3.2*cos(2*pi*t/25);
D = [t y];
[per,sqrcoef,C,DFS] = dfs(D);
% plot the periodogram
plot(per,sqrcoef,'x')
% plot the data set, together with its discrete Fourier series
plot(t,y,'x')
hold
```

Current plot held

```
plot(t,DFS,'o')
```

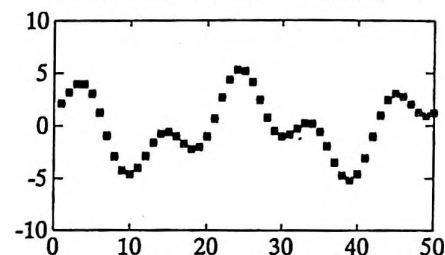
per =	C =	
	-0.0000	0
50.0000	-0.0000	0.0000
25.0000	3.2000	0.0000
16.6667	-0.0000	0.0000
12.5000	0.0000	0.0000
10.0000	-2.0000	1.0000
8.3333	-0.0000	-0.0000
7.1429	0.0000	-0.0000
6.2500	0.0000	-0.0000
5.5556	-0.0000	0.0000
5.0000	0.0000	-0.0000
4.5455	-0.0000	0.0000
4.1667	-0.0000	0.0000
3.8462	-0.0000	-0.0000
3.5714	-0.0000	-0.0000
3.3333	0.0000	-0.0000
3.1250	0.0000	0.0000
2.9412	0.0000	0.0000
2.7778	0.0000	0.0000
2.6316	0.0000	0.0000
2.5000	0.0000	-0.0000
2.3810	0.0000	0.0000
2.2727	-0.0000	0.0000
2.1739	-0.0000	-0.0000
2.0833	0.0000	-0.0000

PERIODOGRAM



DATA SET, 'x'

DISCRETE FOURIER SERIES, 'o'



Note that the data set is indistinguishable from its discrete Fourier series, as apparent by the overlapping of the symbols 'x' and 'o'.

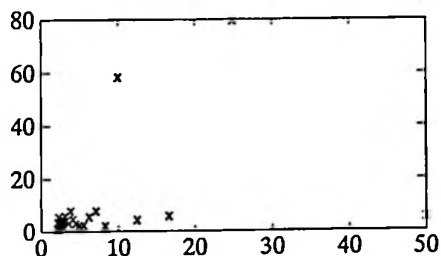
```
% Now, add some noise to the previous data, and repeat
noise = 2*(rand(t) - .5);
y = y + noise;
D = [t y];
[per,sqrcoef,C,DFS] = dfs(D);
plot(per,sqrcoef,'x')
plot(t,y,'x')
hold
```

Current plot held

```
plot(t,DFS,'o')
```

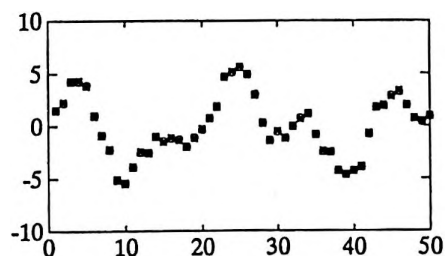
per =	C =	
	0.0185	0
50.0000	-0.1248	-0.1455
25.0000	3.1817	-0.0103
16.6667	-0.0528	0.2176
12.5000	-0.1608	-0.0117
10.0000	-2.1438	0.9064
8.3333	-0.0281	0.0704
7.1429	-0.2543	-0.1660
6.2500	0.2033	0.0691
5.5556	0.0202	-0.0864
5.0000	-0.0811	-0.0348
4.5455	-0.1093	0.0002
4.1667	0.1705	0.0379
3.8462	0.2987	-0.0256
3.5714	-0.0043	0.1229
3.3333	0.1642	0.0214
3.1250	-0.1701	-0.1589
2.9412	-0.0757	-0.1485
2.7778	0.0394	0.0877
2.6316	0.0665	0.1179
2.5000	0.0927	-0.0087
2.3810	-0.0240	-0.0212
2.2727	0.0263	0.2064
2.1739	-0.1148	-0.0642
2.0833	-0.0276	0.0055

PERIODOGRAM



DATA SET, 'x'

DISCRETE FOURIER SERIES, 'o'



MATLAB EXAMPLE

*sinusoidal
components,
periods are
NOT in the DFS*

In this second example, a data set is constructed having sinusoidal components of periods 15 and 32. Fifty data points are used, with $\Delta T = 1$, so that again $P = (50)(1) = 50$. Again, the largest allowable value of K is used; $K = 24$. Observe that the periods of the sinusoids in the data are *not* in the discrete Fourier series.

First, the 'pure' data (no noise) is analyzed. The periodogram 'peaks' at the period $\frac{50}{3} = 16\frac{1}{3}$, which is close to one of the 'hidden' periods, 15. The period 32 component is not easily detectable from the periodogram.

Then, some noise is added to the data. The situation is similar to that described above, but with a little more 'fuzz'.

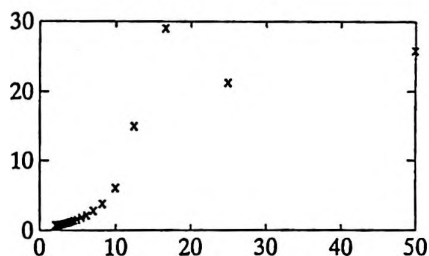
```
t = [1:50]';
y = sin(2*pi*t/15) - cos(2*pi*t/15) + 1.4*sin(2*pi*t/32);
D = [t y];
[per,sqrcoef,C,DFS] = dfs(D);
plot(per,sqrcoef,'x')
plot(t,y,'x')
hold
```

Current plot held

```
plot(t,DFS,'o')
```

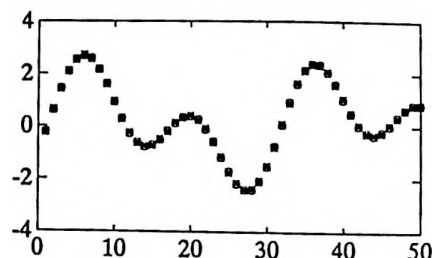
per =	C =	
	0.3216	0
50.0000	1.0299	-0.0431
25.0000	-0.7310	0.4322
16.6667	0.1483	1.1506
12.5000	-0.2025	-0.5628
10.0000	-0.0739	-0.2294
8.3333	-0.0331	-0.1464
7.1429	-0.0127	-0.1080
6.2500	-0.0007	-0.0854
5.5556	0.0071	-0.0703
5.0000	0.0124	-0.0593
4.5455	0.0163	-0.0507
4.1667	0.0191	-0.0439
3.8462	0.0213	-0.0381
3.5714	0.0229	-0.0332
3.3333	0.0242	-0.0289
3.1250	0.0253	-0.0251
2.9412	0.0261	-0.0216
2.7778	0.0268	-0.0184
2.6316	0.0273	-0.0154
2.5000	0.0277	-0.0126
2.3810	0.0281	-0.0100
2.2727	0.0283	-0.0074
2.1739	0.0285	-0.0049
2.0833	0.0286	-0.0024

PERIODOGRAM



DATA SET, 'x'

DISCRETE FOURIER SERIES, 'o'



```

noise = 1.5*(rand(t) - .5);
y = y + noise;
D = [t y];
[per,sqrcoef,C,DFS] = dfs(D);
plot(per,sqrcoef,'x')
plot(t,y,'x')
hold

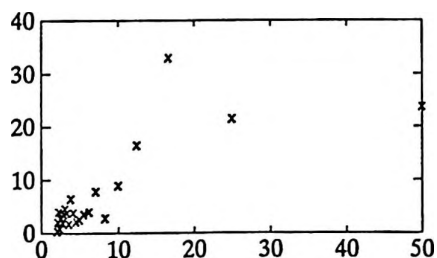
```

Current plot held

```
plot(t,DFS,'o')
```

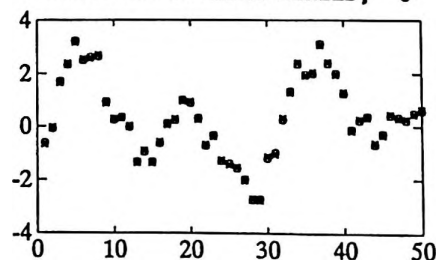
per =	C =	
	0.3355	0
50.0000	0.9363	-0.1522
25.0000	-0.7447	0.4245
16.6667	0.1087	1.3138
12.5000	-0.3230	-0.5716
10.0000	-0.1817	-0.2996
8.3333	-0.0542	-0.0936
7.1429	-0.2034	-0.2325
6.2500	0.1518	-0.0336
5.5556	0.0223	-0.1351
5.0000	-0.0484	-0.0853
4.5455	-0.0657	-0.0506
4.1667	0.1470	-0.0154
3.8462	0.2453	-0.0574
3.5714	0.0197	0.0590
3.3333	0.1474	-0.0129
3.1250	-0.1023	-0.1442
2.9412	-0.0307	-0.1330
2.7778	0.0563	0.0474
2.6316	0.0772	0.0730
2.5000	0.0973	-0.0192
2.3810	0.0101	-0.0259
2.2727	0.0480	0.1474
2.1739	-0.0576	-0.0531
2.0833	0.0078	0.0017

PERIODOGRAM



DATA SET, 'x'

DISCRETE FOURIER SERIES, 'o'

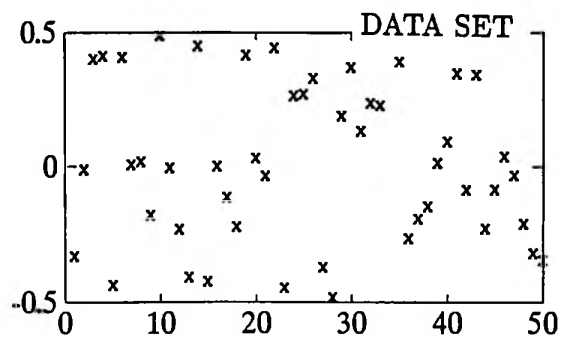
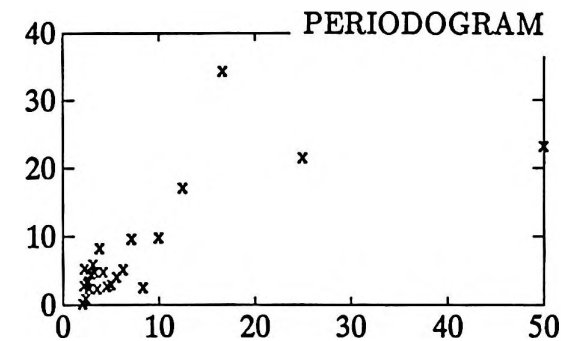


**MATLAB
EXAMPLE**

*periodogram
of noise*

'Noise' is usually associated with high frequency (small period) components. In this example, the periodogram corresponding to some noise (from a uniform distribution) is found:

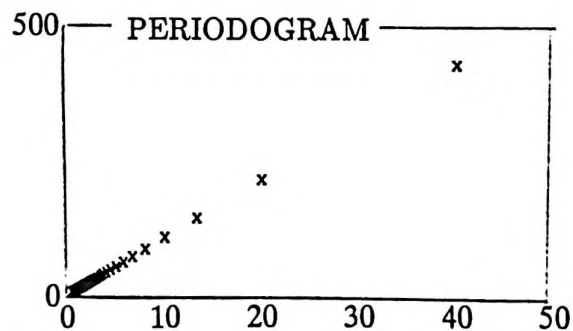
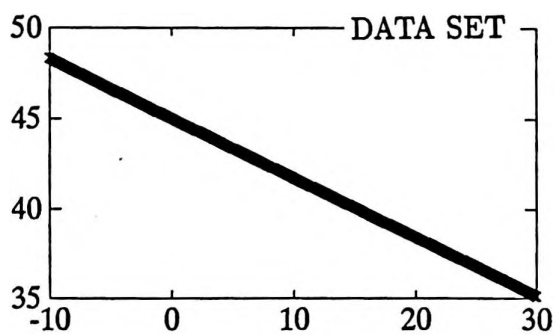
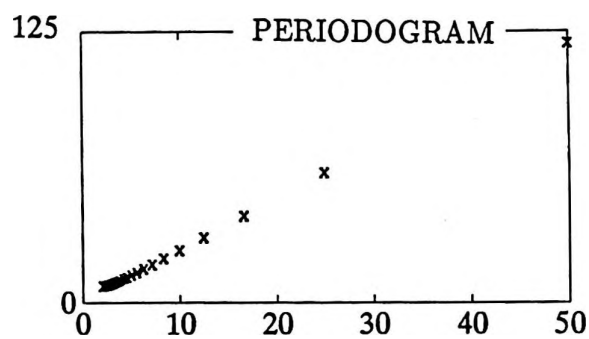
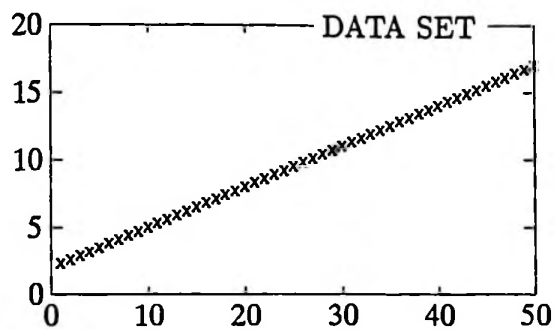
```
t = [1:50]';
noise = rand(t) - .5;
D = [t noise];
[per,sqrcoef] = dfs(D);
subplot(221)
plot(per,sqrcoef,'x')
plot(t,noise,'x')
```



**MATLAB
EXAMPLE**

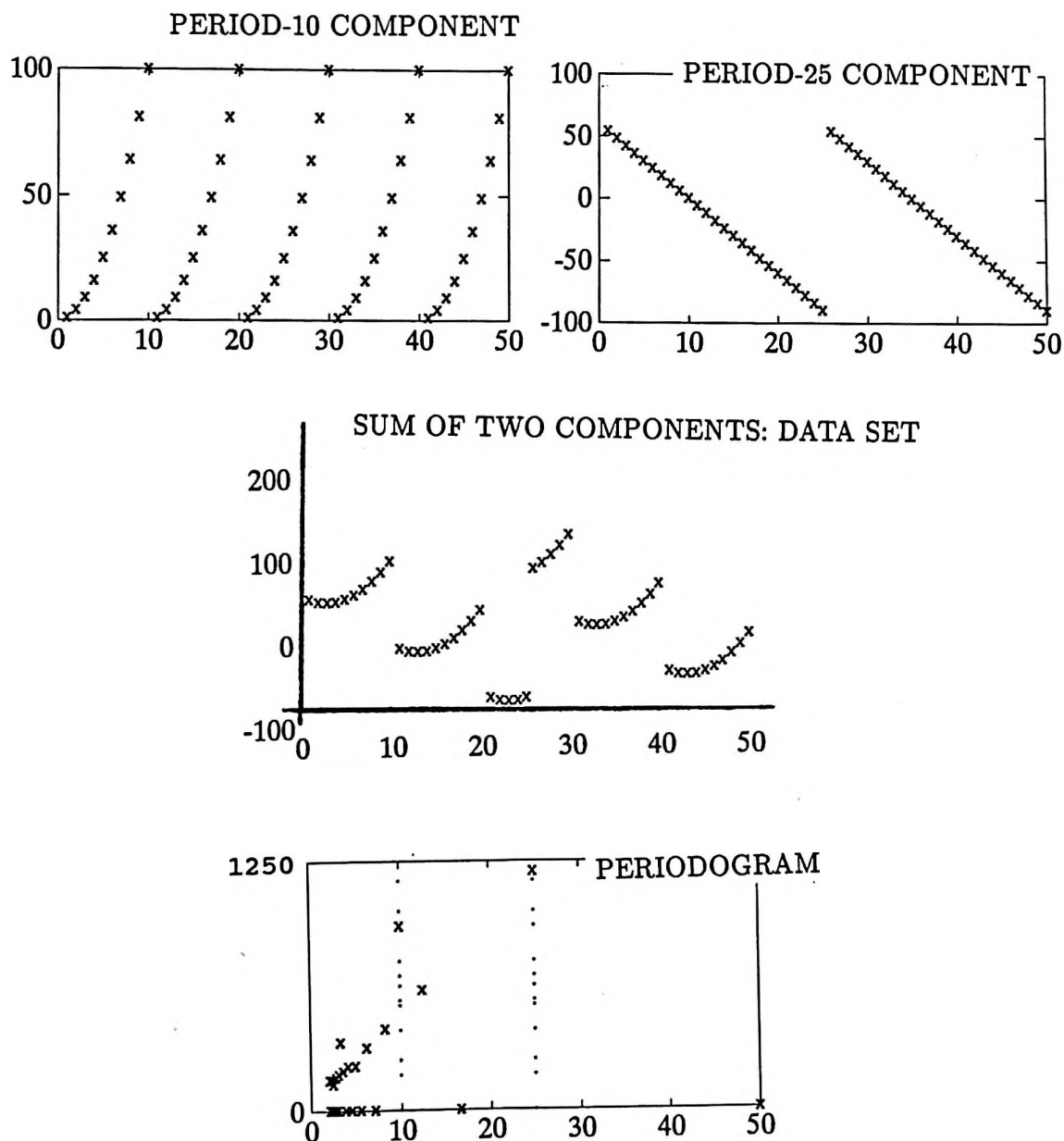
*periodogram of
a 'ramp'*

The periodograms of a couple 'ramps' (linear functions) are found next:



**MATLAB
EXAMPLE**
*non-sinusoidal
components*

Finally, the periodogram of a data set with two non-sinusoidal components, periods 10 and 25, are found. Here, $N\Delta T = (50)(1)$, so these periods $\frac{50}{5} = 10$ and $\frac{50}{2} = 25$ appear in the discrete Fourier series. The vertical 'dotted' lines on the periodogram indicate the periods 10 and 25. Observe that the periodogram does appear to 'peak' at these values.



2.7 The Periodogram, via the Discrete Fourier Transform

*discrete
Fourier
transform
(DFT)*

The *discrete Fourier transform* (DFT) is a tool used to determine the spectral (frequency) content of equally-sampled data. Since the *periodogram* (Section 2.6) also gives information about frequency content, one might suspect that there is a relationship between the periodogram and the DFT. Such is indeed the case. In this section, the discrete Fourier transform is defined, and the relationship between the DFT and the periodogram is explored.

*fast Fourier
transform
(FFT);
an efficient
implementation
of the DFT*

As will be seen, there is a great deal of redundancy inherent in the definition of the discrete Fourier transform. Elimination of this redundancy has produced a class of *implementations* of the DFT that have come to be called *fast Fourier transforms* (FFTs). By using the built-in MATLAB fast Fourier transform command, and exploiting the relationship between the DFT and the periodogram, one obtains a much more efficient way to find the periodogram corresponding to a data set. A 'time comparison' is included at the end of this section.

*motivation for
the definition of
the discrete
Fourier transform*

Let N be a positive integer, and let $\{(t_i, y_i)\}_{i=1}^N$ be a data set with a time list (t_1, \dots, t_N) that is uniform with positive increment ΔT . The *discrete Fourier transform* is defined so that it is independent of the starting time t_1 and increment ΔT , as follows. First, translate the list $(t_1, t_2, \dots, t_n, \dots, t_N)$ to the list $(0, \Delta T, \dots, (n-1)\Delta T, \dots, (N-1)\Delta T)$, via the transformation

$$t_n \mapsto (t_n - t_1), \quad n = 1, \dots, N.$$

Next, require that $P := N\Delta T$, as in Section 2.6. Under these conditions, note that

$$e^{-i\frac{2\pi m t_n}{P}} = e^{-i\frac{2\pi m(n-1)\Delta T}{N\Delta T}} = e^{-i\frac{2\pi m(n-1)}{N}}.$$

With these comments in mind, observe that the next definition requires knowledge only of the data values y_n , and the fact that these values did arise from a uniform time list:

DEFINITION
discrete Fourier
transform
(DFT)

Let (y_1, \dots, y_N) be N data values from a data set $\{(t_i, y_i)\}_{i=1}^N$ with a uniform time list. Then,

$$\text{DFT}(m) := \sum_{n=1}^N y_n e^{-i \frac{2\pi m(n-1)}{N}}, \quad m = 0, \dots, N-1,$$

are the N independent values of the *discrete Fourier transform* of the data set.

illustrating the
redundancy in the
DFT when $N = 8$

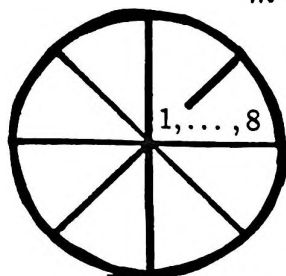
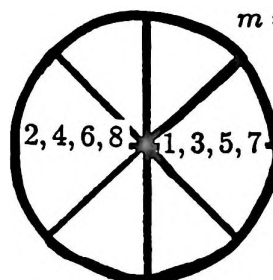
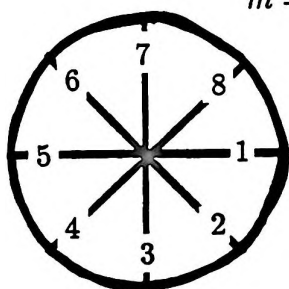
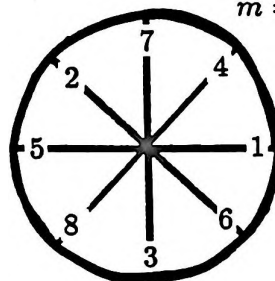
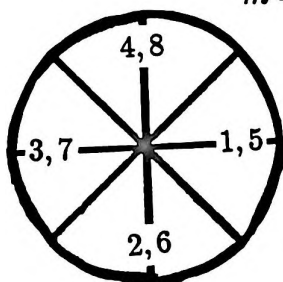
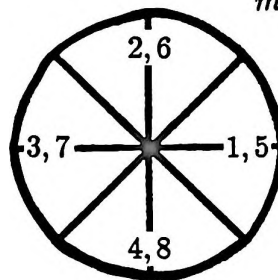
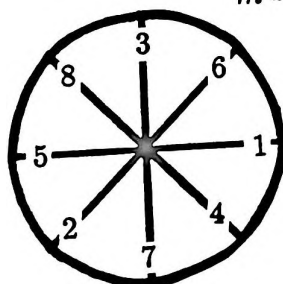
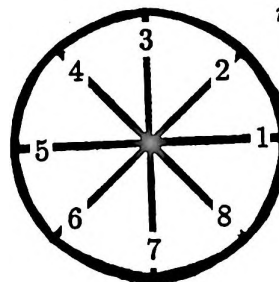
To illustrate the redundancy in the DFT, consider its computation when $N = 8$. How many complex products, each of the form $y_n e^{-i \frac{2\pi m(n-1)}{N}}$, must be computed? For a fixed value of m , there are N such products, as n varies from 1 to N . Also, there are N values of m . Thus, at first glance, it appears that $8 \cdot 8$ products must be computed, when $N = 8$. However, it will next be shown that not all 64 of these products are distinct.

When $N = 8$, every exponent that appears in the DFT has a denominator of 8, so the appropriate picture is of the unit circle in the complex plane, with 2π radians divided into 8 equal pieces. Refer to the next figure as you continue reading. The sketches there illustrate the computations for the entire DFT. The numbers at different angles *inside* the unit circle indicate the subscript n of the data value y_n that must multiply the exponential at that particular angle.

When $m = 0$, no 'genuine' complex products are involved:
 $\text{DFT}(0) = y_1 + \dots + y_N$.

When $m = 1$, the exponents $-2\pi(1)(n-1)/8$ in the exponential cause movement backward around the unit circle, one step at a time, as n goes from 1 to N . For example, y_4 must multiply the exponential at an angle of $-2\pi(1)(4-1)/8 = -3\pi/4$.

When $m = 2$, one moves backward around the unit circle, *two steps at a time*, so that, e.g., now y_4 must multiply the exponential at an angle of $-2\pi(2)(4-1)/8 = -3\pi/2$.

$m = 0$  $m = 4$  $m = 1$  $m = 5$  $m = 2$  $m = 6$  $m = 3$  $m = 7$ 

*How much
redundancy?*

Now, the number of *distinct* complex products that must be computed can be counted. The products at angles 0 and π are not truly complex, since $e^{0i} = 1$ and $e^{\pi i} = -1$; so these will not be counted as complex products.

At $\theta = 2\pi(1)/8$, there are 4 distinct complex products ($n = 2, 4, 6, 8$).

At $\theta = 2\pi(2)/8$, there are 6 ($n = 2, 3, 4, 6, 7, 8$).

At $\theta = 2\pi(3)/8$, there are 4 ($n = 2, 4, 6, 8$).

At $\theta = 2\pi(5)/8$, there are 4 ($n = 2, 4, 6, 8$).

At $\theta = 2\pi(6)/8$, there are 6 ($n = 2, 3, 4, 6, 7, 8$).

At $\theta = 2\pi(7)/8$, there are 4 ($n = 2, 4, 6, 8$).

Thus, computation of the entire DFT actually requires only $4 + 6 + 4 + 4 + 6 + 4 = 28$ distinct complex products.

The MATLAB command `fft` is an efficient computation of the DFT, that (at least partially) eliminates the redundancy illustrated here.

**uniform
hypotheses
for the following
Propositions**

The next few propositions develop the relationship between the DFT and the periodogram (Section 2.6). In these propositions, the following situation is assumed to hold:

- N is a positive integer.
- $\{(t_i, y_i)\}_{i=1}^N$ is a data set with a time list (t_1, \dots, t_N) that is uniform with positive increment ΔT . All values of t_i and y_i are real numbers.
- K is the largest integer satisfying $N \geq 2K + 1$.
If N is odd, then $N = 2K + 1$, so that $K = \frac{N-1}{2}$.
If N is even, then $N = 2K + 2$, so that $K = \frac{N-2}{2}$.
- The numbers $a_0, a_1, b_1, \dots, a_K, b_K$ are the coefficients of the discrete Fourier series corresponding to the data set (see p. 222).
- The numbers $\text{DFT}(m)$, for $m = 0, \dots, N-1$, are the N independent values of the discrete Fourier transform.

PROPOSITION

$$\begin{aligned} \text{DFT}(0) &= N \cdot a_0, \text{ and} \\ \text{DFT}(m) &= e^{i \frac{2\pi m t_1}{P}} \cdot \frac{N}{2} (a_m - i b_m), \text{ for } m = 1, \dots, K. \end{aligned}$$

PROOF

First, define a function g via

$$g(t_n, m) := \sum_{n=1}^N y_n e^{-i \frac{2\pi m t_n}{P}} .$$

Observe that

$$\begin{aligned} g(t_n - t_1, m) &= \sum_{n=1}^N y_n e^{-i \frac{2\pi m (t_n - t_1)}{P}} \\ &= \sum_{n=1}^N y_n e^{-i \frac{2\pi m (n-1) \Delta T}{N \Delta T}} \\ &= \text{DFT}(m) , \text{ and} \\ g(t_n - t_1, m) &= \sum_{n=1}^N y_n e^{-i \frac{2\pi m (t_n - t_1)}{P}} \\ &= e^{i \frac{2\pi m t_1}{P}} \sum_{n=1}^N y_n e^{-i \frac{2\pi m t_n}{P}} \\ &= e^{i \frac{2\pi m t_1}{P}} \cdot g(t_n, m) . \end{aligned}$$

Combining these results yields

$$\text{DFT}(m) = e^{i \frac{2\pi m t_1}{P}} \cdot g(t_n, m) .$$

Now,

$$\begin{aligned} g(t_n, m) &:= \sum_{n=1}^N y_n e^{-i \frac{2\pi m t_n}{P}} \\ &= \sum_{n=1}^N y_n \left[\cos\left(\frac{2\pi m t_n}{P}\right) - i \sin\left(\frac{2\pi m t_n}{P}\right) \right] \\ &= \sum_{n=1}^N y_n \cos\left(\frac{2\pi m t_n}{P}\right) - i \sum_{n=1}^N y_n \sin\left(\frac{2\pi m t_n}{P}\right) \\ &= \frac{N}{2} \left[\frac{2}{N} \sum_{n=1}^N y_n \cos\left(\frac{2\pi m t_n}{P}\right) \right] - i \frac{N}{2} \left[\frac{2}{N} \sum_{n=1}^N y_n \sin\left(\frac{2\pi m t_n}{P}\right) \right] . \end{aligned}$$

When $m = 0$,

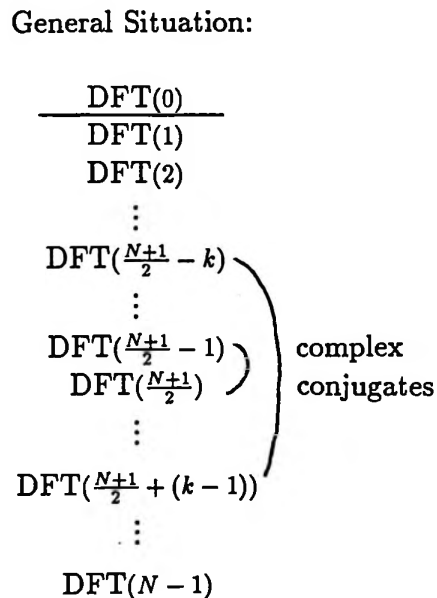
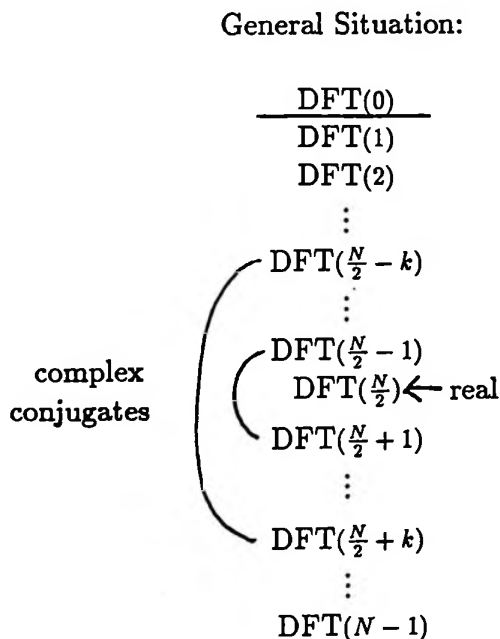
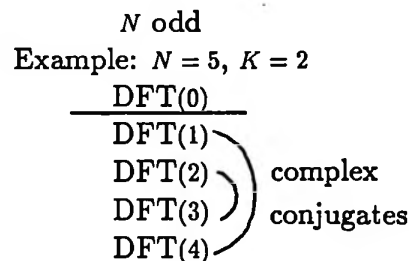
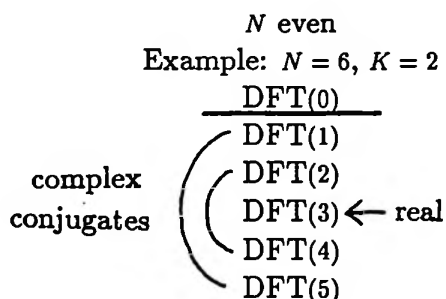
$$\begin{aligned} \text{DFT}(0) &= e^{i \frac{2\pi(0)t_1}{P}} \cdot g(t_n, 0) \\ &= N \left[\frac{1}{N} \sum_{n=1}^N y_n \right] = N \cdot a_0 . \end{aligned}$$

For $m = 1, \dots, K$,

$$\begin{aligned} \text{DFT}(m) &= e^{i \frac{2\pi m t_1}{P}} \cdot g(t_n, m) \\ &= e^{i \frac{2\pi m t_1}{P}} \cdot \frac{N}{2} (a_m - ib_m) . \quad \blacksquare \end{aligned}$$

the previous proposition describes the first half of the DFT list

With this proposition, the first half (approximately) of the N DFT values has been described in terms of the discrete Fourier series coefficients. The next proposition shows that the remaining DFT values are completely described in terms of the first half. The following sketches illustrate the situation when N is even, and when N is odd.



PROPOSITION When N is even, then $\text{DFT}(\frac{N}{2} - k)$ and $\text{DFT}(\frac{N}{2} + k)$ are complex conjugates, for $k = 1, \dots, K$. Also, $\text{DFT}(\frac{N}{2})$ is a real number,

$$\text{DFT}(\frac{N}{2}) = y_1 - y_2 + y_3 - y_4 + \dots + y_{N-1} - y_N.$$

When N is odd, then $\text{DFT}(\frac{N+1}{2} - k)$ and $\text{DFT}(\frac{N+1}{2} + (k-1))$ are complex conjugates, for $k = 1, \dots, K$.

PROOF
 N even

Suppose first that N is even. Let $m = \frac{N}{2} \pm k$, and let $\text{Re}(z)$ denote the real part of a complex number z . Then,

$$\begin{aligned} \text{Re}(e^{-i\frac{2\pi(\frac{N}{2} \pm k)(n-1)}{N}}) &= \text{Re}(e^{-i(\pi(n-1) \pm \frac{2\pi k(n-1)}{N})}) \\ &= \cos(\pi(n-1) \pm \overbrace{\frac{2\pi k(n-1)}{N}}^{:=K}) \\ &= \cos(\pi(n-1)) \cos K \mp \sin(\pi(n-1)) \sin K \\ &= \cos(\pi(n-1)) \cos K. \end{aligned}$$

It follows that the real parts of $\text{DFT}(\frac{N}{2} + k)$ and $\text{DFT}(\frac{N}{2} - k)$ are the same.

Continuing, let $\text{Im}(z)$ denote the imaginary part of a complex number z . Then,

$$\begin{aligned} \text{Im}(e^{-i\frac{2\pi(\frac{N}{2} \pm k)(n-1)}{N}}) &= \text{Im}(e^{-i(\pi(n-1) \pm \frac{2\pi k(n-1)}{N})}) \\ &= -\sin(\pi(n-1) \pm \overbrace{\frac{2\pi k(n-1)}{N}}^K) \\ &= -[\sin(\pi(n-1)) \cos K \pm \cos(\pi(n-1)) \sin K] \\ &= \mp \cos(\pi(n-1)) \sin K. \end{aligned}$$

In particular, the imaginary parts of $\text{DFT}(\frac{N}{2} + k)$ and $\text{DFT}(\frac{N}{2} - k)$ have opposite signs. This shows that $\text{DFT}(\frac{N}{2} - k)$ and $\text{DFT}(\frac{N}{2} + k)$ are complex conjugates, when N is even.

When $m = \frac{N}{2}$, one has

$$\begin{aligned} \text{DFT}(\frac{N}{2}) &= \sum_{n=1}^N y_n e^{-i\frac{2\pi \frac{N}{2}(n-1)}{N}} \\ &= \sum_{n=1}^N y_n e^{-i\pi(n-1)} \\ &= y_1 - y_2 + y_3 - y_4 + \dots + y_{N-1} - y_N, \end{aligned}$$

since $e^{-i\pi(n-1)}$ is alternately $+1$ and -1 .

N odd

Next, let N be odd. Write

$$\frac{N+1}{2} - k = \frac{N}{2} - (k - \frac{1}{2}) \quad \text{and} \\ \frac{N+1}{2} + (k-1) = \frac{N}{2} + (k - \frac{1}{2}) .$$

Replacing k by $k - \frac{1}{2}$ in the previous arguments completes the proof. ■

The next result relates the discrete Fourier transform to the periodogram.

PROPOSITION Let \bar{z} denote the complex conjugate of a complex number z . Then,

$$\sqrt{\text{DFT}(m) \cdot \overline{\text{DFT}(m)}} = \frac{N}{2} \sqrt{a_m^2 + b_m^2} , \quad m = 1, \dots, K .$$

PROOF

Let $K := \frac{2\pi m t_1}{P}$, and recall that for complex numbers z_1 and z_2 , $\overline{z_1 z_2} = \bar{z}_1 \cdot \bar{z}_2$. Then,

$$\begin{aligned} & \sqrt{\text{DFT}(m) \cdot \overline{\text{DFT}(m)}} \\ &= \sqrt{e^{i \frac{2\pi m t_1}{P}} \frac{N}{2} (a_m - ib_m) \cdot \overline{e^{i \frac{2\pi m t_1}{P}} \frac{N}{2} (a_m - ib_m)}} \\ &= \sqrt{e^{iK} \frac{N}{2} (a_m - ib_m) e^{-iK} \frac{N}{2} (a_m + ib_m)} \\ &= \frac{N}{2} \sqrt{a_m^2 + b_m^2} . \quad \blacksquare \end{aligned}$$

*the periodogram
in terms of
the DFT*

Consequently, the periodogram can be written as

$$\left\{ \left(\frac{P}{k}, \sqrt{\text{DFT}(k) \cdot \overline{\text{DFT}(k)}} \right) \mid k = 1, \dots, K \right\} .$$

*time savings:
2 min 50 sec
versus
6 sec*

Since the built-in MATLAB commands for implementing the DFT are so efficient, computation of the periodogram via the DFT provides a great time-savings. The MATLAB example at the conclusion of this section first computes the periodogram corresponding to a large data set (496 points) using the discrete Fourier series approach, from Section 2.6. This took 2 minutes and 50 seconds on the author's computer. However, finding the same periodogram by using MATLAB's built-in commands for the DFT, and exploiting the relationship between the DFT and the periodogram, took only 6 seconds!

MATLAB IMPLEMENTATION

Finding the Periodogram, using a fast Fourier transform

MATLAB FUNCTION `pervfft(D)`

The following MATLAB function finds the periodogram corresponding to a data set (with the maximum allowable value of K), by using the built-in MATLAB `fft` command to find the DFT, and then computing the values

$$\left\{ \left(\frac{P}{k}, \sqrt{\text{DFT}(k) \cdot \overline{\text{DFT}(k)}} \right) \mid k = 1, \dots, K \right\}.$$

To use the function, type:

```
[per,sqrcoef] = pervfft(D);
```

The name '`pervfft`' stands for '*periodogram via a fast Fourier transform*'.

required input, D

The required input is a data set $\{(t_i, y_i)\}_{i=1}^N$, with $N \geq 3$. The time values must be stored in a column vector `t` and the corresponding data values in a column vector `y`. Then, `D = [t y]` is the $N \times 2$ matrix containing the data set. The time list contained in `t` must be uniform (increment $\Delta T > 0$). The program begins by checking that this requirement is met; if not, the program is halted and the message 'not a uniform time list' is displayed. (As written, increments between time values are said to 'differ' if they differ by more than 0.0000001. This value may be changed for different tolerances.)

outputs: `per` `sqrcoef`

The output `per` is a column vector containing the periods $P, \frac{P}{2}, \frac{P}{3}, \dots, \frac{P}{K}$, where $P = N\Delta T$.

The output `sqrcoef` is a column vector containing the numbers

$$\sqrt{\text{DFT}(k) \cdot \overline{\text{DFT}(k)}} = \frac{N}{2} \sqrt{a_k^2 + b_k^2},$$

for $k = 1, \dots, K$.

The periodogram is then obtained with the command:

```
plot(per,sqrcoef)
```

```

function [per,sqrcoef] = pervfft(D)
t = D(:,1);
% First, check that the time list is uniform:
d = diff(t);
% entries that differ by more than .0000001 are called 'different'
p = ( abs(ones(d)*d(1) - d) > .0000001);
err = find(p);
if max(err) ~= 0
    'not a uniform time list'
    return
end
N = length(t);
dT = t(2) - t(1);
P = N*dT;
y = D(:,2);
K = floor( (N-1)/2 );
dft = fft(y);
dft = dft(2:K+1);
sqrcoef = sqrt(dft .* conj(dft));
per = P*ones(K,1);
k = [1:K]';
per = per ./ k;

```

**MATLAB
EXAMPLE**

The following example gives a time-comparison between computing the periodogram via the program in Section 2.6, and via a fast Fourier transform.

```

% First, construct a large data set
t = [1:.2:100]';
y = sin((2*pi/7)*t);
length(t)

ans =

    496

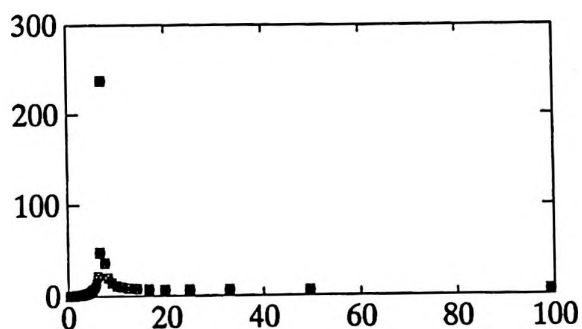
D = [t y];
% Use the discrete Fourier series approach, from Section 2.6
[per,sqrcoef] = dfs(D);
% That took 2 minutes and 50 seconds on the author's computer

```

```
% Now, use the fast Fourier transform approach  
[per2,sqrcoef2] = pervfft(D);  
% That took 6 seconds!  
% Here's the periodogram:  
subplot(221)  
plot(per,sqrcoef,'x')  
hold
```

Current plot held

```
plot(per2,sqrcoef2,'o')
```



CHAPTER 3
FILTER THEORY

3.1 Mathematical Filters

What is a
mathematical
filter?

A (non-mathematical) 'filter' is usually a mechanical device used to process material, often to extract a particular component, like only the finest particles in a granular sand mixture. A mathematical filter is used for a similar purpose—to process data in some desirable way. For example, filters are used for removal of noise, smoothing, separation of signals, differentiating, and integrating.

analog
versus
digital
filters

Analog filters are used to process continuous signals, and *digital filters* are used to process discrete signals that are equally-spaced; that is, data sets $\{(t_i, y_i)\}_{i=1}^N$ where the time list is uniform.

NOTATION
for
digital filters

Let (y_n) be a list of data values that arose from a data set with a uniform time list. The list (y_n) is assumed to be of the form $(y_n)_{n=1}^N$ or $(y_n)_{n=-\infty}^{\infty}$.

A digital filter is a function that acts on the list (y_n) , and produces a new 'filtered' list, (f_n) , where

$$f_n = \sum_{k=-\infty}^{\infty} c_k y_{n-k} + \sum_{k=-\infty}^{\infty} d_k f_{n-k},$$

and where c_k and d_k are real constants. For finite lists $(y_n)_{n=1}^N$, f_n is defined if and only if

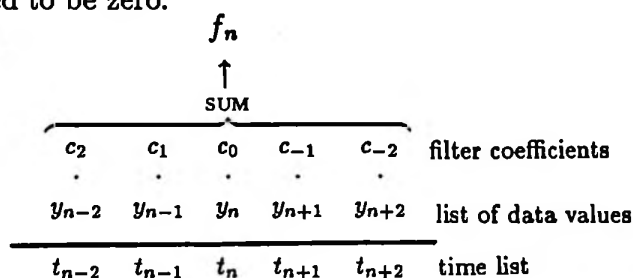
$$1 \leq n - k \leq N \text{ whenever } c_k \neq 0.$$

filter
coefficients

The numbers c_k and d_k are called the *filter coefficients*. In practice, all but a finite number of the filter coefficients are zero, so that the sums are actually finite.

expression is
'centered' at y_n

The subscripts $n - k$ serve to 'center' the expression at y_n , as illustrated below. In the following diagram, all coefficients d_k are assumed to be zero.



The data point (t_n, y_n) corresponds to the filter point (t_n, f_n) .

NOTATION
for
digital filters
(continued)

current,
past, and
future values

nonrecursive
filters,
all $d_k = 0$

recursive
filters,
some $d_k \neq 0$

When filter value f_n is being computed, f_n is called the *current filter value*, and y_n is called the *current data value*. The numbers f_{n-k} and y_{n-k} , for $k = 1, \dots, \infty$, are called the *past filter and data values*, respectively. The numbers f_{n-k} and y_{n-k} , for $k = -1, \dots, -\infty$, are called the *future filter and data values*, respectively.

If all coefficients d_k are zero, then only the data values y_n are used to compute the filter output. In this case, the filter is called *nonrecursive*.

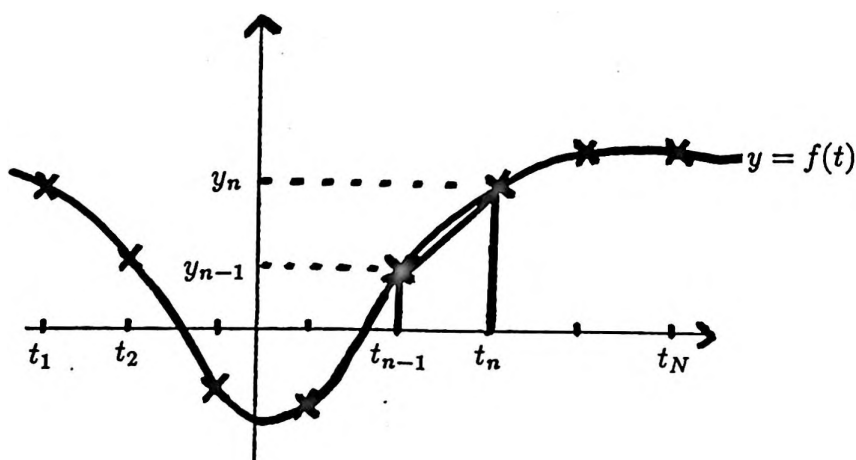
If any of the coefficients d_k are nonzero, then computation of f_n requires other filter values. In this case, the filter is called *recursive*. In particular, if future values of the filter are used, then a system of linear algebraic equations must be solved to find f_n .

Primarily nonrecursive digital filters will be discussed in this dissertation.

EXAMPLE
Trapezoid Rule
for approximate
integration
is a
recursive filter

Let $\{(t_i, y_i)\}_{i=1}^N$ be a data set with a uniform time list having spacing $\Delta T > 0$. If this data set arose by sampling from a continuous function f , that is, if $y_n = f(t_n)$, then one may wish to use these values to approximate the integral

$$\int_{t_1}^{t_n} f(t) dt, \text{ for } n = 2, \dots, N. \quad (1)$$



Let f_n be the approximation to (1) that is obtained by using the Trapezoid Rule for approximation (e.g., see [S&B, p.121]). Set $f_1 := 0$. Then,

$$f_n = \frac{1}{2}\Delta T(y_n + y_{n-1}) + f_{n-1}, \text{ for } n = 2, \dots, N.$$

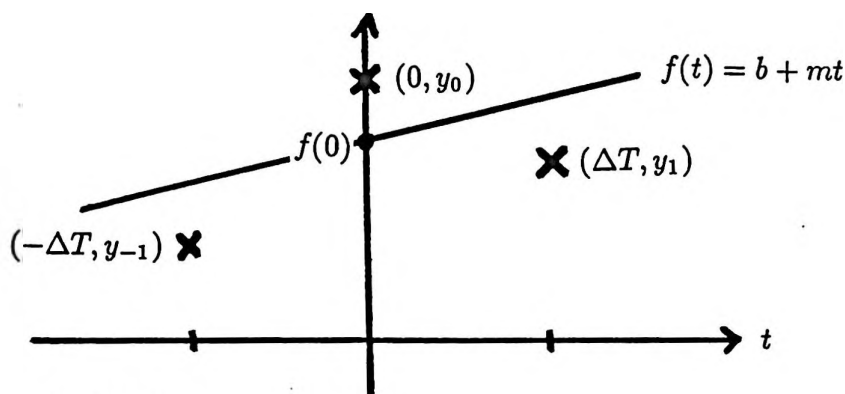
This is a recursive filter with $c_0 = c_1 = \frac{1}{2}\Delta T$, and with $d_1 = 1$. All other coefficients are zero.

EXAMPLE
smoothing by 3's

Suppose one desires to 'smooth' a data set as follows. Given three successive points (which, for convenience, are centered at 0),

$$(-\Delta T, y_{-1}), (0, y_0), \text{ and } (\Delta T, y_1),$$

it is desired to 'fit' these three points (in the least-squares sense) with a line $f(t) = b + mt$, and then use the midpoint value $f(0) = b$ as the 'smoothed' value.



By defining

$$f_1(t) := 1 \text{ and } f_2(t) := t,$$

$$\mathbf{t} = \begin{bmatrix} -\Delta T \\ 0 \\ \Delta T \end{bmatrix} \text{ and } \mathbf{y} = \begin{bmatrix} y_{-1} \\ y_0 \\ y_1 \end{bmatrix},$$

$$\mathbf{f}_1(t) := \begin{bmatrix} f_1(-\Delta T) \\ f_1(0) \\ f_1(\Delta T) \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \text{ and } \mathbf{f}_2(t) := \begin{bmatrix} f_2(-\Delta T) \\ f_2(0) \\ f_2(\Delta T) \end{bmatrix} = \begin{bmatrix} -\Delta T \\ 0 \\ \Delta T \end{bmatrix},$$

$$\mathbf{X} = [\mathbf{f}_1(t) \quad \mathbf{f}_2(t)] = \begin{bmatrix} 1 & -\Delta T \\ 1 & 0 \\ 1 & \Delta T \end{bmatrix} \text{ and } \mathbf{b} = \begin{bmatrix} b \\ m \end{bmatrix},$$

and using the results from Section 2.2, one obtains

$$\begin{aligned}
\mathbf{b} &= (\mathbf{X}^t \mathbf{X})^{-1} \mathbf{X}^t \mathbf{y} \\
&= \begin{bmatrix} 3 & 0 \\ 0 & 2(\Delta T)^2 \end{bmatrix}^{-1} \mathbf{X}^t \mathbf{y} \\
&= \begin{bmatrix} \frac{1}{3} & 0 \\ 0 & \frac{1}{2(\Delta T)^2} \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 \\ -\Delta T & 0 & \Delta T \end{bmatrix} \begin{bmatrix} y_{-1} \\ y_0 \\ y_1 \end{bmatrix} \\
&= \begin{bmatrix} \frac{1}{3}(y_{-1} + y_0 + y_1) \\ \frac{1}{2\Delta T}(y_1 - y_{-1}) \end{bmatrix} .
\end{aligned}$$

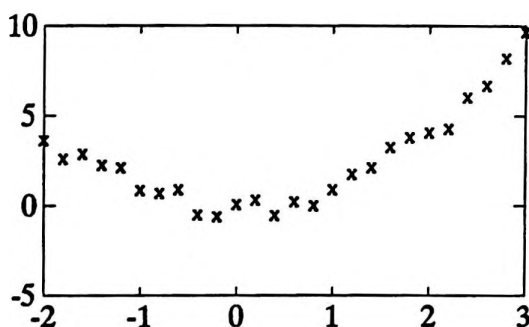
Then, $f(0) = b = \frac{1}{3}(y_{-1} + y_0 + y_1)$. Thus, the current filter value is found by averaging the immediate past, current, and immediate future data values. Generalizing this process gives the filter

$$f_n = \frac{1}{3}(y_{n-1} + y_n + y_{n+1}) ,$$

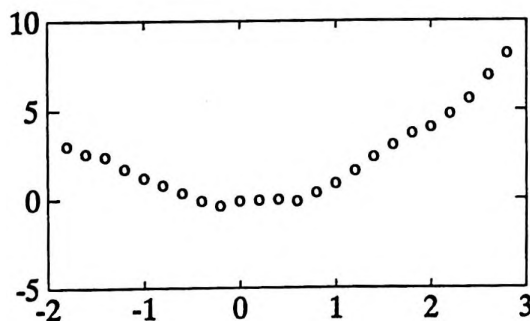
which is nonrecursive, with $c_{-1} = c_0 = c_1 = \frac{1}{3}$. Not surprisingly, this filter is called a *moving average filter*, or *smoothing by 3's*.

EXAMPLE
applying the
smoothing by 3's
filter

A data set, generated by $y(t) = t^2$ and then corrupted with noise, is plotted below using the plot symbol 'x'. The *smoothing by 3's* filter is applied, and the filtered output is plotted with 'o'. Note that the filtered curve is certainly 'smoother', so the filter is appropriately named. A better understanding of what this filter does will come from studying its corresponding *transfer function* in Section 3.2.



data set, 'x'



smoothing by 3's filter
output, 'o'

the
filter lag
problem

Consider applying the *smoothing by 3's* filter to the data values from a finite data set $\{(t_i, y_i)\}_{i=1}^N$. Since computation of filter value f_n requires knowledge of the values y_{n-1} , y_n and y_{n+1} , it is not possible to compute f_1 (since y_0 is unknown) or f_N (since y_{N+1} is unknown). This inability to get filter output corresponding to the entire data set becomes worse as the number of nonzero coefficients in the filter increases.

window of
coefficients

In filter literature, the *smoothing by 3's* filter is often described via the phrase: 'we are looking at the data through the window $[\frac{1}{3} \ \frac{1}{3} \ \frac{1}{3}]$ '. One imagines this 'window' moving successively to the right through the data values, producing the filtered values, as illustrated below.

$$\begin{array}{ccccccc}
 & f_2 & & & & & f_n \\
 & \uparrow & & & & & \uparrow \\
 & \underbrace{\quad\quad\quad} & & & & & \underbrace{\quad\quad\quad} \\
 & [\frac{1}{3} \quad \frac{1}{3} \quad \frac{1}{3}] & & & & & [\frac{1}{3} \quad \frac{1}{3} \quad \frac{1}{3}] \\
 & \cdot \quad \cdot \quad \cdot & & & & & \cdot \quad \cdot \quad \cdot \\
 y_1 & y_2 & y_3 & y_4 & y_5 & \cdots & y_{n-1} & y_n & y_{n+1}
 \end{array}$$

a note
regarding the
filter coefficients

Suppose that the sum of the filter coefficients for a nonrecursive filter is 1, that is,

$$\sum_k c_k = 1.$$

(This is true in the previous example, since $\frac{1}{3} + \frac{1}{3} + \frac{1}{3} = 1$.) Then, when a constant function $y_n := K$ is input to the filter, the same constant emerges as the filter output:

$$f_n = \sum_k c_k y_{n-k} = \sum_k c_k (K) = K \sum_k c_k = K(1) = K.$$

a MATLAB
function for
applying
nonrecursive filters

A MATLAB function for applying nonrecursive filters is given next. It is assumed that the nonrecursive filter is of the form

$$f_n = \sum_{k=-K}^K c_k y_{n-k}$$

for a positive integer K ; some of the coefficients c_k may equal zero. There are $2K+1$ filter coefficients: $c_K, \dots, c_1, c_0, c_{-1}, \dots, c_{-K}$.

MATLAB IMPLEMENTATION

Applying a Nonrecursive Filter

MATLAB
function
nonrec(D,FC)

The MATLAB function **nonrec** applies the nonrecursive filter

$$f_n = \sum_{k=-K}^K c_k y_{n-k}$$

to the data values in a data set **D**. The filter coefficients are stored in a column vector **FC**.

To use the function, type:

[f tf] = nonrec(D,FC);

REQUIRED INPUTS

t_1	y_1	
t_2	y_2	
\vdots	\vdots	
\vdots	\vdots	
\vdots	\vdots	c_K
\vdots	\vdots	\vdots
\vdots	\vdots	c_1
t_n	y_n	c_0
\vdots	\vdots	\vdots
\vdots	\vdots	c_{-1}
\vdots	\vdots	\vdots
\vdots	\vdots	c_{-K}
t_N	y_N	

$\left. \begin{matrix} c_K \\ \vdots \\ c_1 \\ c_0 \\ c_{-1} \\ \vdots \\ c_{-K} \end{matrix} \right\} K \text{ coefficients}$
 $\rightarrow f_n$
 $\left. \begin{matrix} c_{-1} \\ \vdots \\ c_{-K} \end{matrix} \right\} K \text{ coefficients}$

- A data set $\{(t_i, y_i)\}_{i=1}^N$ is required, with a uniform time list. Here, N is a positive integer that gives the number of data points.

The time values are stored in an N -column vector called **t**, and the corresponding data values are stored in an N -column vector called **y**. Then, **D** = [**t y**] is the $N \times 2$ matrix containing the data set.

- The $2K+1$ real number filter coefficients $c_K, \dots, c_1, c_0, c_{-1}, \dots, c_{-K}$ must be stored, in the indicated order, in a column vector called **FC** (for 'Filter Coefficients'). Here, K is a nonnegative integer.
- It must be the case that $N > 2K$ if any filter output is to be observed. Otherwise, the output matrices **f** and **tf** will be empty.

OUTPUT

The function outputs 2 column vectors, **f** and **tf**.

The vector **f** contains the filtered output. Filtered output f_n corresponds to data value y_n .

The vector **tf** contains the time values for the filtered output (**tf** stands for 'time for filtered output'). Filtered output f_n corresponds to time value t_n .

The first and last K values in **y** cannot be processed, due to filter lag. Therefore, the length of **f** (and **tf**) is $N - 2K$.

plotting the original data versus the filtered data The original data can be plotted versus the filtered data with the commands:

```
plot(t,y,'x')
hold
plot(tf,f,'o')
```

source code The source code for the function `nonrec` is given next.

```
% Copyright 1994 Carol J.V. Fisher
function [f,tf] = nonrec(D,FC)
t = D(:,1);
y = D(:,2);
N = length(y);
% There are 2K+1 filter coefficients.
K = (length(FC) - 1)/2;
if K ~= floor(K)
    'There must be an odd number of filter coefficients'
end
% Initialize the matrix to hold the filtered output.
f = zeros(N-2*K,1);
for n = K+1:N-K
    f(n-K) = sum( FC .* y(n-K:n+K) );
end
tf = t(K+1:N-K);
```

EXAMPLE The next example illustrates the use of `nonrec`. A noisy data set is constructed. Three filters are then applied:

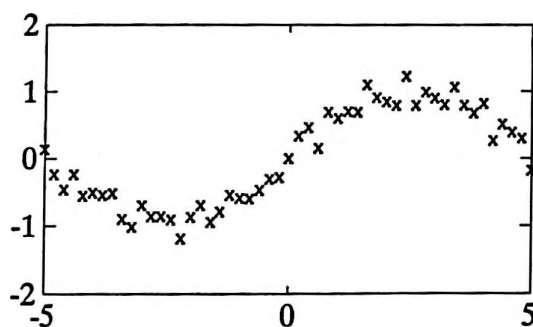
- the *smoothing by 3's* filter (with window $\frac{1}{3}[1 \ 1 \ 1]$);
- the *smoothing by 5's* filter (with window $\frac{1}{5}[1 \ 1 \ 1 \ 1 \ 1]$); and
- the *smoothing by 7's* filter (with window $\frac{1}{7}[1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1]$).

Observe that the filtered data does appear to become increasingly 'smoother'. Also notice the increasing filter lag problem. A further understanding of these filters comes from studying their corresponding *transfer functions*, which is the subject of the next section.

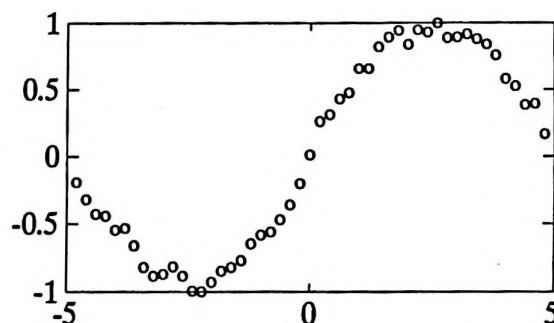
```

t = [-5:.2:5]';
y = sin(2*pi*t/10);
noise = rand(t) - .5;
y = y + .5*noise;
subplot(221)
plot(t,y,'x')
D = [t y];
[f1 tf1] = nonrec(D,[1/3 1/3 1/3]');
plot(tf1,f1,'o')
[f2 tf2] = nonrec(D,[1/5 1/5 1/5 1/5 1/5]');
plot(tf2,f2,'o')
[f3 tf3] = nonrec(D,[1/7 1/7 1/7 1/7 1/7 1/7 1/7]');
plot(tf3,f3,'o')

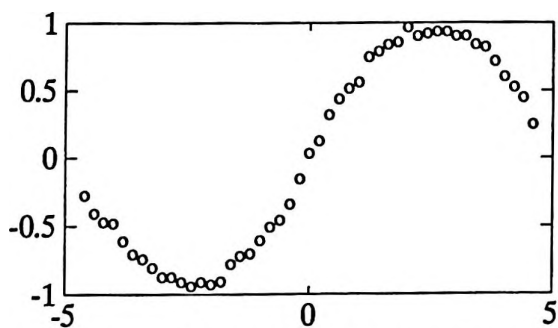
```



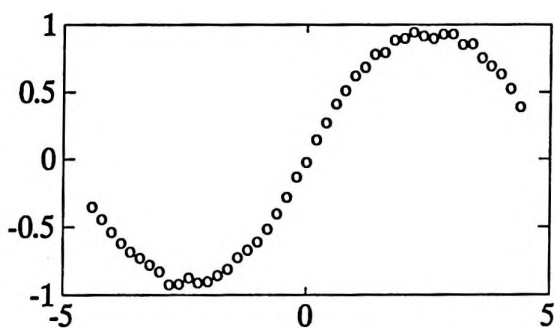
data set



smoothing by 3's filter output



smoothing by 5's filter output



smoothing by 7's filter output

3.2 Transfer Functions

*nonrecursive
digital filters*

In this section, the *nonrecursive digital filter*

$$f_n = \sum_{k=-K}^K c_k y_{n-k} \quad (\text{NF})$$

is investigated, where K is a positive integer, and the filter coefficients c_k are real numbers.

Under certain symmetry requirements on the coefficients c_k , the action of this filter on sums of sinusoidal components is easily explained, via the *transfer function* corresponding to the filter. The development of the transfer function calls on the linear algebra concepts of *eigenvalues* and *eigenvectors*, so a quick review is in order:

*vector spaces
and linear
transformations
are reviewed in
Appendix 2*

Vector spaces and linear transformations between vector spaces are reviewed in Appendix 2. The definitions of *eigenvalue* and *eigenvector* are given next.

DEFINITIONS

*eigenvalue;
eigenvector*

Let V be a vector space over F , and let T be a linear transformation from V into V .

An *eigenvalue* of T is a scalar $\lambda \in F$ for which there exists a nonzero vector $v \in V$ with $Tv = \lambda v$.

If λ is an eigenvalue of T , then any nonzero vector v satisfying $Tv = \lambda v$ is called an *eigenvector of T corresponding to λ* .

*the action of T
on its eigenvectors
is just scalar
multiplication*

It is important to observe that the action of T on its eigenvectors is particularly simple—it is just scalar multiplication. That is, the linear transformation T maps an eigenvector v to the scaled vector λv . Note also that any eigenvalue of a linear transformation between *real* vector spaces is, by definition, a real number.

EXAMPLE
*the vector space
 \mathbb{R}^∞*

Let \mathbb{R}^∞ denote the set of ‘doubly infinite’ lists with a designated origin; i.e., each member of \mathbb{R}^∞ is of the form

$$(\dots, y_{-3}, y_{-2}, y_{-1}, \hat{y}_0, y_1, y_2, y_3, \dots),$$

where $y_i \in \mathbb{R}$ for all integers i , and the element indicated by the ‘ $\hat{}$ ’ is the origin. The ‘origin’ is needed for a well-defined addition on \mathbb{R}^∞ , and will only be shown when necessary.

*addition and
scalar
multiplication
in \mathbb{R}^∞*

For an element of \mathbb{R}^∞ , define multiplication by $\alpha \in \mathbb{R}$ via

$$\alpha(\dots, y_{-1}, \widehat{y_0}, y_1, \dots) := (\dots, \alpha y_{-1}, \widehat{\alpha y_0}, \alpha y_1, \dots) .$$

Addition in \mathbb{R}^∞ is defined by first aligning the origins of the elements being added, and then adding componentwise:

$$\begin{aligned} & (\dots, x_{-1}, \widehat{x_0}, x_1, \dots) \\ + & (\dots, y_{-1}, \widehat{y_0}, y_1, \dots) \\ = & (\dots, x_{-1} + y_{-1}, \widehat{x_0 + y_0}, x_1 + y_1, \dots) . \end{aligned}$$

With this addition and multiplication by real numbers, \mathbb{R}^∞ is a real vector space.

In preparation for viewing the nonrecursive filter (NF) as a linear transformation between vector spaces, it is first necessary to view finite input lists (y_n) and output lists (f_n) as elements of \mathbb{R}^∞ .

*every finite list
of data values
can be viewed
as an element
in \mathbb{R}^∞*

Every finite list $(y_n)_{n=1}^N$ of data values can be associated with an element of \mathbb{R}^∞ by 'padding it with zeros' in both directions, and designating y_1 as the origin; that is,

$$(y_1, y_2, \dots, y_N) \mapsto (\dots, 0, 0, 0, \widehat{y_1}, y_2, \dots, y_N, 0, 0, 0, \dots) .$$

Once this association with an element in \mathbb{R}^∞ is made, there is no longer a 'filter lag' problem: that is, filtered values f_i can be found for all integers i . In this way, one has the following input and output lists in \mathbb{R}^∞ :

$$\begin{aligned} & (\dots, 0, 0, \widehat{y_1}, y_2, \dots, y_N, 0, 0, \dots) \\ & (\dots, f_{-1}, f_0, \widehat{f_1}, f_2, \dots, f_N, f_{N+1}, f_{N+2}, \dots) \end{aligned}$$

Note that f_n will be zero, for values of n that are sufficiently large, and sufficiently negative.

*a nonrecursive
filter is a linear
transformation*

Next, the nonrecursive filter (NF) is used to define a transformation from \mathbb{R}^∞ to \mathbb{R}^∞ , and it is shown that this transformation is linear. In the following discussion, it is assumed that the (finite) list $(y_n)_{n=1}^N$ and the filter list (f_n) are associated with elements in \mathbb{R}^∞ , as discussed above.

the map F
that takes a
data list (y_n) to
its corresponding
filtered list (f_n)
is linear

Let $F: \mathbb{R}^\infty \rightarrow \mathbb{R}^\infty$ be defined by $F(y) := f$, where $y := (y_n)$ is a list of data values, and $f := (f_n)$ is its corresponding list of filter values, with

$$f_n = \sum_{k=-K}^K c_k y_{n-k}. \quad (1)$$

For the remainder of this section, the entry f_n of $F(y)$ is optionally denoted by $F(y)_n$, the entry y_n of y by y_n , and the sum $\sum_{k=-K}^K$ by \sum_k . With this notation, the sum in (1) is rewritten as

$$F(y)_n = \sum_k c_k y_{n-k}.$$

F is linear

Let $x = (x_n)_{n=-\infty}^\infty$ and $y = (y_n)_{n=-\infty}^\infty$ be elements of \mathbb{R}^∞ . Then,

$$\begin{aligned} F(x+y)_n &= \sum_k c_k (x+y)_{n-k} \\ &= \sum_k c_k x_{n-k} + \sum_k c_k y_{n-k} \\ &= F(x)_n + F(y)_n, \end{aligned}$$

which implies that $F(x+y) = F(x) + F(y)$. Also, for $\alpha \in \mathbb{R}$,

$$F(\alpha x)_n = \sum_k c_k (\alpha x)_{n-k} = \alpha \sum_k c_k x_{n-k} = \alpha F(x)_n,$$

which implies that $F(\alpha x) = \alpha F(x)$. Thus, F is a linear operator from \mathbb{R}^∞ to \mathbb{R}^∞ .

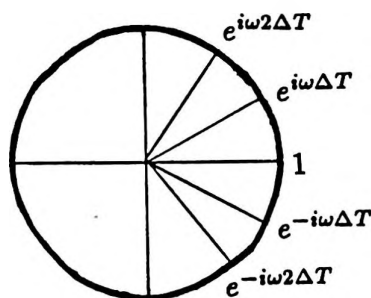
input the list
 $(e^{i\omega n \Delta T})$ to
the filter

Let ΔT be a positive real number, and let $\omega \in \mathbb{R}$.

As a motivation for later results, define a list e_ω via

$$(e_\omega)_n := e^{i\omega n \Delta T}.$$

The sketch below illustrates some list entries when $\omega > 0$.



Use e_ω as the input list to the nonrecursive filter (NF). In so doing, one obtains the output list (f_n) , where

$$\begin{aligned} f_n &= \sum_k c_k (e_\omega)_{n-k} = \sum_k c_k e^{i\omega(n-k)\Delta T} \\ &= \sum_k c_k e^{i\omega n\Delta T} e^{-i\omega k\Delta T} = \left(\sum_k c_k e^{-i\omega k\Delta T} \right) e^{i\omega n\Delta T} \\ &:= C \cdot (e_\omega)_n, \end{aligned}$$

where $C := \sum_{k=-K}^K c_k e^{-i\omega k\Delta T}$. Thus, when the list e_ω is input to (NF), the filtered list Ce_ω emerges. In general, C may not be a real number. However, the next result shows that under certain symmetry requirements on the filter coefficients c_k , C will be real:

LEMMA

Let c_k , $-K \leq k \leq K$, be real numbers satisfying

$$c_k = c_{-k} \quad \text{for } k = 1, \dots, K.$$

Then, the number

$$C := \sum_{k=-K}^K c_k e^{-i\omega k\Delta T}$$

is real, for all real numbers ω and ΔT .

PROOF

$$\begin{aligned} \sum_{k=-K}^K c_k e^{-i\omega k\Delta T} &= c_0 + \sum_{k=1}^K c_k e^{-i\omega k\Delta T} + \sum_{k=-1}^{-K} c_k e^{-i\omega k\Delta T} \\ &= c_0 + \sum_{k=1}^K c_k e^{-i\omega k\Delta T} + \sum_{j=1}^K \overbrace{c_{-j} e^{i\omega j\Delta T}}^{j := -k} \\ &= c_0 + \sum_{k=1}^K c_k e^{-i\omega k\Delta T} + \sum_{j=1}^K c_j e^{i\omega j\Delta T} \\ &= c_0 + \sum_{k=1}^K c_k (e^{-i\omega k\Delta T} + e^{i\omega k\Delta T}) \\ &= c_0 + \sum_{k=1}^K c_k (2 \cos(\omega k\Delta T)), \end{aligned}$$

which is a real number. ■

DEFINITION A nonrecursive digital filter

symmetric
nonrecursive filter

$$f_n = \sum_{k=-K}^K c_k y_{n-k} ,$$

for which $c_k = c_{-k}$, $k = 1, \dots, K$, is called *symmetric*.

The next result shows that any nonzero list in \mathbb{R}^∞ that is formed by evaluating the functions $\sin(\omega t)$ and $\cos(\omega t)$ at the time values

$$(\dots, t_1 - 2\Delta T, t_1 - \Delta T, \hat{t}_1, t_1 + \Delta T, t_1 + 2\Delta T, \dots) ,$$

where $t_1 \in \mathbb{R}$, $\Delta T > 0$, and $\omega \in \mathbb{R}$, is an eigenvector for the linear transformation $F: \mathbb{R}^\infty \rightarrow \mathbb{R}^\infty$ defined in (1), providing that the nonrecursive filter used to define F is *symmetric*:

THEOREM Let $t_1 \in \mathbb{R}$, $\Delta T > 0$, and $\omega \in \mathbb{R}$. Form lists

$$(\sin \omega(t_1 + n\Delta T))_{n=-\infty}^{\infty} \quad \text{and} \quad (\cos \omega(t_1 + n\Delta T))_{n=-\infty}^{\infty} \quad (2)$$

in \mathbb{R}^∞ by evaluating the functions $\sin \omega t$ and $\cos \omega t$ on the time list

$$(\dots, t_1 - 2\Delta T, t_1 - \Delta T, \hat{t}_1, t_1 + \Delta T, t_1 + 2\Delta T, \dots) .$$

Assume that t_1 , ΔT and ω are such that the lists in (2) are nonzero.

Then, the lists in (2) are eigenvectors for the linear transformation $F: \mathbb{R}^\infty \rightarrow \mathbb{R}^\infty$ defined by the symmetric nonrecursive filter

$$f_n = \sum_{k=-K}^K c_k y_{n-k} , \quad c_k = c_{-k} \text{ for } k = 1, \dots, K .$$

Both lists have corresponding eigenvalue

$$\sum_{k=-K}^K c_k e^{-i\omega k \Delta T} .$$

PROOF

The following argument uses the facts that for all $z, w \in \mathbb{C}$ and for all $\alpha, \beta \in \mathbb{R}$,

$$\operatorname{Re}(\alpha z + \beta w) = \alpha \operatorname{Re}(z) + \beta \operatorname{Re}(w)$$

$$\operatorname{Im}(\alpha z + \beta w) = \alpha \operatorname{Im}(z) + \beta \operatorname{Im}(w) ,$$

where $\text{Re}(z)$ and $\text{Im}(z)$ denote the real and imaginary parts, respectively, of a complex number z .

Define

$$C := \sum_{k=-K}^K c_k e^{-i\omega k \Delta T} ,$$

and recall that C is a real number, since $c_k = c_{-k}$ for $k = 1, \dots, K$.

Also define

$$\mathbf{S} := (\sin \omega(t_1 + n\Delta T))_{n=-\infty}^{\infty} ,$$

so that

$$\mathbf{S}_n = \sin \omega(t_1 + n\Delta T) .$$

Then,

$$\begin{aligned} F(\mathbf{S})_n &= \sum_k c_k \sin \omega(t_1 + (n-k)\Delta T) \\ &= \sum_k c_k \text{Im}(e^{i\omega(t_1 + (n-k)\Delta T)}) \\ &= \text{Im} \left(\sum_k c_k e^{i\omega(t_1 + (n-k)\Delta T)} \right) \\ &= \text{Im} \left(\sum_k c_k e^{i\omega(t_1 + n\Delta T)} e^{-i\omega k \Delta T} \right) \\ &= \text{Im} \left(e^{i\omega(t_1 + n\Delta T)} \overbrace{\sum_k c_k e^{-i\omega k \Delta T}}^{\text{real number}} \right) \\ &= C \cdot \text{Im}(e^{i\omega(t_1 + n\Delta T)}) \\ &= C \cdot (\sin \omega(t_1 + n\Delta T)) \\ &= C \cdot \mathbf{S}_n . \end{aligned}$$

It follows that $F(\mathbf{S}) = C\mathbf{S}$. Thus, \mathbf{S} is an eigenvector for F with eigenvalue C .

By defining

$$\mathbf{C} := (\cos \omega(t_1 + n\Delta T))_{n=-\infty}^{\infty} ,$$

and replacing \mathbf{S} by \mathbf{C} , \sin by \cos , and Im by Re in the argument above, it follows that \mathbf{C} is also an eigenvector for F with eigenvalue C . ■

DEFINITION
*transfer function
 for a symmetric
 nonrecursive filter*

Let F be the symmetric nonrecursive filter given by

$$f_n = \sum_{k=-K}^K c_k y_{n-k}, \quad c_k = c_{-k} \text{ for } k = 1, \dots, K.$$

Let ΔT be a fixed positive number. The function $H: \mathbf{R} \rightarrow \mathbf{R}$ defined by

$$H(\omega) := \sum_{k=-K}^K c_k e^{-i\omega k \Delta T} = c_0 + \sum_{k=1}^K 2c_k \cos(\omega k \Delta T)$$

is called the *transfer function corresponding to F and spacing ΔT* .

*information
 given by the
 transfer function*

The transfer function has the property that when the functions $\sin \omega t$ or $\cos \omega t$ are sampled at the equally-spaced time values $(\dots, t_1 - 2\Delta T, t_1 - \Delta T, t_1, t_1 + \Delta T, t_1 + 2\Delta T, \dots)$, and then input to the symmetric nonrecursive filter, the same lists emerge, except scaled by the constant $H(\omega)$.

Since F is linear, if a sum of sinusoidal components is input to F , then the same sum will emerge, with each component appropriately scaled. The examples following the next lemma illustrate this process.

LEMMA
 *H has
 period $\frac{2\pi}{\Delta T}$*

The transfer function H of the previous definition has period $\frac{2\pi}{\Delta T}$.

PROOF

$$\begin{aligned} H\left(\omega + \frac{2\pi}{\Delta T}\right) &= \sum_{k=-K}^K c_k e^{-i\left(\omega + \frac{2\pi}{\Delta T}\right)k\Delta T} \\ &= \sum_{k=-K}^K c_k e^{-i\omega k \Delta T} \overbrace{e^{-i2\pi k}}^{=1} \\ &= H(\omega). \quad \blacksquare \end{aligned}$$

Thus, it suffices to study the transfer function H on any interval of length $\frac{2\pi}{\Delta T}$, say the interval $[0, \frac{2\pi}{\Delta T}]$.

EXAMPLE
transfer functions
for the
smoothing by K 's
filters, where
 $K \geq 3$ is an
odd integer

Let $K \geq 3$ be an odd integer. Let $K = 2m + 1$ for a positive integer m . The smoothing by K 's filter,

$$f_n = \frac{1}{K} \sum_{k=-m}^m y_{n-k} ,$$

has coefficients

$$c_{-m} = \cdots = c_{-1} = c_0 = c_1 = \cdots = c_m = \frac{1}{K} .$$

The corresponding transfer function is

$$\begin{aligned} H(\omega) &= c_0 + \sum_{k=1}^m 2c_k \cos(\omega k \Delta T) \\ &= \frac{1}{K} \left(1 + 2 \sum_{k=1}^m \cos(\omega k \Delta T) \right) . \end{aligned}$$

rewriting the
transfer function
in terms of
cyclic frequency:
 $\tilde{H}(f)$

The positive number ω gives the radian frequency of the functions $\sin \omega t$ and $\cos \omega t$. It is usually more convenient to work with the transfer function expressed in terms of cyclic frequency f , where $\omega = 2\pi f$. To this end, define a function \tilde{H} by

$$\tilde{H}(f) := H(2\pi f) = H(\omega) .$$

Therefore,

$$\tilde{H}(f) = \sum_{k=-K}^K c_k e^{-i2\pi f k \Delta T} = c_0 + \sum_{k=1}^K 2c_k \cos(2\pi f k \Delta T) .$$

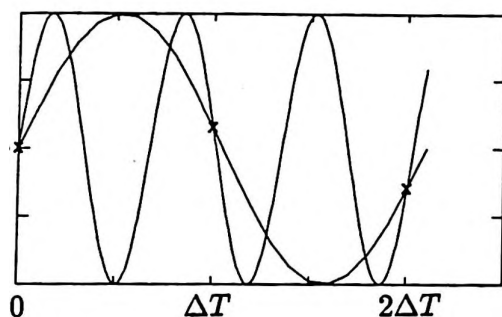
\tilde{H}
has period $\frac{1}{\Delta T}$

Since H has period $\frac{2\pi}{\Delta T}$, it follows that \tilde{H} has period $\frac{1}{\Delta T}$:

$$\begin{aligned} \tilde{H}\left(f + \frac{1}{\Delta T}\right) &:= H\left(2\pi\left(f + \frac{1}{\Delta T}\right)\right) \\ &= H\left(2\pi f + \frac{2\pi}{\Delta T}\right) \\ &= H(2\pi f) \\ &:= \tilde{H}(f) . \end{aligned}$$

\tilde{H} is
only graphed
on $[0, \frac{1}{2\Delta T}]$

With a sample spacing of ΔT , it is unreasonable to try and detect periods less than $2\Delta T$, since these smaller periods are 'confused' with larger ones due to the effects of aliasing (see pages 91-93 and the sketch below). For this reason, the function \tilde{H} is only graphed, in practice, on the interval of cyclic frequencies $[0, \frac{1}{2\Delta T}]$.



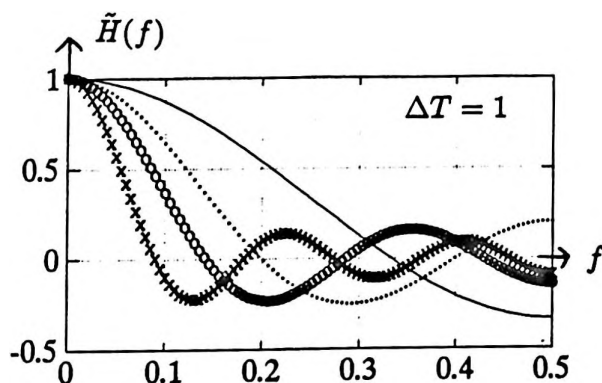
ONLY PERIODS GREATER THAN $2\Delta T$
CAN BE DETECTED

graphs of
 \tilde{H} for the
smoothing by K 's
filters

The functions \tilde{H} corresponding to the *smoothing by K 's filters* are graphed next, for various values of K and ΔT .

the 'smoothing
by K 's' filters
are low-pass filters

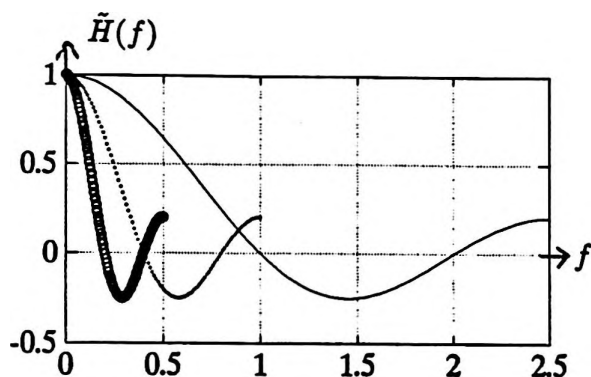
The graph below illustrates why the *smoothing by K 's filters* are commonly referred to as *low-pass filters*: they tend to pass low frequencies ($\tilde{H}(f)$ is close to 1) and suppress high frequencies ($\tilde{H}(f)$ is close to 0).



smoothing by 3's filter, '—'
smoothing by 5's filter, '·'
smoothing by 7's filter, 'o'
smoothing by 11's filter, 'x'

*smaller sampling
time allows
detection of
higher frequencies*

The graph below illustrates that a smaller sampling time ΔT allows detection of higher frequencies (smaller periods). For example, the function \tilde{H} corresponding to $\Delta T = 0.2$ gives information about frequencies in the interval $[0, 2.5]$, whereas when $\Delta T = 1$, information is only obtained on the interval $[0, 0.5]$.



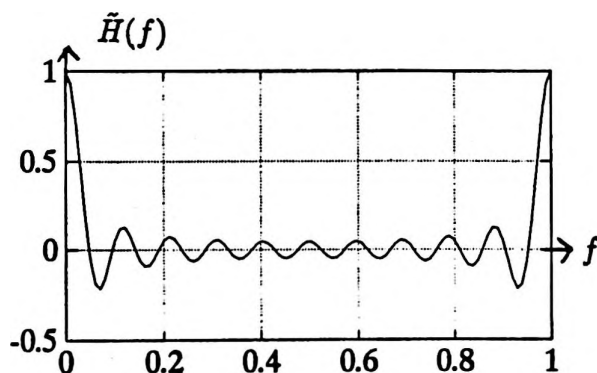
smoothing by 5's filter

$\Delta T = 0.2$, '—'

$\Delta T = 0.5$, '.'

$\Delta T = 1$, 'o'

When K is large, the *smoothing by K 's filter* does a better job of erasing high frequencies. For example, the graph below shows one full period of the transfer function corresponding to the *smoothing by 21's filter*, with sample spacing $\Delta T = 1$.



ONE FULL PERIOD OF THE
SMOOTHING BY 21's
TRANSFER FUNCTION,
SPACING $\Delta T = 1$

EXAMPLE
*understanding
 the information
 given by the
 transfer function*

The transfer function for the *smoothing by 5's* filter, $\Delta T = 1$, is graphed below. Observe that $\tilde{H}(.2) = 0$, and $\tilde{H}(.1) \approx .65$.

Construct a list (y_n) by evaluating the function

$$f(t) = \sin(2\pi(.2)t) + 4 \cos(2\pi(.1)t)$$

at each number in the uniform time list ($\Delta T = 1$)

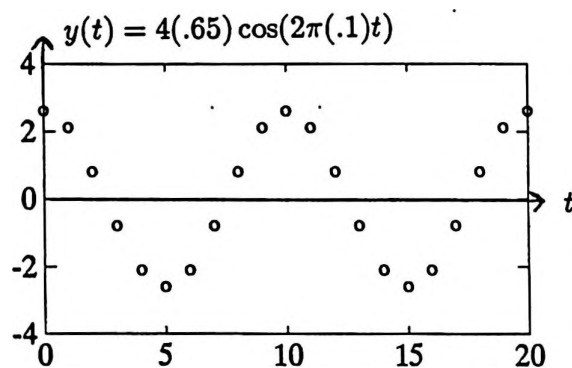
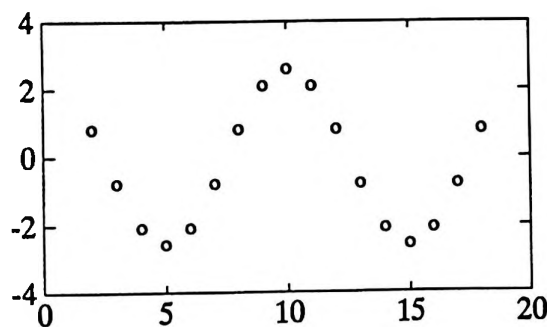
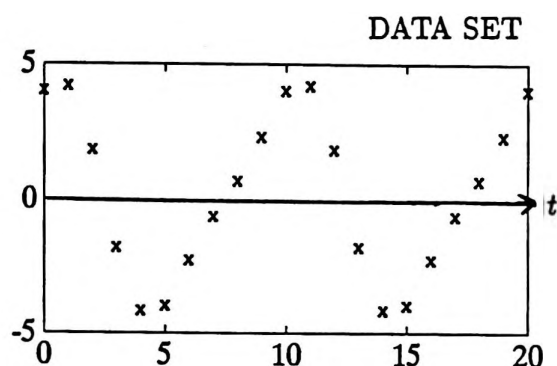
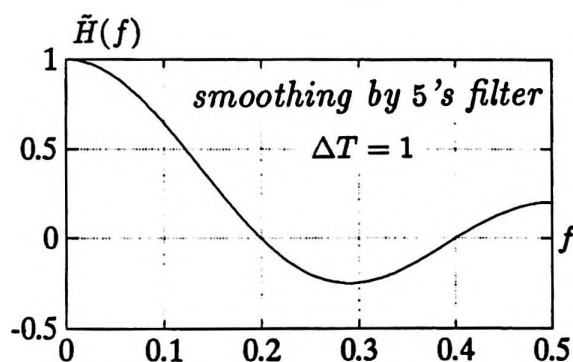
$$(0, 1, 2, \dots, 20).$$

The function f has a component of cyclic frequency 0.2 (period 5), amplitude 1; and a component of cyclic frequency 0.1 (period 10), amplitude 4.

When the list (y_n) is input to the *smoothing by 5's* filter, the period-5 component should emerge, scaled by $\tilde{H}(.2) = 0$; that is, it should be completely erased.

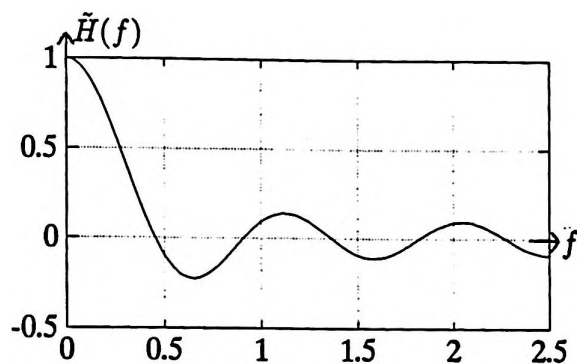
The period-10 component should emerge, scaled by $\tilde{H}(.1) \approx 0.65$; thus, its new amplitude should be approximately $(.65)(4) = 2.6$.

The graphs below illustrate that the filter behaves as dictated by its transfer function.



AFTER FILTERING WITH
 THE SMOOTHING BY 5's FILTER

EXAMPLE As a second example, the transfer function for the *smoothing by 11's filter*, with $\Delta T = 0.2$, is graphed below. Observe that $\tilde{H}(.2) \approx 0.7$.



smoothing by 11's filter

$$\Delta T = 0.2$$

Construct a list (y_n) by evaluating the function

$$f(t) = \sin(2\pi(.2)t)$$

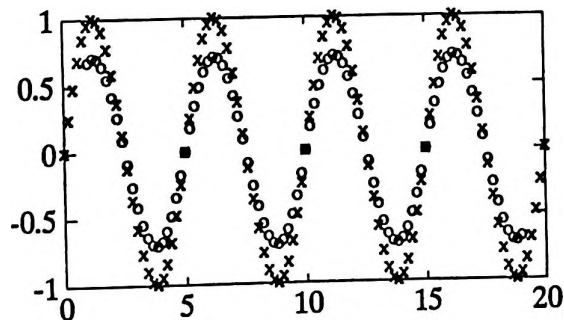
at each number in the uniform time list ($\Delta T = 0.2$)

$$(0, 0.2, 0.4, \dots, 20).$$

The function f has cyclic frequency 0.2 (period 5), and amplitude 1.

When the list (y_n) is input to the *smoothing by 11's filter*, it should emerge, scaled by $\tilde{H}(.2) \approx 0.7$.

The graph below illustrates that the filter behaves as dictated by its transfer function.



DATA SET, 'x'

FILTERED DATA, 'o'

A MATLAB program for finding the transfer function corresponding to a symmetric nonrecursive filter, and spacing ΔT , is given next.

MATLAB IMPLEMENTATION

Finding the Transfer Function for a Symmetric Nonrecursive Filter

MATLAB function `transfct(C,dT)` The MATLAB function `transfct`, with source code given below, computes the transfer function \tilde{H} corresponding to a symmetric nonrecursive filter

$$f_n = \sum_{k=-K}^K c_k y_{n-k}, \quad c_k = c_{-k} \quad \text{for } k = 1, \dots, K,$$

and spacing ΔT .

To use the function, type:

```
[Hf,f] = transfct(C,dT);
```

The transfer function is then plotted with the command:

```
plot(f,Hf)
```

REQUIRED INPUTS

The function requires the following inputs:

- The $K + 1$ distinct filter coefficients,

$$c_0, c_1, \dots, c_K.$$

These must be stored in the (row or column) vector c .

- The spacing ΔT , where $\Delta T > 0$; this is given by the MATLAB variable `dT`.

OUTPUTS

The program outputs two column vectors, Hf and f .

The vector f contains 100 equally-spaced values from the interval $[0, \frac{1}{2\Delta T}]$.

The vector Hf contains the values $\tilde{H}(f)$.

source code

The source code for the function `transfct` is given next:

```
% This computes the transfer function for a symmetric nonrecursive filter
function [Hf,f] = transfct(C,dT)
% C is a vector containing the filter coefficients
% [c0 c1 c2 ... cK]
% Get 100 points, equally spaced in the period [0,1/(2*dT)]
f = linspace(0,1/(2*dT),100)';
K = length(C) - 1;
Hf = zeros(f);
for k = 1:K
    Hf = Hf + C(k+1)*cos(2*pi*k*dT*f);
end
Hf = C(1) + 2*Hf;
```

The following diary of an actual MATLAB session illustrates the use of `transfct`. The function `nonrec`, for applying a non-recursive filter, was discussed in Section 3.1.

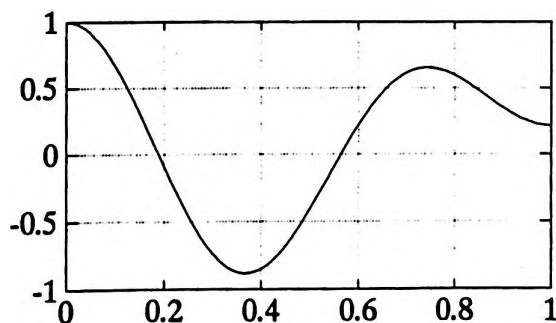
MATLAB EXAMPLE

```
% Construct a transfer function:
FC = [3 2.6 -1 1 -1 2.6 3]*(1/10.2);
FC = FC';
% Since the filter coefficients sum to 1, constants are passed perfectly.
% C must contain the distinct coefficients c0 c1 c2 c3 :
C = [1 -1 2.6 3]*(1/10.2);
% Find and graph the transfer function for spacing 0.5 :
[Hf,f] = transfct(C,.5);
plot(f,Hf)
grid
% Inspect the values of the transfer function:
[f Hf]

ans =
```

0	1.0000
0.0101	0.9964
0.0202	0.9857
0.0303	0.9678
⋮	⋮
0.1111	0.5984
0.1212	0.5293
0.1313	0.4567
⋮	⋮
0.1717	0.1426
0.1818	0.0612
0.1919	-0.0202
0.2020	-0.1009
0.2121	-0.1803
⋮	⋮
0.5455	-0.1188
0.5556	-0.0529
0.5657	0.0124
0.5758	0.0763

```
% Observe that frequencies of approximately .2 and .55 are 'erased'.
% Frequency .12 is scaled by .5 .
% Construct a 'known unknown' for spacing .5 :
t = [0:.5:20]';
y = 4 + 2*cos(2*pi*.2*(t-7)) - 3*sin(2*pi*.55*(t+8)) - cos(2*pi*.12*t);
```

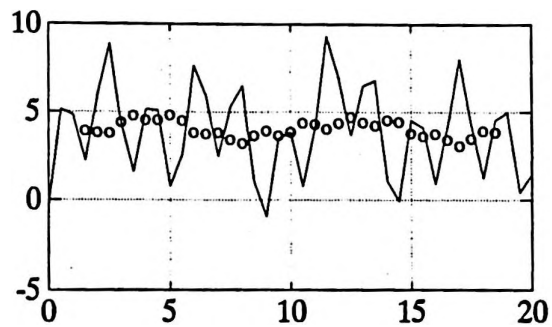


```
plot(t,y)
grid
hold
```

Current plot held

```
% Now, filter the data set:
[fil,tfil] = nonrec([t y],FC);
plot(tfil,fil,'o')
```

% As expected, mainly the frequency .12 component emerges!



3.3 Designing and Improving Filters

Introduction

The previous section investigated the problem: given a symmetric nonrecursive filter, find its transfer function. The current section shows how to go from a desired transfer function to a symmetric nonrecursive filter. The Fourier series of the desired transfer function plays an important role in this process.

complex Fourier series

Continuous Fourier series results were reviewed in Section 1.6. For the current purpose, it is easiest to employ the equivalent *complex* form of the Fourier series. The relevant results are summarized below. The interested reader is referred to [Ham, 95–99] for additional information.

COMPLEX FOURIER SERIES

Let g be a periodic real-valued function of one real variable. Suppose that g is piecewise continuous on \mathbf{R} , and has fundamental period P .

The *complex Fourier series* of g , denoted by $\text{Four}(g)$, is the infinite sum

$$\text{Four}(g)(t) := \sum_{k=-\infty}^{\infty} C_k e^{i \frac{2\pi k t}{P}},$$

where

$$C_k = \frac{1}{P} \int_P e^{-i \frac{2\pi k t}{P}} g(t) dt.$$

The coefficients C_k are related to the coefficients a_k and b_k (page 81) by

$$C_k = \frac{a_{(-k)} + ib_{(-k)}}{2} \quad \text{for } k < 0,$$

$$C_0 = \frac{a_0}{2},$$

$$C_k = \frac{a_k - ib_k}{2} \quad \text{for } k > 0.$$

if g is even,
then all
 C_k are real, and
 $C_k = C_{(-k)}$
for all k

If g is an even function ($g(-t) = g(t)$ for all t), then $b_k = 0$ for $1 \leq k < \infty$, and hence

$$C_k = \begin{cases} \frac{a_{(-k)}}{2} & \text{for } k < 0 \\ \frac{a_k}{2} & \text{for } k \geq 0 \end{cases}$$

In particular, if g is even, then all values of C_k are real numbers, and $C_k = C_{(-k)}$ for all integers k .

rewriting the
transfer function
 \tilde{H}

In order to compare a transfer function \tilde{H} (as a function of cyclic frequency) with its complex Fourier series representation, it is rewritten as follows:

$$\begin{aligned}\tilde{H}(f) &:= \sum_{k=-K}^K c_k e^{-i2\pi k f \Delta T} = \sum_{l=K}^{-K} c_{(-l)} e^{i2\pi l f \Delta T} \quad (l := -k) \\ &= \sum_{k=-K}^K c_{(-k)} e^{i2\pi k f \Delta T} .\end{aligned}\quad (1)$$

the complex
Fourier series
for \tilde{H}

Recall from the previous section that \tilde{H} has period $\frac{1}{\Delta T}$. Therefore, the complex Fourier series for \tilde{H} is given by

$$\text{Four}(\tilde{H})(f) = \sum_{k=-\infty}^{\infty} C_k e^{i \frac{2\pi k f}{1/\Delta T}} = \sum_{k=-\infty}^{\infty} C_k e^{i2\pi k f \Delta T} . \quad (2)$$

By comparing (1) and (2), it is apparent that the truncated Fourier series will approximate the transfer function, provided that $C_k = c_{(-k)}$ for $k = -K, \dots, K$.

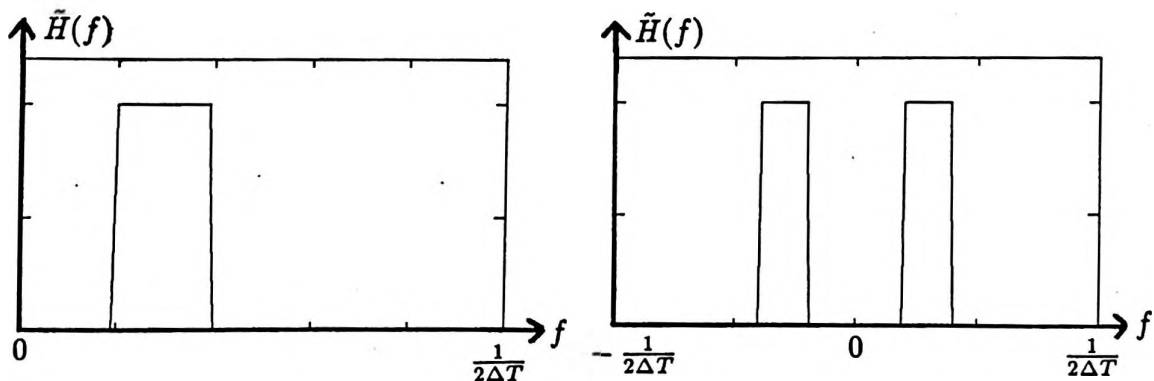
The procedure for finding filter coefficients from a given (desired) transfer function is summarized next:

PROCEDURE:
finding
filter coefficients
from a given
transfer function;
extend \tilde{H}

- Let ΔT be a given positive number, representing the uniform spacing in the time list of a data set to be processed. Let \tilde{H} denote a desired transfer function on the interval of frequencies $[0, \frac{1}{2\Delta T}]$.
- Extend \tilde{H} to $[-\frac{1}{2\Delta T}, \frac{1}{2\Delta T}]$ via

$$\tilde{H}(-t) = \tilde{H}(t) \quad \text{for } t \in [0, \frac{1}{2\Delta T}] .$$

(See the diagram below.) Call this extension by the same name.



*the extension
is an even function*

- Extend \tilde{H} periodically to all of \mathbf{R} . (See the diagram below.) That is, define

$$\tilde{H}\left(t + k\left(\frac{1}{\Delta T}\right)\right) := \tilde{H}(t)$$

for all integers k , and for all $t \in [-\frac{1}{2\Delta T}, \frac{1}{2\Delta T}]$. Call this extension by the same name. The resulting function \tilde{H} is even, with fundamental period $\frac{1}{\Delta T}$.

computing C_k

- Since \tilde{H} is even, its complex Fourier series coefficients C_k are real numbers, and $C_k = C_{(-k)}$ for all integers k . For $k \geq 0$,

$$C_k = \frac{a_k}{2},$$

where

$$\begin{aligned} a_k &= \frac{2}{P} \int_P \tilde{H}(f) \cos \frac{2\pi k f}{P} df && \text{(see p. 81)} \\ &= 2\Delta T \int_{-\frac{1}{2\Delta T}}^{\frac{1}{2\Delta T}} \tilde{H}(f) \cos(2\pi k f \Delta T) df && (P = \frac{1}{\Delta T}) \\ &= 4\Delta T \int_0^{\frac{1}{2\Delta T}} \tilde{H}(f) \cos(2\pi k f \Delta T) df && (\tilde{H} \text{ is even}) \end{aligned} \quad (3)$$

*computing the
filter coefficients*

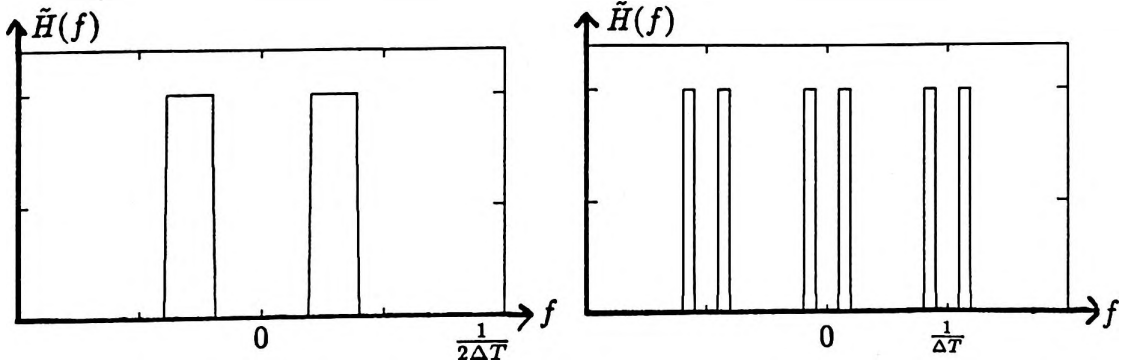
- For a positive integer K , the coefficients c_k given by

$$c_k = c_{-k} = C_k \quad \text{for } k = 0, \dots, K$$

will give a symmetric nonrecursive filter

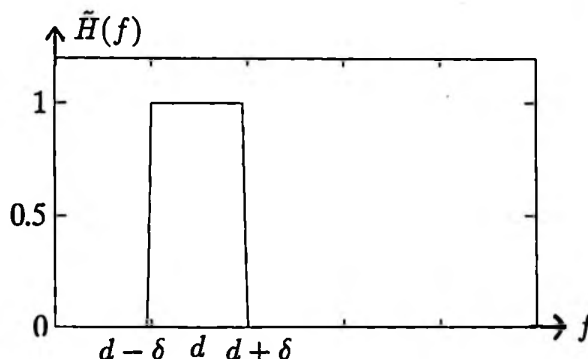
$$f_n = \sum_{k=-K}^K c_k y_{n-k}$$

with a transfer function that approximates $\tilde{H}(f)$. In general, the greater the value of K , the better the approximation.



EXAMPLE
designing a
band-pass filter

The procedure just described is illustrated next. Let $\Delta T = 0.5$, so that $\frac{1}{2\Delta T} = 1$. A transfer function is desired that will pass the frequencies in an interval $[d - \delta, d + \delta] \subset [0, \frac{1}{2\Delta T}]$, and suppress all other frequencies. Such a filter is called a *band-pass filter*. The desired transfer function is shown below on $[0, \frac{1}{2\Delta T}]$.



Using (3),

$$\begin{aligned}
 a_k &= 4\Delta T \int_0^{\frac{1}{2\Delta T}} \tilde{H}(f) \cos(2\pi k f \Delta T) df \\
 &= 4(0.5) \int_0^1 \tilde{H}(f) \cos(2\pi k f (0.5)) df \\
 &= 2 \int_{d-\delta}^{d+\delta} (1) \cos(k\pi f) df \\
 &= \frac{2}{k\pi} [\sin(k\pi(d+\delta)) - \sin(k\pi(d-\delta))] , \text{ for } k > 0 .
 \end{aligned}$$

Also,

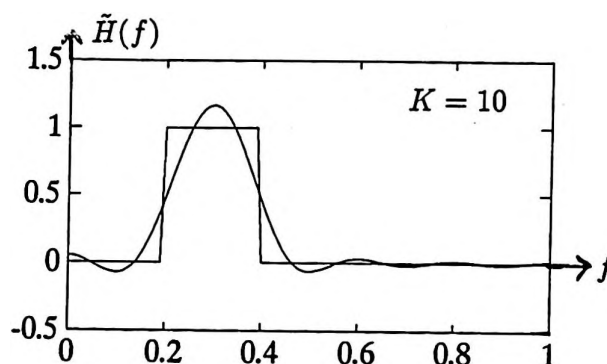
$$a_0 = 4\Delta T \int_{d-\delta}^{d+\delta} (1) df = 4(0.5)(2\delta) = 4\delta .$$

$K = 10$

Let $d = 0.3$, $\delta = 0.1$, and $K = 10$. The resulting symmetric nonrecursive filter has coefficients:

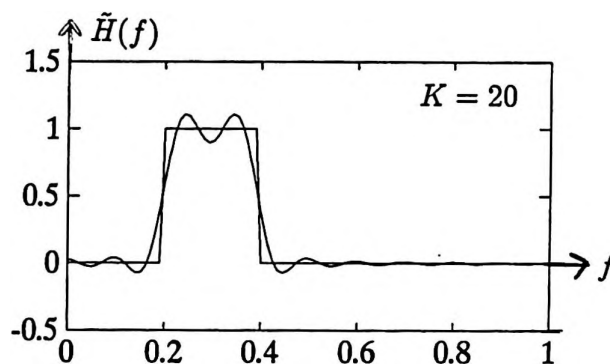
$$\begin{aligned}
 (c_0, c_1, \dots, c_{10}) &= \left(\frac{a_0}{2}, \frac{a_1}{2}, \dots, \frac{a_{10}}{2} \right) \\
 &\approx (.2, .1156, -.0578, -.1633, -.1225, 0, .0816, .0700, .0145, -.0128, 0) .
 \end{aligned}$$

The transfer function for this filter is shown below, superimposed with the 'ideal' transfer function.



$K = 20$

Now, increase K to 20, keeping all other parameters the same. The transfer function of the resulting symmetric nonrecursive filter is shown below, superimposed with the 'ideal' transfer function.



*improving the
transfer function;
Lanczos
smoothing*

The 'ripples' that appear in the transfer functions can be 'smoothed' by a process called *Lanczos smoothing*. Lanczos is credited with observing that the 'ripple' in a truncated Fourier series has approximately the period of the last term kept in the series. He argued that smoothing the partial sum by integrating (averaging) over this period would remove the main effects of the ripple. This procedure is discussed next. The interested reader is referred to [Ham, 109–112] for additional information.

the truncation,
 $T(f)$

Consider the truncation

$$T(f) := \sum_{k=-K}^K C_k e^{i2\pi k f \Delta T}$$

of the infinite sum

$$\text{Four}(\tilde{H})(f) := \sum_{k=-\infty}^{\infty} C_k e^{i2\pi k f \Delta T}.$$

The last terms kept correspond to indices $k = K$ and $k = -K$, with corresponding period $\frac{1}{K\Delta T}$. Define a 'smoothed' function $S(f)$, obtained by replacing the number $T(f)$ with the average value of the function T over an interval of length $\frac{1}{K\Delta T}$ centered at f :

$$\begin{aligned} S(f) &:= \frac{1}{1/(K\Delta T)} \int_{f-\frac{1}{2K\Delta T}}^{f+\frac{1}{2K\Delta T}} T(f) df \\ &= \frac{1}{1/(K\Delta T)} \int_{f-\frac{1}{2K\Delta T}}^{f+\frac{1}{2K\Delta T}} \sum_{k=-K}^K C_k e^{i2\pi k f \Delta T} df \\ &= K\Delta T \sum_{k=-K}^K C_k \int_{f-\frac{1}{2K\Delta T}}^{f+\frac{1}{2K\Delta T}} e^{i2\pi k f \Delta T} df \\ &= K\Delta T \sum_{k=-K}^K \frac{C_k}{\pi k \Delta T} e^{i2\pi k f \Delta T} \left[\frac{e^{i(\pi k/K)} - e^{-i(\pi k/K)}}{2i} \right] \\ &= K \sum_{k=-K}^K \frac{C_k}{\pi k} e^{i2\pi k f \Delta T} \sin(\pi k/K) \\ &= \sum_{k=-K}^K C_k \left[\frac{\sin(\pi k/K)}{\pi k/K} \right] e^{i2\pi k f \Delta T}. \end{aligned}$$

sigma factors

Observe that in the resulting function $S(f)$, the k^{th} term of $T(f)$ has been scaled by the number

$$\frac{\sin(\pi k/K)}{\pi k/K}.$$

These scaling factors are commonly called the *sigma factors*. Observe that

$$\frac{\sin(\pi(-k)/K)}{\pi(-k)/K} = \frac{-\sin(\pi k/K)}{-\pi k/K} = \frac{\sin(\pi k/K)}{\pi k/K}.$$

Also, when $k = 0$, the sigma factor is defined to equal 1, since

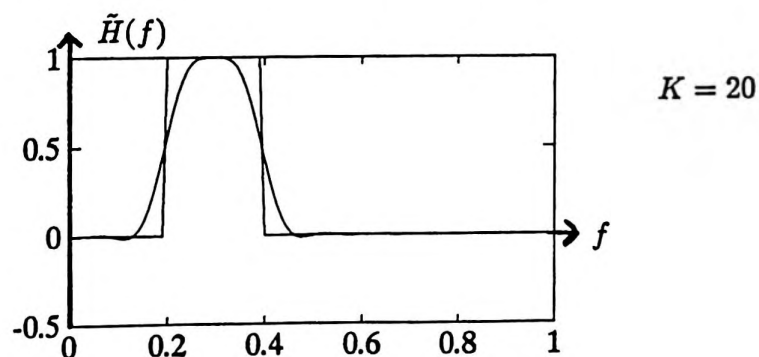
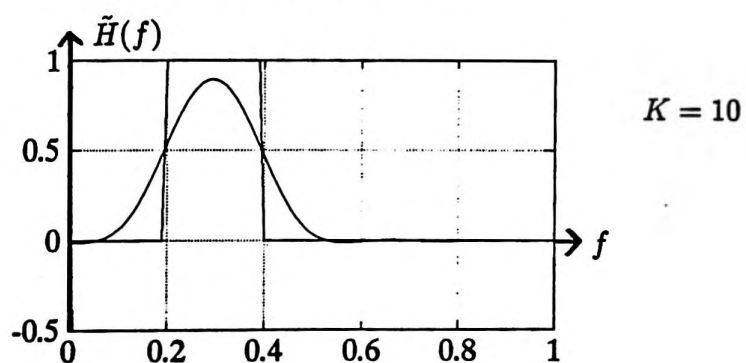
$$\lim_{k \rightarrow 0} \frac{\sin(\pi k/K)}{\pi k/K} = 1.$$

The distinct values of the sigma factors are shown below, when $K = 10$:

k	sigma factor
0	1
1	0.9836
2	0.9355
3	0.8584
4	0.7568
5	0.6366
6	0.5046
7	0.3679
8	0.2339
9	0.1093
10	0

EXAMPLE
Lanczos
smoothing

The transfer functions of the previous example, after Lanczos smoothing is applied, become:



MATLAB IMPLEMENTATION

Finding a Symmetric Nonrecursive Filter Corresponding to a Desired Transfer Function

MATLAB function findfil

The MATLAB function `findfil`, with source code given on the following page, computes the coefficients of a symmetric nonrecursive filter corresponding to a desired transfer function.

To use the function, type:

```
[c,Hf,f] = findfil(tH,H,K,dT,smooth);
```

REQUIRED INPUTS

The required inputs are:

- A positive number ΔT that represents the uniform spacing in the time list of a data set to be processed. The MATLAB variable `dT` contains the desired value of ΔT .
- A description of the desired transfer function on the interval $[0, \frac{1}{2\Delta T}]$. The description must be contained in two column vectors, `tH` and `H`, of equal length: the column vector `tH` contains equally-spaced entries from the interval $[0, \frac{1}{2\Delta T}]$, and `H` contains the corresponding desired values of the transfer function.
- K is a positive integer that gives the last term to be kept in the truncated Fourier series. The resulting nonrecursive filter will have $K + 1$ distinct filter coefficients, c_0, c_1, \dots, c_K .

OPTIONAL INPUT

- If Lanczos smoothing is desired, set `smooth = 1`. Any other value for `smooth`, or removing this variable from the input list, gives an unsmoothed filter.

OUTPUTS

The function outputs three column vectors, `c`, `Hf`, and `f`.

- The column vector `c` contains the $K + 1$ distinct coefficients c_0, c_1, \dots, c_K of the corresponding nonrecursive filter.
- The column vectors `f` and `Hf` contain the information necessary to plot the transfer function corresponding to the nonrecursive filter (which is an approximation to the desired transfer function). This approximation can be plotted, superimposed with the actual transfer function, via the commands:

```
plot(f,Hf)
hold
plot(tH,H)
```

source code

The source code for the function `findfil` is given next:

```

function [C,Hf,f] = findfil(tH,H,K,dT,smooth)
if nargin==4
    smooth = 0;
end
N = length(tH);
P = tH(N);
C = zeros(K+1,1);
C(1) = (1/N)*sum(H);
for k = 2:(K+1)
    tvec = (2*pi*(k-1)*dT)*tH;
    cosv = cos(tvec);
    C(k) = (1/N)*sum(H .* cosv);
end
if smooth==1
    k = [1:K]';
    k = pi*k/K;
    sigma = sin(k) ./ k;
    C(2:K+1) = sigma .* C(2:K+1);
end
[Hf,f] = transfct(C,dT);

```

MATLAB
function
`symtofc`

The MATLAB function `symtofc` (*'symmetric to filter coefficients'*) converts the $K+1$ distinct coefficients c_0, c_1, \dots, c_K from a non-recursive symmetric filter, to the full set of coefficients

$$c_{-K}, \dots, c_{-1}, c_0, c_1, \dots, c_K$$

(where $c_{-k} = c_k$), for use in the function `nonrec`.

To use the function, type:

```
FC = symtofc(C);
```

```

function FC = symtofc(C);
K = length(C) - 1;
FC = flipud(C(2:(K+1)));
FC = [FC;C];

```

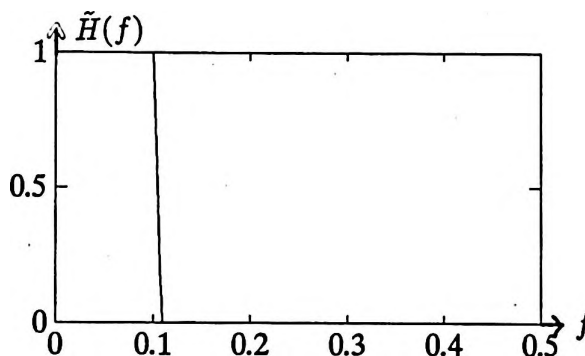
INPUT
and
OUTPUT

The input is a column vector c containing the coefficients c_0, c_1, \dots, c_K .
The output is a column vector FC containing the full set of coefficients $c_{-K}, \dots, c_{-1}, c_0, c_1, \dots, c_K$.

EXAMPLE
*designing a
 low-pass
 filter*

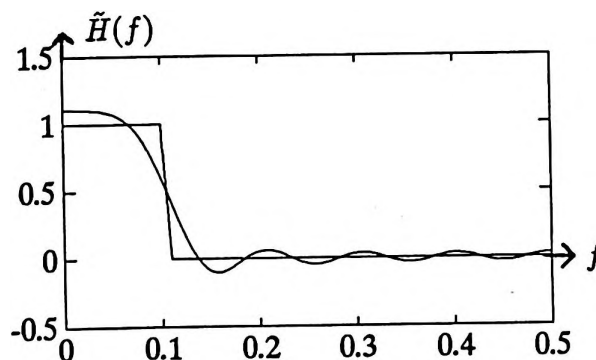
The following diary of an actual MATLAB session illustrates the use of the functions `findfil` and `symtofc`. Here, a low-pass filter (which passes low frequencies, and suppresses high frequencies) is designed, corresponding to spacing $\Delta T = 1$. The designed filter is then used to process a 'known unknown'.

```
% Construct the desired transfer function:
tH1 = [0:.01:.10]';
tH2 = [.11:.01:.5]';
y1 = ones(tH1);
y2 = zeros(tH2);
tH = [tH1;tH2];
H = [y1;y2];
% Graph the desired transfer function:
plot(tH,H)
```



```
% Find a corresponding unsmoothed filter, dT = 1, K = 10:
[C,Hf,f] = findfil(tH,H,10,1);
% Plot the transfer function of the corresponding filter,
% superimposed with the ideal transfer filter:
plot(f,Hf)
hold
Current plot held

plot(tH,H)
hold
Current plot released
```

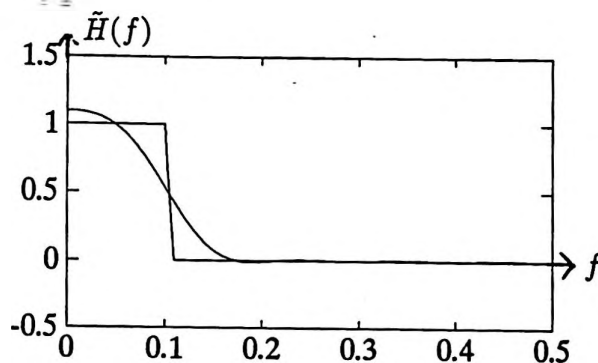


```
% Now, find a corresponding smoothed filter, dT = 1, K = 10:
[C,Hf,f] = findfil(tH,H,10,1,1);
plot(f,Hf)
hold
```

Current plot held

```
plot(tH,H)
hold
```

Current plot released



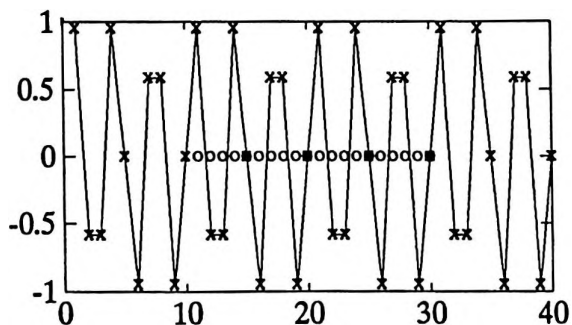
```
C
% Here are the smoothed filter coefficients c0,c1,...,c10 :
C =
```

```
0.2157
0.1978
0.1506
0.0905
0.0359
0.0000
-0.0143
-0.0129
-0.0055
-0.0002
0.0000
```

```
t = [1:40]';
y = sin(2*pi*.3*t);
plot(t,y)
hold
```

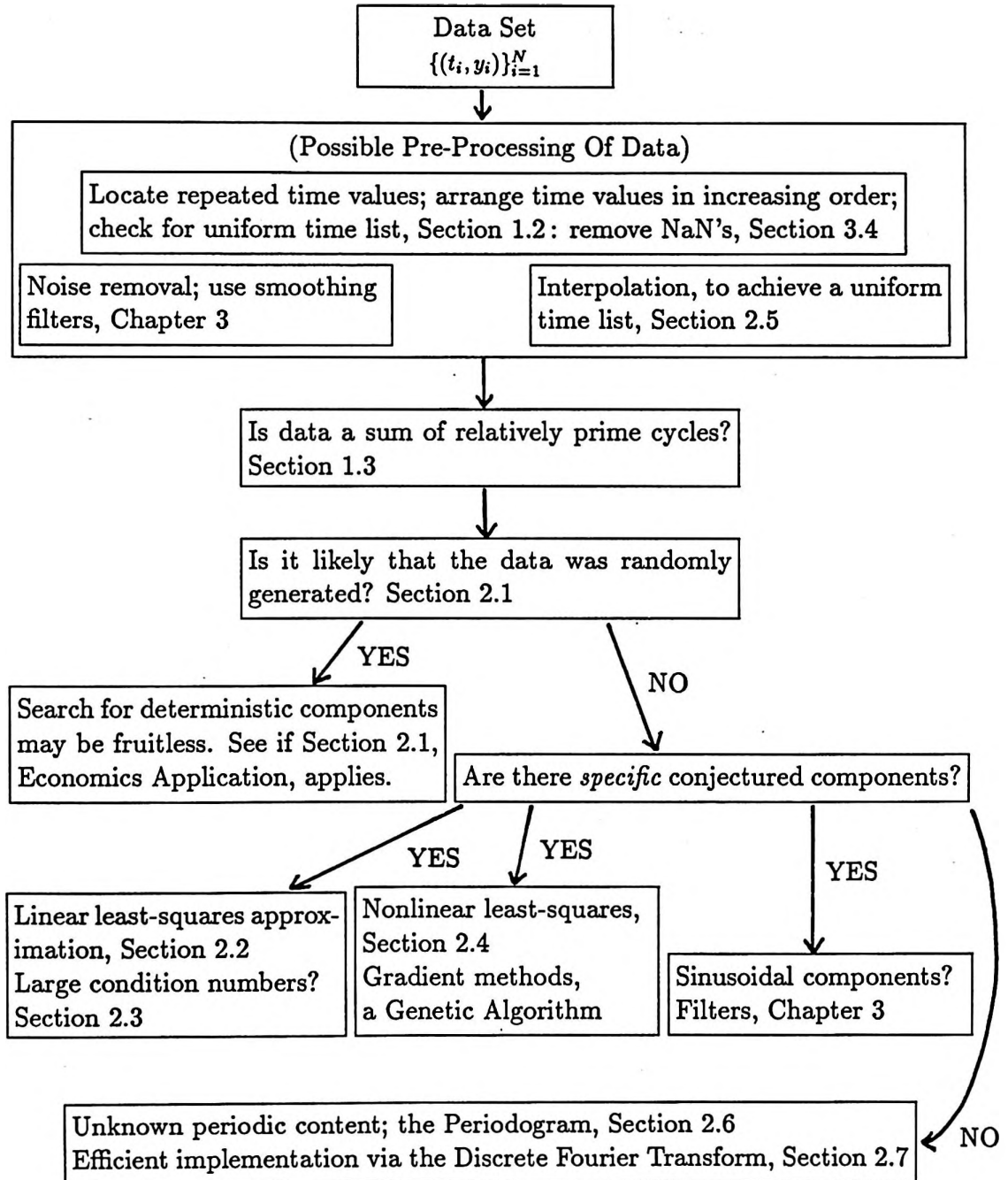
Current plot held

```
plot(t,y,'x')
FC = symtofc(C);
D = [t y];
[f,tf] = nonrec(D,FC);
plot(tf,f,'o')
```



3.4 Identifying Hidden Periodicities: Approach and Examples

The following flowchart, which also appears in the preliminary pages of this dissertation, suggests a strategy for analyzing a data set. This section contains examples illustrating the application of the procedure presented here.



missing values
in a data set;
NaN,
Not A Number

Data sets sometimes have missing values. Such entries can be represented via the MATLAB matrix entry NaN ('Not a Number'). However, any arithmetic calculation using a NaN yields NaN as the final result. Therefore, NaN entries must be removed or replaced by interpolated values before any processing can be done on a data set.

The following MATLAB function can be used to remove ordered pairs of the form (t, NaN) from a data set.

MATLAB FUNCTION delnan	The MATLAB function <code>delnan</code> ('delete not a number'), with source code given below, removes the NaN values from a data set D. To use the function, type: <code>Dr = delnan(D);</code>
INPUT and OUTPUT	The input is a matrix D; the first column of D contains the time values of data points, and the second column contains the corresponding data values. Thus, each row of D is a data point. The output is a matrix Dr that is identical to D, except that rows of the form (t, NaN) have been removed.

source code
for delnan

The source code for `delnan` is given below:

```
function Dr = delnan(D)
% This function removes the NaNs from a data set, D
Dcol = D(:,2);
i = find(isnan(Dcol));
Dr = D;
for j = 1:(length(i));
    Dr( i(j),: ) = [];
    i = i-1;
end
```

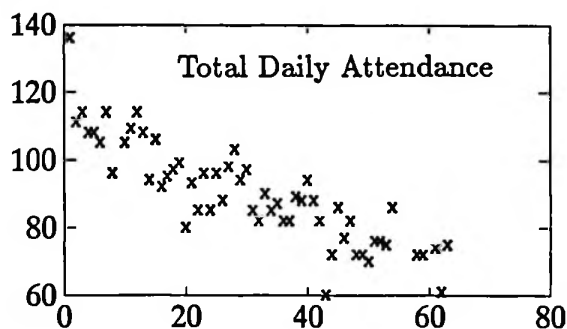
The use of `delnan` is illustrated in the next example.

EXAMPLE 1 The data set graphed below gives the total daily attendance in three undergraduate classes taught by the author during the Fall 1993 semester at Idaho State University. Two sections of Math 111 (Algebra) and one section of Math 120 (Calculus for the Life and Social Sciences) were taught. Each class met four days per week: Monday, Tuesday, Wednesday, and Friday. The data values recorded as NaN represent days when classes did not meet. This data set is stored in a MATLAB matrix D . The symbols M, T, W, F (the days of the week that classes met) are shown below only for the reader's information, and are not included in the matrix D .

$D =$

M	1	136
T	2	111
W	3	114
F	4	108
M	5	108
T	6	105
W	7	114
F	8	96
M	9	NaN
T	10	105
W	11	109
F	12	114
M	13	108
T	14	94
W	15	106
	:	:

Data Set with 'Not a Number' Entries



*remove NaN's
from the data set;*
Dr
tr
yr

First, the MATLAB function `delnan` is used to remove the `NaN` entries from `D`. The resulting matrix is named `Dr` ('r' for 'removed'). The first column of `Dr` is named `tr`, and the second column is named `yr`.

Observe that row 9 of `D`, which contained a `NaN` entry, has been removed.

```
Dr = delnan(D);
tr = Dr(:,1);
yr = Dr(:,2);
```

Dr =

```

1    136
2    111
3    114
4    108
5    108
6    105
7    114
8     96
10   105
11   109
12   114
13   108
14    94
15   106

:     :
```

Data Set with 'Not a Number' Entries Removed

fit D_r
with a parabola

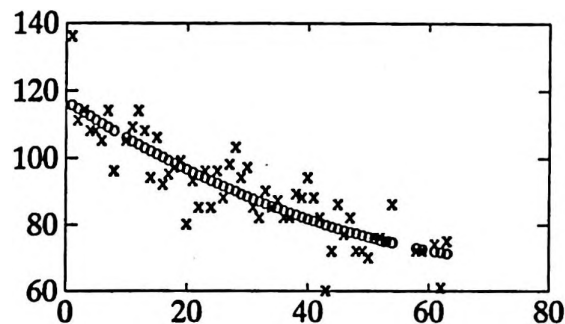
The 'downward trend' in the data is quite striking, so linear least-squares approximation techniques (p. 150) are used first to fit the matrix $D_r = [tr \ yr]$ with a parabola. The parabola component obtained is named y_1 :

```
f1 = ones(tr);
f2 = tr;
f3 = tr.^2;
X = [f1 f2 f3];
b = (X'*X) \ (X'*yr)

b =
    116.7717
     -1.1486
      0.0068

y1 = X*b;
plot(tr, yr, 'x', tr, y1, 'o')
error = (yr - y1)'*(yr - y1)

error =
    2.9694e+003
```



'Downward Trend' in the Data Set

Daily Attendance, $D_r = [tr \ yr]$, 'x'

Parabola Component, $[tr \ y_1]$, 'o'

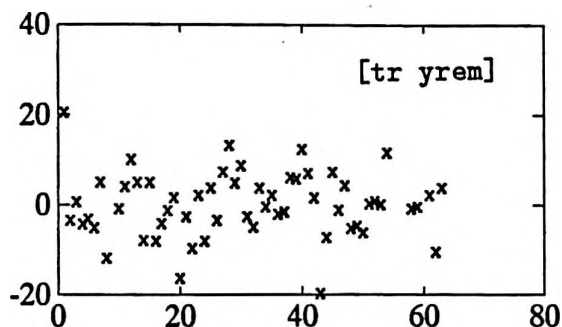
*subtract
parabola;
analyze the
remainder yrem*

The parabola component y_1 is subtracted from y_r ; the difference is named y_{rem} . The list y_{rem} is tested for random behavior using the Turning Point Test (p. 122):

```
yrem = yr - y1;
plot(tr,yrem,'x')
rand = tptest(yrem)
```

rand =

58.0000	0	38.6667	1.0000	43.0000	0.5320
:	:	:	:	:	:



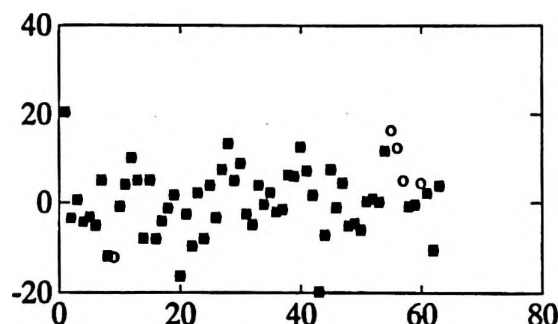
Parabola Component Removed

The information given in `rand` shows that if y_{rem} were generated by random means alone, then one would expect to see about 38.7 turning points. There are actually 43 turning points in y_{rem} . The probability that 43 or more turning points would occur, should the data be truly random, is about 53.2%. A decision is made to search for sinusoidal components.

use spline
interpolation

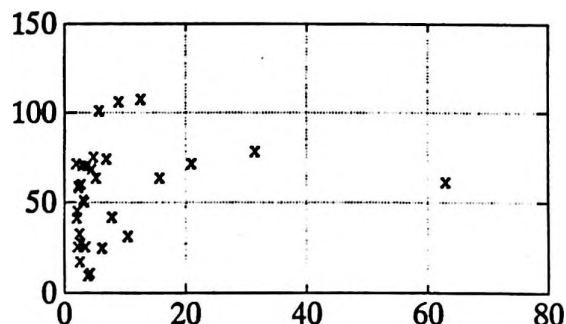
There are no obvious sinusoidal components in `yrem`. The periodogram (p. 241) is obtained as an initial data analysis tool. Since a uniform time list is required to find the periodogram, spline interpolation (p. 211) is used to interpolate the data set `[tr yrem]`. Then, the periodogram of the interpolated data set is computed. Both the interpolated data set and its periodogram are graphed below.

```
yremint = spline(tr,yrem,t);
plot(tr,yrem,'x')
hold
Current plot held
plot(t,yremint,'o')
hold
Current plot released
[per,sqrcoef] = pervfft([t yremint]);
plot(per,sqrcoef,'x')
```



Spline Interpolated Data Set, 'o'

Observe the overlapping of the symbols 'x' and 'o' at all points in the original data set. The points indicated by 'o' alone are the interpolates corresponding to the missing time values.



Periodogram corresponding to
Spline Interpolated Data Set

using a
genetic algorithm

A genetic algorithm (p. 188) is used to search for sinusoidal components in the range of periods from 2 to 63.

A good fit is found using sinusoids with periods 14.5, 5.5, and 32.5. Selected commands used in the fitting process are shown below. Also, the graph of the best fit is given.

```
G = genetic([tr yrem],2,63,.5,122,3,4);
```

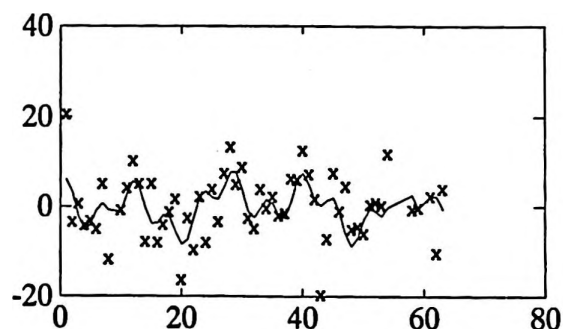
```
averror =  
2.6202e+003
```

```
averror =  
2.5993e+003
```

```
averror =  
2.6072e+003
```

```
averror =  
2.5881e+003
```

(Some of the
Scrolled
Information)



```
G(:,1:4)
```

Best Fit to yrem from Genetic Algorithm

```
ans =
```

1.0000	5.5000	32.5000	14.5000
2.0000	6.0000	14.0000	33.5000
3.0000	18.5000	14.0000	34.0000
4.0000	18.5000	14.0000	33.5000

```
G(:,5)
```

```
ans =
```

```
1.0e+003 *  
2.1007  
2.1230  
2.1992  
2.1951
```

```
[yb,row,per,coef] = bfitgen(tr,G);  
plot(tr,yrem,'x')  
hold  
Current plot held  
plot(tr,yb)
```

```
per
```

```
per =  
5.5000 32.5000 14.5000
```

```
coef
```

```
coef =  
A B C1 D1  
-0.1870 0.0191 3.1433 0.5401  
C2 D2 C3 D3  
0.0528 2.3873 -3.3723 2.4670
```

using gradient
methods to
improve the fit

Gradient methods (p.181) are used to improve the fit. After two iterations, a 'best' fit is found, with approximate periods 5.7, 33.0, and 13.6. Selected commands used are shown below. The graph of the best fit is given, as well as the graphs of each of the three sinusoidal components.

These three components have a reasonable interpretation: period 5.7 is slightly longer than a weekly component; period 13.6 represents the time between exams in the course (this component peaks at the exam dates); and period 34.5 is approximately half the semester (this component peaks at mid-semester).

```
[E,f] = nonlin([5.5 32.5 14.5],[tr yrem],.01,1);
```

```
E(25,1:11)
```

```
ans =
```

```
-0.6543    0.0260    5.7047    1.6267    3.4054  
33.0589    0.1360    2.4950    13.4513    0.0694    4.4829
```

```
E(25,12)
```

```
ans =
```

```
1.8584e+003
```

```
[E,f] = nonlin([5.7 33.1 13.5],[tr yrem],.001,1);
```

```
E(25,1:11)
```

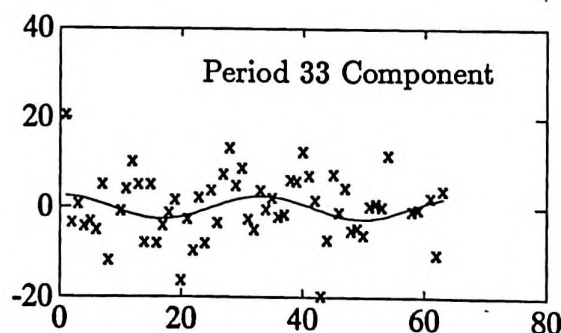
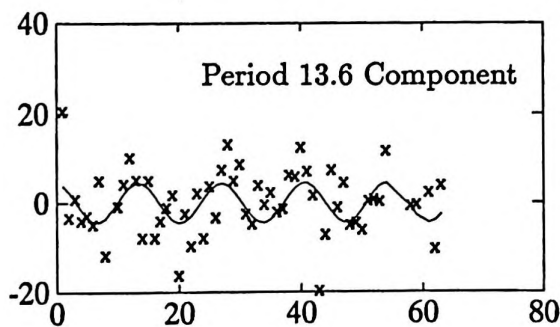
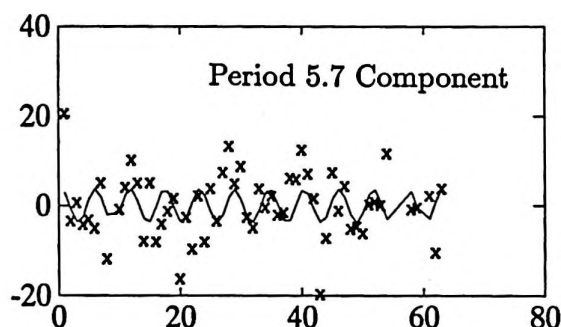
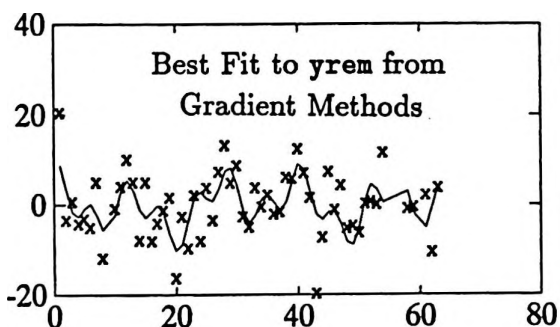
```
ans =
```

```
-0.5755    0.0252    5.7119    1.4574    3.4746  
33.0090    0.1377    2.5493    13.5910   -0.5250    4.4891
```

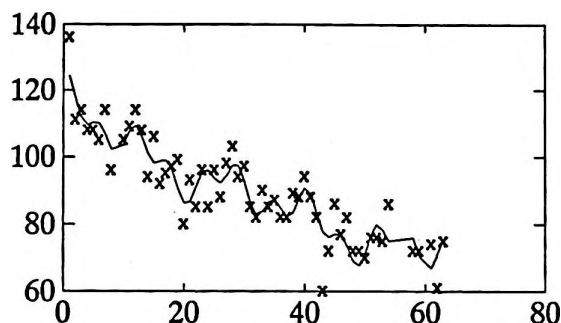
```
E(25,12)
```

```
ans =
```

```
1.8519e+003
```



The sum of the parabola component and the three sinusoidal components are shown with the original data:



EXAMPLE 2 The data set graphed below gives the daily balance in a checking account from 1/14/92 to 12/31/93. Both a 'point' graph and a 'line' graph of the data are shown.

There are several trends in the data, due to different types of employment for one contributor to the account:

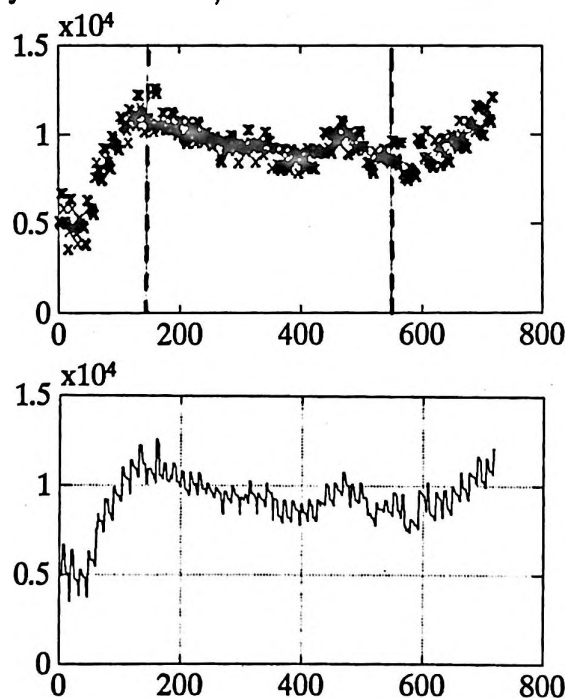
- From 1/14/92 to 5/22/92, one contributor was employed full-time, on a 9-month pay schedule.
- From 7/13/92 to 7/30/93, the same contributor was employed half-time, on a 12-month pay schedule.
- From 8/27/93 to 12/31/93, the same contributor was employed full-time, on a 12-month pay schedule.

These three periods are roughly delineated in the first graph below, by the dashed vertical lines.

The first column of `dlybal` is named `t`, and the second column is named `y`.

```
dlybal =
1.0e+004 *
0.0001    0.5142
0.0002    0.5034
0.0003    0.5016
0.0004    0.4948
0.0005    0.6677
0.0006    0.6677
0.0007    0.6677
0.0008    0.6677
0.0009    0.5839
0.0010    0.5143
.          .
.          .
.          .
```

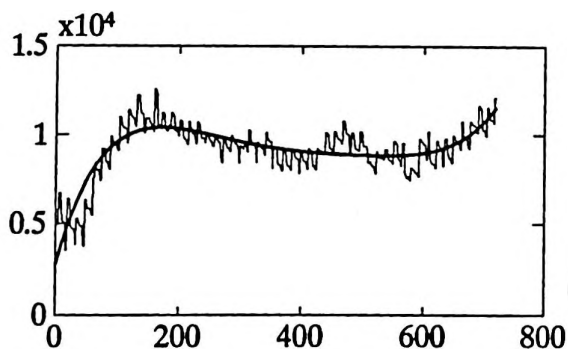
```
t = dlybal(:,1);
y = dlybal(:,2);
```



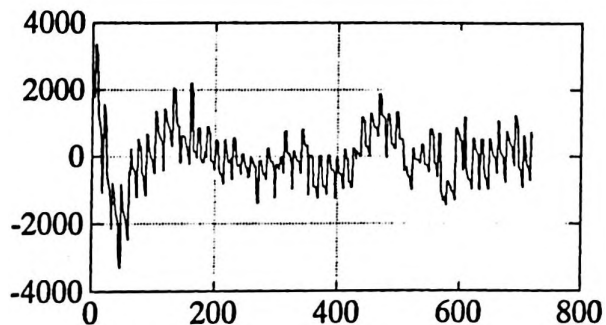
*least-squares
fit*

A least-squares polynomial fit is used to account for the different employment types. After some experimentation, the analyst decides that the best fit to account for these trends is obtained by using a fifth order polynomial. The polynomial fit is named `poly`, and the difference between `y` and `poly` is named `diff1`.

```
f1 = ones(t);
f2 = t;
f3 = t.^2;
f4 = t.^3;
f5 = t.^4;
f6 = t.^5;
X = [f1 f2 f3 f4 f5 f6];
b = (X'*X) \ (X'*y)
Warning: Matrix is close to singular or badly scaled.
Results may be inaccurate. RCOND = 7.998835e-030
poly = X*b;
plot(t,y,t,poly,'.')
diff1 = y - poly;
plot(t,diff1)
```



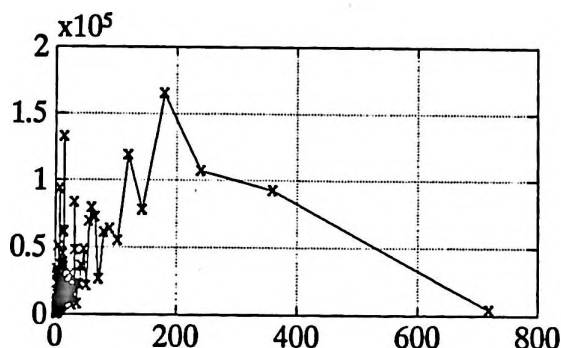
POLYNOMIAL FIT, `poly`



GRAPH OF `diff1`

*periodogram
of diff1*

As a preliminary analysis tool, the periodogram of *diff1* is found. Inspection of the matrix `[per sqcoef]` shows that the three highest peaks occur at periods 14, 120, and 180.



genetic algorithm

As another preliminary analysis tool, a genetic algorithm is applied to *diff1*. Due to memory constraints on the analyst's computer, only a very limited application is made. However, a component close to 180 does emerge in one of the 'best-fit' strings. Also, inspection of each population shows that period 365 appears in many 'high-fit' strings.

*gradient
methods*

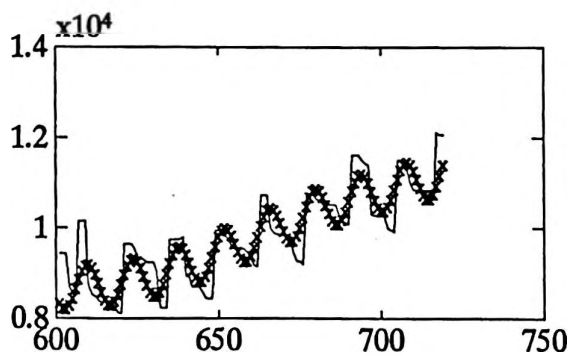
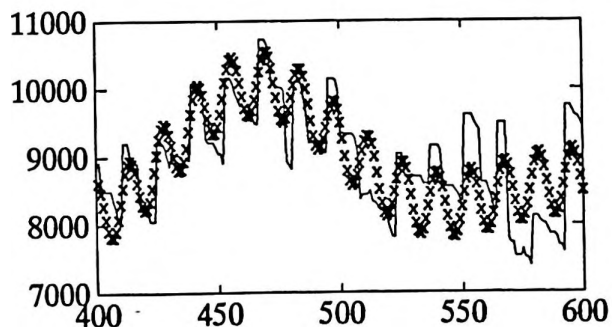
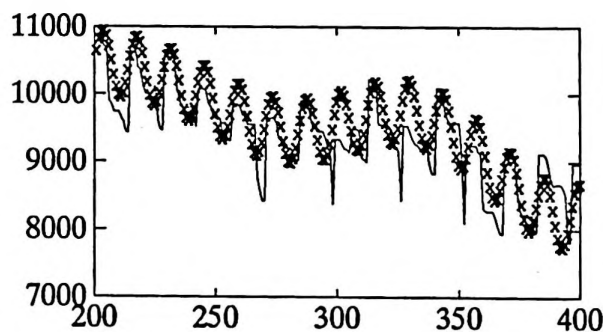
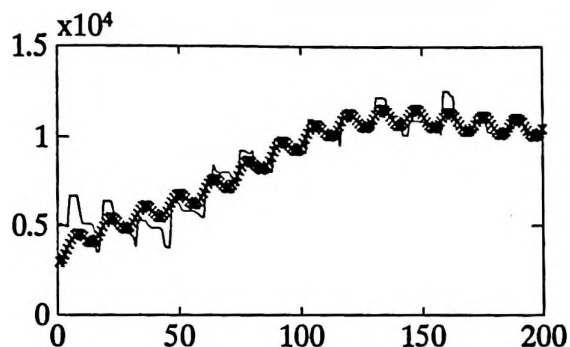
Based on these initial analyses, gradient methods are used to fit *diff1* with sinusoids of periods 14, 120, 180, and 365.

After two iterations of the MATLAB function `nonlin`, a best fit to *diff1* is obtained using periods of approximately 14, 116, 166, and 286. These are roughly 2-week, $\frac{1}{3}$ -year, $\frac{1}{2}$ -year, and $\frac{3}{4}$ -year components.

```
[E,app] = nonlin([14 120 180 365],[t diff1],.001,1);
E(24,1:14)
ans =
Columns 1 through 7
113.1623 -0.2060 13.9982 -165.2899 -449.1588 116.2768 -101.6867
Columns 8 through 14
362.8687 163.3660 -297.0426 431.4670 286.5257 -272.6909 -246.5112
[E,app] = nonlin([14 116 163 286],[t diff1],.0005,1);
E(24,1:14) ans =
Columns 1 through 7
129.0910 -0.2557 13.9984 -163.5705 -449.5768 115.9341 -74.3696
Columns 8 through 14
364.8528 165.7173 -368.5421 379.7280 286.4843 -274.9777 -227.2658
```

*the total
approximation*

By adding poly to the best fit obtained from `diff1`, the total approximation to the data is found. This approximation is graphed below using an 'x', superimposed over the actual data set (as a line graph). The graph is broken into 4 smaller pieces for easier readability.



Appendix 1

Mathematical Logic

*the purpose of
this appendix*

The purpose of this appendix is to give the reader a formal introduction to mathematical logic, emphasizing the skills and language that are needed to understand the arguments and proofs in this dissertation.

*mathematical
expressions;
mathematical
sentences*

A mathematical *expression* is analogous to an English noun; it is a name given to some mathematical object of interest. A mathematical expression is often a number, a function, or a set.

A mathematical *sentence* expresses a complete thought. The next example explores the difference between expressions and sentences.

EXAMPLE
*expressions
versus
sentences*

' (t_i, y_i) ' is an expression. The chosen variables (t_i and y_i) suggest that it represents an ordered pair of real numbers; a data point. ' $\{(t_i, y_i)\}_{i=1}^N$ ' is an expression. It is a set; a collection of data points.

' $(t_i, y_i) = (i, i^2)$ ' is a sentence. Note that sentences have verbs; the verb in this sentence is the equal sign, '='.

implications

'IF data is generated by the rule $y_n = n$, THEN $y_5 = 5$ ' is a sentence. It is a sentence of the form 'If P , then Q '; sentences of this form are called *implications*, and will be treated in detail in this appendix.

Of particular importance in mathematics are sentences that are either *true*, or *false*, but not both, and these are called *statements*:

DEFINITION
statement

A statement is a sentence that is either TRUE or FALSE.

*notation
for statements*

Capital letters, like P and Q , are frequently used to denote statements. Thus, P could denote the true statement ' $1+1=2$ ', and Q could denote the false statement ' $1+1=3$ '.

variable;
universal set

A *variable* is a symbol (often a letter) that is used to represent an arbitrary member of a specified set. This 'specified set' is called the *universal set* associated with the variable. Thus, the *universal set* gives the elements that one is allowed to draw on for a particular variable.

Many sentences become statements, once choices are made for the variables that appear in the sentence. For example, the truth of a sentence like ' $x = 3$ ' depends on the choice made for the number x . If x is 3, the sentence is true; otherwise, it is false. Such a sentence is called a *conditional sentence*:

DEFINITION
conditional sentence

A sentence with at least one variable, that becomes a statement whenever the variables are replaced by elements from their universal sets, is called a *conditional sentence*.

notation for conditional sentences;
 $P(x)$

The conditional sentence ' $x = 3$ ' can be conveniently denoted by $P(x)$. Thus, $P(3)$ is true, but $P(y)$ is false for $y \neq 3$.

The conditional sentence ' $x + y = 2$ ' can be denoted by $P(x, y)$. Thus, the sentences $P(1, 1)$ and $P(0.5, 1.5)$ are true, but $P(0, 3)$ is false.

For simplicity, the following convention will be adopted for the remainder of this appendix: the notation $P(x)$ will be used to represent a conditional sentence, with the understanding that x may represent more than one variable. For example, the conditional sentence $P(t, y)$ can be denoted by $P(x)$, by defining $x = (t, y)$.

EXAMPLE
statements and non-statements

The sentence $\{1, 2\} = \{2, 1\}$ is a statement; it is true. Two *sets* are equal when they contain precisely the same elements.

The sentence $(1, 2) = (2, 1)$ is a statement; it is false. In order for two *lists* to be equal, corresponding entries must be equal. See Section 1.2 for properties of lists.

The sentence '*The price of GE stock at 12:00 PM on 8/28/58 was greater than its price one day earlier at the same time*' is a statement. Even though the truth (true or false) of this sentence may be unknown to the reader, the sentence is either true, or false.

'*This sentence is false*' is not a statement. If it were true, then it would have to be false. If it were false, then it would have to be true. Therefore, it is neither true or false.

connectives

Connectives are used to make statements into 'larger' statements. The five basic connectives in mathematics are:

not, and, or, \Rightarrow , \Leftrightarrow .

These connectives are defined via the truth table below, and are discussed in the following paragraphs:

P	Q	not P	P and Q	P or Q	$P \Rightarrow Q$	$P \Leftrightarrow Q$
T	T	F	T	T	T	T
T	F	F	F	T	F	F
F	T	T	F	T	T	F
F	F	T	F	F	T	T

**negations;
not P**

The statement 'not P ' is called the negation of P . If P is true, then 'not P ' is false; if P is false, then 'not P ' is true.

**'and' statements;
 P and Q**

The statement ' P and Q ' is true only when both P and Q are true. For example, the conditional sentence ' $x = 1$ and $x = -1$ ' is false for all real numbers x , since a number cannot simultaneously equal 1 and -1 .

**'or' statements;
 P or Q**

The statement ' P or Q ' is true when *at least one* of P or Q is true. For example, the conditional sentence ' $x = 1$ or $x = -1$ ' is true for $x \in \{1, -1\}$, and false otherwise.

The reader is cautioned to distinguish between the English word 'or', and the mathematical word 'or'. In English, the phrase 'Carol or Bob went to the meeting' usually means that either Carol went, or Bob went, but not both went. In mathematics, the statement ' P or Q ' is true if P is true, or Q is true, or both P and Q are true.

**implications;
 $P \Rightarrow Q$**

The statement ' $P \Rightarrow Q$ ' is read as ' P implies Q ', and is perhaps the most commonly-occurring type of mathematical sentence. Therefore, it should not be surprising that the sentence has many synonyms, including:

synonyms for

$P \Rightarrow Q$

If P , then Q

Whenever P , then Q

Q , if P

Q , whenever P

P is sufficient for Q

Q is necessary for P

*hypothesis
and conclusion
of an implication*

Given an implication $P \Rightarrow Q$, the statement P is called the *hypothesis* of the implication, and Q is called the *conclusion* of the implication.

Roughly, a sentence of the form ' $P \Rightarrow Q$ ' is true if it has the property that whenever P is true, then Q is also true. If P is true, but Q is false, then the sentence ' $P \Rightarrow Q$ ' is false.

*vacuously
true*

Observe from the truth table that if the hypothesis of ' $P \Rightarrow Q$ ' is false, then the statement ' $P \Rightarrow Q$ ' is true; in this case, ' $P \Rightarrow Q$ ' is said to be *vacuously true*. (The reason for this aspect of the definition should become clear, as soon as the discussion of statements of the form 'For all x , $P(x) \Rightarrow Q(x)$ ' is completed.)

*equivalence;
 $P \Leftrightarrow Q$*

The sentence ' $P \Leftrightarrow Q$ ' is read as ' P is equivalent to Q ' or ' P if and only if Q '. The symbol ' \Leftrightarrow ' is more frequently used when an equivalence is displayed (set off and centered); and the phrase 'if and only if' is more frequently used when an equivalence appears in text.

The statement ' $P \Leftrightarrow Q$ ' is true exactly when P and Q have the *same* truth values; either they are both true, or they are both false.

*equivalent
statements
can be used
interchangeably*

Whenever two mathematical statements are equivalent, then they always have the same truth values, and hence can be used interchangeably. The next example presents an important mathematical equivalence.

EXAMPLE
DeMorgan's Laws

The following truth tables prove that:

For all statements P and Q ,

$$\begin{aligned}\text{not}(P \text{ and } Q) &\Leftrightarrow (\text{not } P) \text{ or } (\text{not } Q), \text{ and} \\ \text{not}(P \text{ or } Q) &\Leftrightarrow (\text{not } P) \text{ and } (\text{not } Q).\end{aligned}$$

These laws give the correct way to negate 'and' and 'or' statements, and are known as *DeMorgan's Laws*.

P	Q	P and Q	$\text{not}(P \text{ and } Q)$	$\text{not } P$	$\text{not } Q$	$(\text{not } P) \text{ or } (\text{not } Q)$
T	T	T	F	F	F	F
T	F	F	T	F	T	T
F	T	F	T	T	F	T
F	F	F	T	T	T	T

P	Q	P or Q	$\text{not}(P \text{ or } Q)$	$\text{not } P$	$\text{not } Q$	$(\text{not } P) \text{ and } (\text{not } Q)$
T	T	T	F	F	F	F
T	F	T	F	F	T	F
F	T	T	F	T	F	F
F	F	F	T	T	T	T

Other equivalent statements, that are important for understanding the proofs in this dissertation, are presented later on in this appendix.

quantifiers

Many commonly-occurring mathematical sentences take the forms:

For all x , $P(x) \Rightarrow Q(x)$; or

There exists x such that $P(x)$; or

There exists a unique x such that $P(x)$.

The phrases '*For all*', '*There exists*', and '*There exists a unique*' are called mathematical *quantifiers*. Quantifiers, and quantified statements, are the next topic of discussion.

DEFINITION	Let x have universal set U , and let $P(x)$ be a conditional sentence. The <i>truth set</i> of $P(x)$ is the set of all $x \in U$ for which $P(x)$ is true.
<i>truth set of a conditional sentence</i>	

EXAMPLE

The sentence ' $x = 3$ ' has an implied universal set $U = \mathbf{R}$. The truth set is $\{3\}$.

The sentence ' $\frac{1}{|x|} = 2$ ' has implied universal set $U = \{x \in \mathbf{R} \mid x \neq 0\}$. The truth set is $\{\frac{1}{2}, -\frac{1}{2}\}$.

The sentence ' $x + y = 1$ ' has implied universal set $\mathbf{R} \times \mathbf{R} := \{(x, y) \mid x \in \mathbf{R} \text{ and } y \in \mathbf{R}\}$. The truth set is $\{(x, 1-x) \mid x \in \mathbf{R}\}$. The *graph* of the equation $x + y = 1$ is a picture of its truth set; in this case, it is the line with y -intercept $(0, 1)$ and slope -1 .

DEFINITION
quantified
statements

Let x have universal set U , and let $P(x)$ be a conditional sentence.

The quantified statement '*For all x , $P(x)$* ' is true if and only if the truth set of $P(x)$ equals U .

The quantified statement '*There exists x such that $P(x)$* ' is true if and only if the truth set of $P(x)$ is nonempty.

The quantified statement '*There exists a unique x such that $P(x)$* ' is true if and only if the truth set of $P(x)$ contains exactly one element.

the use of
'if and only if'
in this definition

Notice the use of the words 'if and only if' in this definition. For example, in the second sentence, the words 'if and only if' are being used to compare the sentences

For all x , $P(x)$

and

the truth set of $P(x)$ equals U .

Therefore, these two sentences *always have the same truth values*. If one is true, so is the other; and if one is false, so is the other. Observe that if the sentence '*the truth set of $P(x)$ equals U* ' is false, then there must exist $y \in U$ for which $P(y)$ is false.

DEFINITION
counterexample

Let $P(x)$ be a conditional sentence, and let U be the universal set for x . A *counterexample* for $P(x)$ is a particular choice of y from the universal set for which $P(y)$ is false.

Thus, a *counterexample* is used to show that a mathematical sentence is not always true.

DEFINITION
proving
a sentence

Let $P(x)$ be a conditional sentence, and let U be the universal set for x . To *prove* $P(x)$ means to show that $P(x)$ is true, for all $x \in U$.

EXAMPLE

The quantified statement '*For all $x \in \mathbf{R}$, $\sqrt{x^2} = x$* ' is false. For a counterexample, choose $y = -2$. Then, the sentence ' *$\sqrt{(-2)^2} = -2$* ' is false.

The quantified statement, '*For all $x \geq 0$, $\sqrt{x^2} = x$* ' is true.

The quantified statement, '*There exists a unique $x \in \mathbf{R}$ such that $x(x^2 + 1) = 0$* ' is true. With universal set \mathbf{R} , the truth set of $x(x^2 + 1) = 0$ is $\{0\}$, which contains exactly one element.

The quantified statement, '*There exists a unique $x \in \mathbb{C}$ such that $x(x^2 + 1) = 0$* ' is false. Here, \mathbb{C} denotes the set of complex numbers. With universal set \mathbb{C} , the truth set of $x(x^2 + 1) = 0$ is $\{0, i, -i\}$, which has more than one element.

Let P be the quantified statement, '*There exists a real number x with $x^7 - \sqrt{2}x^6 + 3x^4 - x + \pi = 0$* '. Then, P is true. It is not necessary to know what particular real number makes the conditional sentence ' $x^7 - \sqrt{2}x^6 + 3x^4 - x + \pi = 0$ ' true; one need only establish that such a number does exist, and this is an easy consequence of the Intermediate Value Theorem.

*proving
implications*

To prove a quantified statement of the form

'*For all x , $P(x) \Rightarrow Q(x)$* ' ,

it is necessary to show that the sentence ' $P(x) \Rightarrow Q(x)$ ' is true for every choice of x from the universal set. If $P(x)$ is false, then the sentence ' $P(x) \Rightarrow Q(x)$ ' is vacuously true. Thus, one need only show that whenever $P(x)$ is true, so is $Q(x)$. It is precisely for this reason that the sentence ' $P \Rightarrow Q$ ' is defined to be true, when P is false.

*common forms
of proof*

The remainder of this appendix discusses the forms of proof that occur most frequently in this dissertation.

*direct proof
of $P \Rightarrow Q$*

A direct proof of

'*For all x , $P(x) \Rightarrow Q(x)$* '

shows that whenever $P(x)$ is true, so is $Q(x)$.

For example, Lemmas 1 and 3 in Section 1.3 use direct proofs.

*the
contrapositive
of an implication*

The statement ' $\text{not } Q \Rightarrow \text{not } P$ ' is called the *contrapositive* of the implication ' $P \Rightarrow Q$ '. The truth table below proves that an implication is equivalent to its contrapositive.

P	Q	$P \Rightarrow Q$	$\text{not } Q$	$\text{not } P$	$\text{not } Q \Rightarrow \text{not } P$
T	T	T	F	F	T
T	F	F	T	F	F
F	T	T	F	T	T
F	F	T	T	T	T

*proof of
 $P \Rightarrow Q$, by
contrapositive*

To prove a sentence of the form ' $P \Rightarrow Q$ ', one can equivalently prove ' $\text{not } Q \Rightarrow \text{not } P$ ' by a direct proof. Thus, the proof shows that whenever Q is not true, then P is not true.

symbols
 \neg , \wedge , \vee

For ease of notation, the symbols \neg for 'not', \wedge for 'and', and \vee for 'or' are introduced.

proof of
 $P \Rightarrow Q$, by
 contradiction

A *contradiction* is a sentence that is always false. For any statement S , the sentence ' $S \wedge (\neg S)$ ' is a contradiction.

Let P , Q and S be statements. The logical equivalence

$$(P \Rightarrow Q) \iff ((P \wedge \neg Q) \Rightarrow (S \wedge \neg S))$$

justifies the method of proof called *proof by contradiction*. To prove ' $P \Rightarrow Q$ ' by contradiction, one supposes that P is true and Q is false; and then reaches a contradiction.

For example, Lemma 4 in Section 1.3 uses a proof by contradiction.

proof of
 $P \Rightarrow (Q \vee R)$

To prove a sentence of the form ' $P \Rightarrow (Q \vee R)$ ', one often proves the logically equivalent sentence ' $(P \wedge \neg Q) \Rightarrow R$ ' (see the truth table below).

For example, Theorem 2 in Section 1.4 uses this proof form.

P	Q	R	$P \Rightarrow (Q \vee R)$	$P \wedge \neg Q$	$(P \wedge \neg Q) \Rightarrow R$
T	T	T	T	F	T
T	T	F	T	F	T
T	F	T	T	T	T
T	F	F	F	T	F
F	T	T	T	F	T
F	T	F	T	F	T
F	F	T	T	F	T
F	F	F	T	F	T

proof of
 $(P \wedge Q) \Rightarrow R$

To prove a sentence of the form ' $(P \wedge Q) \Rightarrow R$ ', one often proves the logically equivalent sentence ' $(P \wedge \neg R) \Rightarrow \neg Q$ '. The proof of this logical equivalence is left as an exercise.

proof of
 $P \iff Q$

To prove a sentence of the form ' $P \iff Q$ ', one usually proves the equivalent sentence ' $(P \Rightarrow Q) \wedge (Q \Rightarrow P)$ '. This logical equivalence justifies the use of the symbol ' \iff '.

This form of proof is used for the Proposition on page 38.

*proof by
induction*

The method of *proof by induction* is indispensable whenever it is desired to show that a statement $P(n)$ is true for all positive integers. The basic technique is:

- Show that $P(1)$ is true.
- Show that whenever $P(k)$ is true for a positive integer k , then $P(k + 1)$ is also true.

If both of these steps can be accomplished, then since $P(1)$ is true, $P(2)$ must be true. And since $P(2)$ is true, then $P(3)$ must be true—and so on. This logic is sometimes referred to as the *domino principle*.

There are many variations on this technique. For examples, see Lemmas 2 and 5 in Section 1.3.

Appendix 2

Vector Spaces, Norms and Inner Products

A mathematical object called a *vector space* provides the structure necessary to discuss *linear combinations* of elements, such as

$$\alpha_1 x_1 + \alpha_2 x_2 + \cdots + \alpha_n x_n .$$

A vector space consists of a set of objects, called *vectors*, on which two operations are defined: addition, and multiplication by scalars. The precise definition follows:

DEFINITION Let V be a set of objects, and let F denote either the real numbers (\mathbb{R}) or the complex numbers (\mathbb{C}). The elements of F are called *scalars*.
vector space over F ;
real vector space;
complex vector space

Let $+$ and \cdot denote operations,

$$+ : V \times V \rightarrow V, \quad (x, y) \mapsto x + y$$

$$\cdot : F \times V \rightarrow V, \quad (\alpha, x) \mapsto \alpha \cdot x .$$

The operation ' $+$ ' is called *addition*, and ' \cdot ' is called *scalar multiplication*. The element $\alpha \cdot x$ is denoted more simply by αx . Observe that $x+y$ and αx must be in V , for all $x, y \in V$ and $\alpha \in F$; that is, V is *closed under addition and scalar multiplication*.

If the axioms listed below are satisfied for all x, y and z in V , and for all scalars α and β , then V is called a *vector space over F* , and the elements of V are called *vectors*:

- addition is commutative: $x + y = y + x$
- addition is associative: $x + (y + z) = (x + y) + z$
- zero vector: There exists a vector $0 \in V$, called the *zero vector*, satisfying $x + 0 = x$ for all $x \in V$.
- additive inverses: For every $x \in V$ there exists a vector $-x \in V$ satisfying $x + (-x) = 0$. The vector $-x$ is called the *additive inverse of x* .
- distributive laws: $\alpha(x + y) = \alpha x + \alpha y$, and $(\alpha + \beta)x = \alpha x + \beta x$
- $1x = x$, and $\alpha(\beta x) = (\alpha\beta)x$

Vector spaces in which the scalars are real numbers are called *real vector spaces*, and those in which the scalars are complex numbers are *complex vector spaces*.

Uniqueness of the zero vector and additive inverses follows readily from the definition. The name $-x$ is justified for the additive inverse, since it can be easily shown that $(-1) \cdot x = -x$. The symbol 0 is used to denote both $0 \in F$ and $0 \in V$, with context determining the proper interpretation.

Primarily real vector spaces are considered in this appendix.

EXAMPLE

$V = \mathbf{R}^n$

as a real
vector space;

n -dimensional
Euclidean space

Let \mathbf{R}^n be the set of all ordered n -tuples of real numbers:

$$\mathbf{R}^n := \{(x_1, \dots, x_n) : x_i \in \mathbf{R}, 1 \leq i \leq n\}.$$

For $x = (x_1, \dots, x_n)$ and $y = (y_1, \dots, y_n)$ in \mathbf{R}^n , and $\alpha \in \mathbf{R}$, define addition and scalar multiplication by

$$\begin{aligned}(x_1, \dots, x_n) + (y_1, \dots, y_n) &:= (x_1 + y_1, \dots, x_n + y_n), \\ \alpha(x_1, \dots, x_n) &:= (\alpha x_1, \dots, \alpha x_n).\end{aligned}$$

The set \mathbf{R}^n , together with these operations, is a real vector space. This space, together with the usual measure of distance between elements, given by

$$d(x, y) = \sqrt{(x_1 - y_1)^2 + \dots + (x_n - y_n)^2},$$

is called *n -dimensional Euclidean space*.

EXAMPLE

$\mathbf{R}^{n \times m}$

The set of all $n \times m$ matrices with real number entries, together with the usual definitions of matrix addition and multiplication by real numbers, forms a real vector space which is denoted by $\mathbf{R}^{n \times m}$. The zero vector in $\mathbf{R}^{n \times m}$ is the $n \times m$ zero matrix. The additive inverse of $X \in \mathbf{R}^{n \times m}$ is found by multiplying each entry of X by the scalar -1 .

In addition to providing foundational material for review, the definitions and theorems that follow are needed to discuss the existence and uniqueness of solutions to the linear least-squares problem.

DEFINITION
*linear
combination*

Let V be a vector space over F , and let $y \in V$. Then, y is said to be a *linear combination of vectors* x_1, \dots, x_n in V provided there exist scalars $\alpha_1, \dots, \alpha_n$ in F for which

$$y = \alpha_1 x_1 + \dots + \alpha_n x_n.$$

DEFINITION
subspace of
a vector space

Let V be a vector space over F . A subset $W \subseteq V$ is called a *subspace of V* if W is itself a vector space over F , with the same operations of addition and scalar multiplication used in V . In particular, W must be closed under addition and scalar multiplication.

DEFINITION
subspace spanned
by S

Let V be a vector space, and let S be a nonempty subset of V . The *subspace spanned by S* is the intersection of all subspaces of V that contain S . If M is the subspace spanned by S , then the vectors in S are said to *span M* .

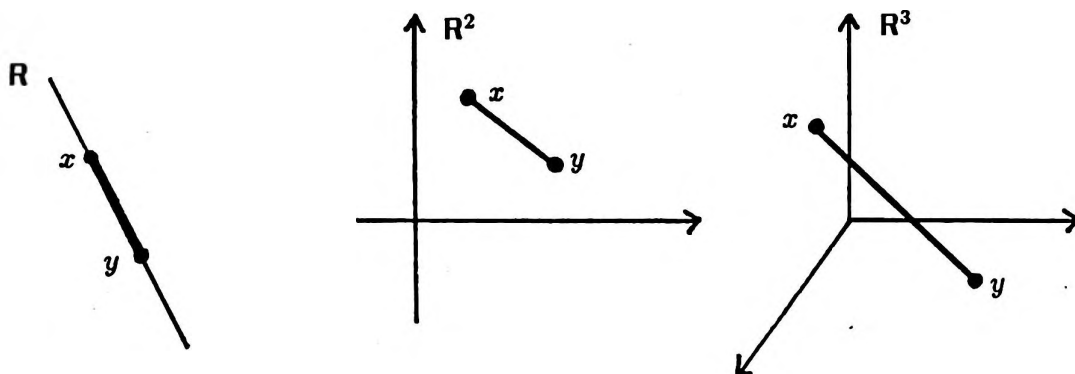
Thus, the subspace spanned by S is the 'smallest' subspace of V that contains S . It can be shown [H&K, p. 37] that the subspace spanned by S consists of all linear combinations of the elements from S .

DEFINITION
convex subset
of a vector space

A subset C of a vector space V is said to be *convex* if whenever $x, y \in C$, then also

$$M := \{\alpha x + (1 - \alpha)y \mid 0 \leq \alpha \leq 1\} \subseteq C.$$

In \mathbb{R} , \mathbb{R}^2 and \mathbb{R}^3 , the set M of the previous definition is the line segment connecting x and y .



It is routine to verify that subspaces are convex.

DEFINITION
*linearly
independent
set*

Let V be a vector space over F . A nonempty subset $S \subseteq V$ is said to be *linearly independent* if it has the following property: whenever a finite collection x_1, \dots, x_n of distinct vectors from S satisfies

$$\alpha_1 x_1 + \dots + \alpha_n x_n = 0$$

for scalars $\alpha_1, \dots, \alpha_n$, then it must be that

$$\alpha_1 = \alpha_2 = \dots = \alpha_n = 0 .$$

*linearly
dependent*

Consequently, if S is not linearly independent, then there exist distinct vectors x_1, \dots, x_n in S , and scalars $\alpha_1, \dots, \alpha_n$, not all zero, for which

$$\alpha_1 x_1 + \dots + \alpha_n x_n = 0 .$$

In this case, S is said to be *linearly dependent*.

DEFINITION
basis for V

Let V be a vector space. A *basis for V* is a linearly independent set of vectors from V that spans V . If V has a finite basis, then it is *finite-dimensional*. The notation $\dim(V) = n$ is used to denote that V has dimension n .

It can be shown that any two bases for a finite-dimensional vector space have the same number of vectors [Anton, p. 162].

DEFINITION
*linear
transformation;
kernel of T ,
 $\ker(T)$;
range of T*

Let V and W be vector spaces over F . In what follows, the notation '+' is used to denote both the addition in V , and the addition in W .

A *linear transformation from V into W* is a function $T: V \rightarrow W$ satisfying, for all $x, y \in V$ and for all $\alpha \in F$,

$$T(x + y) = T(x) + T(y) , \text{ and}$$

$$T(\alpha x) = \alpha T(x) .$$

The set $\{x \in V \mid T(x) = 0\}$ is called the *kernel of T* and denoted by $\ker(T)$.

The set $\{w \in W \mid \text{there exists } v \in V \text{ with } T(v) = w\}$ is called the *range of T* , and is denoted by $\mathcal{R}(T)$.

THEOREM

Let V and W be vector spaces over F , and let $T: V \rightarrow W$ be a linear transformation.

The set $\ker(T)$ is a subspace of V , and the set $\mathcal{R}(T)$ is a subspace of W [Anton, p. 231].

If V has dimension n , then ([Anton, p. 233])

$$\dim(\mathcal{R}(T)) + \dim(\ker(T)) = n .$$

EXAMPLE

*a matrix as
a linear
transformation;
rank of M*

Let M be an $n \times m$ matrix with real entries. The matrix M naturally defines a linear transformation from \mathbb{R}^m to \mathbb{R}^n , as follows: let $x \in \mathbb{R}^m$ be represented by an $m \times 1$ vector. Then, Mx is an $n \times 1$ vector, so $Mx \in \mathbb{R}^n$. It is routine to verify that this map

$$x \xrightarrow{M} Mx$$

satisfies the requirements of a linear transformation. Let M denote both the matrix M , and the linear transformation defined by M , with context determining the correct interpretation.

By the previous theorem, the range of M is a subspace of \mathbb{R}^n ; the dimension of the range is called the *rank of M* . Each column of M is a vector in \mathbb{R}^n . It can be shown that the range of M is the subspace of \mathbb{R}^n spanned by the column vectors of M .

The kernel of M is the set

$$\ker(M) = \{x \in \mathbb{R}^m \mid Mx = 0\} ,$$

where, here, 0 denotes the $n \times 1$ zero vector.

If the number of rows in M is the same as the number of columns, then M is called a *square* matrix. By definition, M is invertible if and only if there exists a matrix M^{-1} satisfying $MM^{-1} = M^{-1}M = I$. There are many equivalent characterizations of invertibility:

THEOREM

*equivalent
characterizations
of invertibility*

[Anton, p. 171] If M is an $n \times n$ matrix, then the following statements are equivalent:

- M is invertible.
- $\ker(M) = \{0\}$
- The determinant of M is nonzero.
- M has rank n .
- The row vectors of M are linearly independent.
- The column vectors of M are linearly independent.

A vector space V serves as the starting point for other useful spaces. If V is endowed with additional structure that allows one to measure the 'size' of elements of V , then V is called a *normed space*. The precise definition follows:

DEFINITION
normed space

Let V be a vector space over F , and let $\alpha \in F$.

If $F = \mathbb{R}$, then $|\alpha|$ denotes the absolute value of α . If $F = \mathbb{C}$, then for $\alpha = a + bi$, $|\alpha| = \sqrt{a^2 + b^2}$.

A *norm on V* is a function,

$$\| \cdot \| : V \rightarrow \mathbb{R}, \quad x \mapsto \|x\|,$$

that assigns to each $x \in V$ a *nonnegative* real number $\|x\|$, and that satisfies the following additional properties, for all $x \in V$ and $\alpha \in F$:

- $\|x\| = 0 \iff x = 0$
- $\|\alpha x\| = |\alpha| \cdot \|x\|$
- triangle inequality: $\|x + y\| \leq \|x\| + \|y\|$

A vector space V , together with a norm on V , is called a *normed space*.

*in a normed space,
distances between
vectors
can be measured*

In a normed space, the *distance between vectors x and y* can be defined via

$$d(x, y) := \|x - y\|.$$

It is routine to verify that the distance function defined in this way satisfies the properties given in the next definition:

DEFINITION
metric space

Let S be a set. A function

$$d: S \times S \rightarrow [0, \infty), \quad (x, y) \mapsto d(x, y),$$

is called a *metric on S* , if it satisfies the following properties:

- $d(x, y) = 0 \iff x = y$
- symmetry: $d(x, y) = d(y, x)$
- triangle inequality: $d(x, y) \leq d(x, z) + d(z, y)$

A set S , together with a metric on S , is called a *metric space*.

Consequently, every normed space is a metric space. Observe that a metric space need not be a vector space (★ just as a *topological space* need not be a vector space). However, many interesting metric spaces are also vector spaces.

DEFINITION	Let S be a metric space, with metric d .
<i>convergent sequence;</i>	Let $(x_n)_{n=1}^{\infty}$ be a sequence in S , and let $x \in S$. The sequence $(x_n)_{n=1}^{\infty}$ is said to <i>converge to</i> x if for every $\epsilon > 0$, there exists a positive integer $N = N(\epsilon)$, such that for all $n > N$, $d(x_n, x) < \epsilon$.
<i>Cauchy sequence;</i>	A sequence $(x_n)_{n=1}^{\infty}$ from S is said to be <i>Cauchy</i> (pronounced 'KO-shee') if for every $\epsilon > 0$, there exists a positive integer $N = N(\epsilon)$, such that if $m, n > N$, then $d(x_m, x_n) < \epsilon$.
<i>complete space</i>	The space S is said to be <i>complete</i> if whenever $(x_n)_{n=1}^{\infty}$ is a Cauchy sequence in S , then there exists $x \in S$ such that the sequence (x_n) converges to x .

THEOREM	Every finite-dimensional subspace of a normed space is complete (in the metric induced by the norm) [Krey, p. 73].
----------------	--

If a vector space V is endowed with structure which enables one to talk about *angles between vectors*, and in particular *orthogonality (perpendicularity) of vectors*, then it is called an *inner product space*. Only real vector spaces are treated in the next definition:

DEFINITION	Let V be a real vector space.
<i>inner product space</i>	An <i>inner product</i> on V is a function, $\langle \cdot, \cdot \rangle : V \times V \rightarrow \mathbf{R}, \quad (x, y) \mapsto \langle x, y \rangle,$ that satisfies the following properties, for all $x, y, z \in V$ and for all $\alpha \in \mathbf{R}$: <ul style="list-style-type: none"> • $\langle x + y, z \rangle = \langle x, z \rangle + \langle y, z \rangle$ • $\langle \alpha x, y \rangle = \alpha \langle x, y \rangle$ • $\langle x, y \rangle = \langle y, x \rangle$ • $\langle x, x \rangle$ is a nonnegative real number • $\langle x, x \rangle = 0 \iff x = 0$ A vector space V with an inner product on V is called an <i>inner product space</i> .

It follows easily from the definition that, for all $\alpha \in \mathbb{R}$ and $x, y \in V$:

- $\langle x, \alpha y \rangle = \alpha \langle x, y \rangle$
- $\langle x, y + z \rangle = \langle x, y \rangle + \langle x, z \rangle$

every
inner product space
is a normed space
and a metric space

It can be shown that every inner product space is a normed space, with norm

$$\|x\| = \sqrt{\langle x, x \rangle}.$$

Thus, every inner product space is also a metric space, with metric

$$d(x, y) = \|x - y\|.$$

Hilbert space

If V is an inner product space which is complete in the metric induced by the inner product, then V is called a *Hilbert space*.

EXAMPLE
 \mathbb{R}^n as an
inner product space

Let $x = (x_1, \dots, x_n)$ and $y = (y_1, \dots, y_n)$ be vectors in \mathbb{R}^n . Then,

$$\langle x, y \rangle := x_1 y_1 + \dots + x_n y_n$$

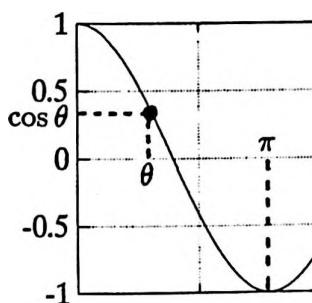
defines an inner product on \mathbb{R}^n , that induces the standard norm and metric on \mathbb{R}^n . It is a consequence of the previous theorem that \mathbb{R}^n with this inner product is a Hilbert space.

The Schwarz inequality, given next, describes the relationship between an inner product and its induced norm:

THEOREM
Schwarz
Inequality

Let V be an inner product space with inner product $\langle \cdot, \cdot \rangle$ and induced norm $\|\cdot\|$. Then, for all $x, y \in V$,

$$|\langle x, y \rangle| \leq \|x\| \|y\|.$$



By the Schwarz Inequality, if x and y are any nonzero vectors in an inner product space, then

$$-1 \leq \frac{\langle x, y \rangle}{\|x\| \|y\|} \leq 1.$$

Consequently, there is a unique angle $0 \leq \theta \leq \pi$ satisfying $\cos \theta = \frac{\langle x, y \rangle}{\|x\| \|y\|}$. If, in addition, $\langle x, y \rangle = 0$, then $\cos \theta = 0$ and hence $\theta = \frac{\pi}{2}$. In this case, x and y are said to be *orthogonal*.

These observations help to justify the following definition:

DEFINITION

*angle between
vectors;*

*orthogonal
vectors*

Let V be an inner product space with inner product $\langle \cdot, \cdot \rangle$ and induced norm $\| \cdot \|$. If x and y are nonzero vectors in V , then the *angle θ between x and y* is the unique angle $0 \leq \theta \leq \pi$ satisfying

$$\cos \theta = \frac{\langle x, y \rangle}{\|x\| \|y\|}.$$

For all $x, y \in V$, x and y are *orthogonal* (or *perpendicular*) if and only if $\langle x, y \rangle = 0$. A set of vectors is *mutually orthogonal* if every distinct pair chosen from the set is orthogonal.

If x is orthogonal to every vector in a set W , then x is said to be *orthogonal to W* .

A set of vectors $\{x_1, \dots, x_n\}$ is *orthonormal* if the vectors are mutually orthogonal, and if each vector in the set has norm 1.

It is routine to verify that if S is any set of mutually orthogonal vectors that does not contain the zero vector, then S is linearly independent.

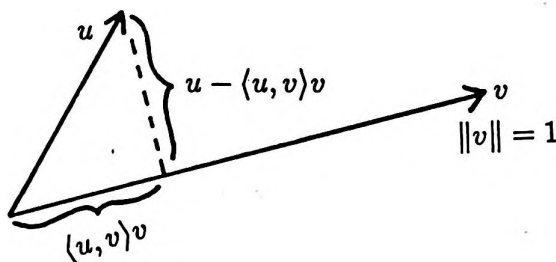
*orthogonal
projections*

Let V be an inner product space, and let $u \in V$. Suppose $v \in V$ has length 1. The vector

$$\langle u, v \rangle v$$

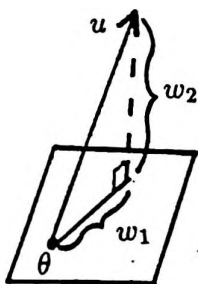
is called the *orthogonal projection of u on v* . The name is justified, since $\langle u, v \rangle v$ is a scalar multiple of v , and $u - \langle u, v \rangle v$ is orthogonal to v :

$$\begin{aligned} \langle u - \langle u, v \rangle v, v \rangle &= \langle u, v \rangle - \langle u, v \rangle \langle v, v \rangle \\ &= \langle u, v \rangle - \langle u, v \rangle (1) = 0. \end{aligned}$$



More generally, if W is a subspace with orthonormal basis v_1, \dots, v_n , and if u is any vector in V , then

$$w_1 := \langle u, v_1 \rangle v_1 + \dots + \langle u, v_n \rangle v_n$$



is the *orthogonal projection of u on W* , and $w_2 := u - w_1$ is the *component of u orthogonal to W* . Clearly, $w_1 \in W$. It is routine to verify that w_2 is orthogonal to W .

The projections thus defined are used in the Gram-Schmidt orthogonalization procedure, discussed next. This procedure is used in Section 2.3, Discrete Orthogonal Functions.

*Gram-Schmidt
orthogonalization
procedure*

Let $S = \{u_1, \dots, u_n\}$ be any set of linearly independent vectors in an inner product space. The *Gram-Schmidt orthogonalization procedure* converts S to a set $\{v_1, \dots, v_n\}$ of orthonormal vectors with the same span as S , as follows:

- Define $v_1 := \frac{u_1}{\|u_1\|}$.
- To get v_2 , project u_2 onto the subspace spanned by v_1 , and then normalize it:

$$v_2 := \frac{u_2 - \langle u_2, v_1 \rangle v_1}{\|u_2 - \langle u_2, v_1 \rangle v_1\|}.$$

- To get v_3 , project u_3 onto the subspace spanned by v_1 and v_2 :

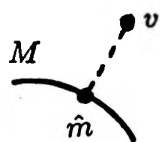
$$v_3 := \frac{u_3 - \langle u_3, v_1 \rangle v_1 - \langle u_3, v_2 \rangle v_2}{\|u_3 - \langle u_3, v_1 \rangle v_1 - \langle u_3, v_2 \rangle v_2\|}.$$

- Continue in this fashion, obtaining v_{i+1} by projecting u_{i+1} onto the subspace spanned by $\{v_1, \dots, v_i\}$.

infimum

In the next theorem, the notation 'inf' denotes the *infimum* of a set of real numbers, that is, the *greatest lower bound*.

MINIMIZING VECTOR THEOREM



[Krey, p. 144] Let V be an inner product space and $M \neq \emptyset$ a convex subset which is complete (in the metric induced by the inner product). Then for every given $v \in V$ there exists a unique $\hat{m} \in M$ such that

$$\inf_{m \in M} \|v - m\| = \|v - \hat{m}\|.$$

This theorem is applied next to the least-squares minimization problem:

THEOREM
existence and
uniqueness of
solutions to the
least-squares
problem

Let X be an $n \times m$ matrix with real entries, and let $y \in \mathbb{R}^n$. Then, there exists a solution to the least-squares problem

$$\inf_{b \in \mathbb{R}^m} \|y - Xb\|. \quad (1)$$

The following conditions are equivalent:

- a) The solution is unique.
- b) $\ker(X) = \{0\}$, where 0 is the $m \times 1$ zero vector.
- c) The columns of X are linearly independent in \mathbb{R}^n .

PROOF
existence

Let $X \in \mathbb{R}^{n \times m}$ and let $y \in \mathbb{R}^n$. The range of X is a subspace of $V = \mathbb{R}^n$; call this subspace M . Since

$$\{\|y - m\| \mid m \in M\} = \{\|y - Xb\| \mid b \in \mathbb{R}^m\},$$

it follows that if a solution to

$$\inf_{m \in M} \|y - m\| \quad (2)$$

exists, then so does a solution to (1). For if \hat{m} solves (2), then choose $\hat{b} \in \mathbb{R}^m$ with $\hat{m} = X\hat{b}$. Then, \hat{b} is a solution to (1).

Since subspaces are convex, and since finite-dimensional subspaces are complete, the Minimizing Vector Theorem guarantees a solution to (2), and hence a solution to (1).

uniqueness

It is next proven that $a) \Rightarrow b) \Rightarrow c) \Rightarrow a)$, completing the proof. Suppose that the solution to (1) is unique; call it b . Suppose for contradiction that there exists $x \neq 0$ in $\ker(X)$. Then, $X(b+x) = Xb + Xx = Xb$, so that $\|y - Xb\| = \|y - X(b+x)\|$. Therefore, $b+x$ is a different solution to (1), providing the desired contradiction. Suppose $\ker(X) = \{0\}$. Since $m = \dim(\ker(X)) + \dim(\text{range of } X)$, it follows that the range of X has dimension m . Therefore, the m columns of X must be linearly independent.

Suppose the columns of X are linearly independent. Again, since $m = \dim(\ker(X)) + \dim(\text{range of } X)$, it follows that $\ker(X) = \{0\}$. Suppose b_1 and b_2 are both solutions of (1). Then, $\|y - Xb_1\| = \|y - Xb_2\|$. But, since solutions to (2) are unique, and since both Xb_1 and Xb_2 are in M , it must be that $Xb_1 = Xb_2$, from which $X(b_1 - b_2) = 0$. Thus, $b_1 - b_2 \in \ker(X)$, from which $b_1 = b_2$. Thus, solutions are unique. ■

EXAMPLE

Let $f_1(t) = t$ and $f_2(t) = 2t$. Let $T = (0, 1, 2)$. Then, $f_1(T) = (0, 1, 2)$ and $f_2(T) = (0, 2, 4)$. The matrix X that arises in the linear least-squares approximation problem (see Section 2.2) is defined via $X_{ij} = f_j(t_i)$. Thus, $X = \begin{bmatrix} 0 & 0 \\ 1 & 2 \\ 2 & 4 \end{bmatrix}$. It is routine to verify that $\ker X = \{(t, -\frac{t}{2}) \mid t \in \mathbb{R}\} \neq \{0\}$. Let $y = (0, 1, 2) \in \mathbb{R}^3$. In this case, there are an infinite number of solutions to the least-squares problem $\min_{b \in \mathbb{R}^2} \|y - Xb\|^2$, as follows:

$$\begin{aligned} \|y - Xb\| = 0 &\iff y - Xb = 0 \\ &\iff \begin{bmatrix} 0 \\ 1 \\ 2 \end{bmatrix} - \begin{bmatrix} 0 \\ b_1 + 2b_2 \\ 2b_1 + 4b_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \\ &\iff b_1 + 2b_2 = 1 \\ &\iff b_2 = \frac{1}{2}(1 - b_1). \end{aligned}$$

In particular, all functions of the form Kf_1 and $\frac{1}{2}(1 - K)f_2$ solve the least-squares problem.

It was noted in Section 2.2 that if X^tX is invertible, then there is a unique solution to the linear least-squares approximation problem. However, in this example, $X^tX = \begin{bmatrix} 5 & 10 \\ 10 & 20 \end{bmatrix}$ is not invertible, since it has determinant 0. Note also that the columns of X are not linearly independent, since the second column is a multiple of the first.

★

Much of the material discussed in Section 2.3 is conveniently expressed in terms of the *singular values* of A . Important definitions and results are briefly summarized here. The remainder of this appendix can be skipped without any loss of continuity.

By definition, a real $m \times m$ matrix Q is *orthogonal* if $Q^tQ = I$, where I is the $m \times m$ identity matrix. Equivalently, Q is orthogonal if and only if $Q^t = Q^{-1}$. If Q is orthogonal, then it can be shown that the columns (rows) of Q form an orthonormal basis for \mathbb{R}^m [G&VL, p. 70].

If A is any real $m \times n$ matrix, then it can be shown that there exist orthogonal matrices $U \in \mathbb{R}^{m \times m}$ and $V \in \mathbb{R}^{n \times n}$ such that

$$U^tAV = \text{diag}(\sigma_1, \dots, \sigma_p) \in \mathbb{R}^{m \times n}, \quad p = \min\{m, n\},$$

where $\sigma_1 \geq \sigma_2 \geq \dots \geq \sigma_p \geq 0$, and where $\text{diag}(\sigma_1, \dots, \sigma_p)$ denotes the matrix with σ_i in row i and column i , and with zeroes elsewhere [G&VL, p. 71].

*singular values
of a matrix*

By definition, the σ_i are the *singular values* of A . The largest singular value, σ_1 , is also denoted by σ_{\max} .

Both the 2-norm (see p. 157) and Frobenius norm (see p. 159) of a matrix A are neatly characterized in terms of the singular values of A [G&VL, p. 72]:

$$\|A\|_F = \sqrt{\sigma_1^2 + \cdots + \sigma_p^2}, \quad p = \min\{m, n\},$$

$$\|A\|_2 = \sigma_{\max}.$$

There is a beautiful relationship between the singular values of A and the eigenvalues of $A^t A$, as follows:

$$\begin{aligned} U^t A V &= \text{diag}(\sigma_1, \dots, \sigma_p) \\ \Rightarrow (U^t A V)^t &= \text{diag}(\sigma_1, \dots, \sigma_p) \\ \Rightarrow V^t A^t U &= \text{diag}(\sigma_1, \dots, \sigma_p) \\ \Rightarrow (V^t A^t U)(U^t A V) &= \text{diag}(\sigma_1^2, \dots, \sigma_p^2) \\ \Rightarrow V^t (A^t A) V &= \text{diag}(\sigma_1^2, \dots, \sigma_p^2) \\ \Rightarrow A^t A &= V \text{diag}(\sigma_1^2, \dots, \sigma_p^2) V^t \end{aligned}$$

Thus, $A^t A$ is similar to $\text{diag}(\sigma_1^2, \dots, \sigma_p^2)$. Since similar matrices have the same eigenvalues, and since the eigenvalues of a diagonal matrix lie on the diagonal, it follows that $A^t A$ has eigenvalues $\sigma_1^2, \dots, \sigma_p^2$. Thus, the singular values of A (which are, by definition, nonnegative numbers) are the square roots of the eigenvalues of $A^t A$.

Appendix 3

Derivation of the Least-Squares Approximation Formula

Let f_1, \dots, f_N be functions from \mathbf{R}^m into \mathbf{R} . Let $\mathbf{b} = (b_1, \dots, b_m)$, and define $\mathbf{f}: \mathbf{R}^m \rightarrow \mathbf{R}^N$ by

$$\mathbf{f}(\mathbf{b}) = \begin{bmatrix} f_1(\mathbf{b}) \\ f_2(\mathbf{b}) \\ \vdots \\ f_N(\mathbf{b}) \end{bmatrix}.$$

The derivative of \mathbf{f} is represented by the $N \times m$ Jacobian matrix

$$\mathbf{Df} = \begin{bmatrix} \frac{\partial f_1}{\partial b_1} & \frac{\partial f_1}{\partial b_2} & \dots & \frac{\partial f_1}{\partial b_m} \\ \frac{\partial f_2}{\partial b_1} & \frac{\partial f_2}{\partial b_2} & \dots & \frac{\partial f_2}{\partial b_m} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_N}{\partial b_1} & \frac{\partial f_N}{\partial b_2} & \dots & \frac{\partial f_N}{\partial b_m} \end{bmatrix}.$$

As in the dissertation proper, the notation \mathbf{A}_{ij} is used to denote the entry in row i and column j of the matrix \mathbf{A} .

Note that $\mathbf{Df}_{ij} = \frac{\partial f_i}{\partial b_j}$. Define $S: \mathbf{R}^m \rightarrow \mathbf{R}$ by

$$S(\mathbf{b}) := \mathbf{f}(\mathbf{b})^t \cdot \mathbf{f}(\mathbf{b}) = \sum_{i=1}^N (f_i(b_1, \dots, b_m))^2.$$

Then,

$$\frac{\partial S}{\partial b_k} = 2 \sum_{i=1}^N f_i(\mathbf{b}) \frac{\partial f_i}{\partial b_k},$$

and so

$$\begin{aligned} \mathbf{DS}^t &= \begin{bmatrix} \frac{\partial S}{\partial b_1} \\ \frac{\partial S}{\partial b_2} \\ \vdots \\ \frac{\partial S}{\partial b_m} \end{bmatrix} = 2 \begin{bmatrix} \sum_{i=1}^N f_i(\mathbf{b}) \frac{\partial f_i}{\partial b_1} \\ \sum_{i=1}^N f_i(\mathbf{b}) \frac{\partial f_i}{\partial b_2} \\ \vdots \\ \sum_{i=1}^N f_i(\mathbf{b}) \frac{\partial f_i}{\partial b_m} \end{bmatrix} \\ &= 2 \begin{bmatrix} \frac{\partial f_1}{\partial b_1} & \frac{\partial f_2}{\partial b_1} & \dots & \frac{\partial f_N}{\partial b_1} \\ \frac{\partial f_1}{\partial b_2} & \frac{\partial f_2}{\partial b_2} & \dots & \frac{\partial f_N}{\partial b_2} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_1}{\partial b_m} & \frac{\partial f_2}{\partial b_m} & \dots & \frac{\partial f_N}{\partial b_m} \end{bmatrix} \begin{bmatrix} f_1(\mathbf{b}) \\ f_2(\mathbf{b}) \\ \vdots \\ f_N(\mathbf{b}) \end{bmatrix} \\ &= 2(\mathbf{Df})^t(\mathbf{f}(\mathbf{b})). \end{aligned}$$

The linear least-squares approximation formula in the text is now an application of this result, taking $f(b) = y - Xb$. The i^{th} component function is

$$f_i = y_i - \sum_{k=1}^m X_{ik} b_k ,$$

so that

$$\frac{\partial f_i}{\partial b_j} = -X_{ij} .$$

Then,

$$\begin{aligned} DS^t &= 2(Df)^t(f(b)) \\ &= 2 \left[\frac{\partial f_j}{\partial b_i} \right] (y - Xb) \\ &= 2[-X_{ji}](y - Xb) \\ &= 2(-X^t)(y - Xb) \\ &= -2(X^t y - X^t Xb) \\ &= 2(X^t Xb - X^t y) . \end{aligned}$$

REFERENCES

- [And] T.W. Anderson, **The Statistical Analysis of Time Series**, John Wiley & Sons, Inc., New York, 1971.
- [Anton] Howard Anton, **Elementary Linear Algebra**, fourth edition, John Wiley & Sons, Inc., New York, 1984.
- [Bar] Robert G. Bartle, **The Elements of Real Analysis**, second edition, John Wiley & Sons, Inc., New York, 1964.
- [Ber] Daniel Bernoulli, *Mém. de l'Académie de Berlin* **9** (1753), p. 173.
- [B&T] R.B. Blackman and J.W. Tukey, **The Measurement of Power Spectra, from the Point of View of Communications Engineering**, Dover, New York, 1959.
- [Blo] Peter Bloomfield, **Fourier Analysis of Time Series: An Introduction**, John Wiley & Sons, New York, 1976.
- [B&J] George E.P. Box, and Gwilym M. Jenkins, **Time Series Analysis; forecasting and control**, Holden-Day, San Francisco, 1970.
- [Boy] William E. Boyce and Richard C. DiPrima, **Elementary Differential Equations and Boundary Value Problems**, third edition, John Wiley & Sons, Inc., New York, 1977.
- [B-B] Buys-Ballot, *Les changements périodiques de température*, Utrecht (1847), p. 34.
- [Car] H.S. Carslaw, **Introduction to the Theory of Fourier's Series and Integrals**, Dover Publications, Inc., 3rd edition, 1930.
- [Cat] Donald E. Catlin, **Estimation, Control, and the Discrete Kalman Filter**, Springer-Verlag, New York, 1989.
- [C&P] James Caveny and Warren Page, *The Fundamental Periods of Sums of Periodic Functions*, College Math. Journal **20** (January 1989), no. 1.
- [C&T] J.W. Cooley and J.W. Tukey, *An algorithm for the machine computation of complex Fourier series*, Mathematics of Computation **19** (1965), no. 90, 297-301.
- [Dale] Dale, Monthly Not. R.A.S. **74** (1914), p. 628.
- [Dghty] Edward R. Dougherty, **Probability and Statistics for the Engineering, Computing, and Physical Sciences**, Prentice Hall, Englewood Cliffs, New Jersey, 1990.
- [Eli] Peter Eliason, **Tactical Trading and the Stock Tactics System**; information is available from W.P. Allen & Co., Inc., Consulting Engineers, PO Box 702, Portland, OR 97207.
- [Fra] John B. Fraleigh, **A First Course in Abstract Algebra**, third edition, Addison-Wesley Publishing Company, Inc., Reading, Mass., 1967.

- [Gold] David E. Goldberg, **Genetic Algorithms in Search, Optimization, and Machine Learning**, Addison-Wesley Publishing Company, Inc, Reading, Massachusetts, 1989.
- [G&VL] Gene H. Golub, and Charles F. Van Loan, **Matrix Computations**, second edition, The Johns Hopkins University Press, Baltimore and London, 1989.
- [Gon] Enrique A. González-Velasco, *Connections in Mathematical Analysis: the Case of Fourier Series*, AMS Monthly **99** (1992), no. 5, 427–441.
- [G&M] C. Granger and O. Morgenstern, **Predictability of Stock Market Prices**, Heath Lexington Books, D.C. Heath and Company, Lexington, Massachusetts, 1970.
- [G&R] U. Grenander and M. Rosenblatt, *Statistical spectral analysis of time series arising from stationary stochastic processes*, Ann. Math. Stat. **24** (1953), 537–558.
- [H&R] David Halliday and Robert Resnick, **Fundamentals of Physics**, John Wiley & Sons, Inc., New York, 1974.
- [HamD] R.W. Hamming, **Digital Filters**, third edition, Prentice Hall, Englewood Cliffs, New Jersey, 1989.
- [Ham] R.W. Hamming, **Numerical Methods for Scientists and Engineers**, 2nd ed., McGraw-Hill, New York, 1973.
- [H&K] Kenneth Hoffman and Ray Kunze, **Linear Algebra**, second edition, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1971.
- [Jef] R.L. Jeffery, **Trigonometric Series—a Survey**, University of Toronto Press, Toronto, 1956.
- [J&W] Gwilym M. Jenkins and Donald G. Watts, **Spectral Analysis and its applications**, Holden-Day, Inc., San Francisco, California, 1968.
- [Joh] David E. Johnson, **Introduction to Filter Theory**, Prentice-Hall, Inc., New Jersey, 1976.
- [Ken] M.G. Kendall, **Time-Series**, Charles Griffin & Company Ltd., London, 1973.
- [Krey] Erwin Kreyszig, **Introductory Functional Analysis with Applications**, John Wiley & Sons, New York, 1989.
- [Lag] Lagrange, *Recherches sur la manière de former des tables des planètes d'après les seules observations*, reprinted in *Oeuvres de Lagrange*, 1873, Vol. VI, 507–627.
- [Mab] Prescott C. Mabon, **Mission: Communications, The Story of Bell Laboratories**, Bell Telephone Laboratories, Inc., New Jersey, 1975.
- [New] James R. Newman, **The World of Mathematics**, Simon and Schuster, New York, 1956.
- [Olm] John M.H. Olmsted, **Real Variables, An Introduction to the Theory of Functions**, Appleton-Century-Crofts, Inc., New York, 1959.

- [O&T] John M. H. Olmsted and Carl G. Townsend, *On the Sum of Two Periodic Functions*, Delta Magazine **1** (Spring 1970), no. 4, 1–10.
- [O&S] Alan V. Oppenheim, Ronald W. Schaffer, **Digital Signal Processing**, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1975.
- [Oxf] **Oxford English Dictionary**, Clarendon Press, Oxford, Vols. II (cosine entry) and IX (sine entry). First published 1933; reprinted 1961, 1970.
- [Par] E. Parzen, *On consistent estimates of the spectrum of a stationary time series*, Ann. Math. Stat. **28** (1957a), 329–348.
- [Prenter] P.M. Prenter, **Splines and Variational Methods**, John Wiley & Sons, New York, 1975.
- [Rosen] Gerald H. Rosen, **A New Science of Stock Market Investing**, Harper & Row, New York, 1990.
- [Sch] A. Schuster, *On the investigation of hidden periodicities with application to a supposed 26 day period of meteorological phenomena*, Terr. Magnetism **3** (1897), 13–41.
- [S&H] Samuel D. Stearns and Don R. Hush, **Digital Signal Analysis**, 2nd edition, Prentice Hall, New Jersey, 1990.
- [S&D] Stewart and Dodgson, *Proc. R.S.* **29** (1879), p. 106.
- [S&B] J. Stoer and R. Bulirsch, **Introduction to Numerical Analysis**, Springer-Verlag, New York, 1980.
- [Sto] Stokes, *Proc. R.S.* **29** (1879), pp. 122, 303.
- [Str] Strachey, *Proc. R.S.* **26** (1877), p. 249.
- [Ter] Trevor J. Terrell, **Introduction to Digital Filters**, John Wiley & Sons, New York, 1980.
- [Weav] Weaver, H. Joseph, **Theory of Discrete and Continuous Fourier Analysis**, John Wiley & Sons, New York, 1989.
- [W&R] Sir Edmund Whittaker and G. Robinson, **The Calculus of Observations**, 4th edition, Blackie and Son, London, 1924.

INDEX

- A**
 addition formulas, 80
 aliasing, 91
 alignment (for adding lists), 14
 amplitude, 74
 analyst, 1
 'and' statement, 293
 angle between vectors, 308
 apparent period, 88
 approximate, 146
 approximation, 143
 augmented matrix, 60
- B**
 band-pass filter, 270
 basis, 303
 bouncing, 179
 buying long, 132
- C**
 Cauchy-Schwarz Inequality, 159
 Cauchy sequence, 306
`cfspline` (MATLAB FUNCTION), 213
 Chebyshev approximation, 145
 Chebyshev's Inequality, 116
 closed interval, 18
 column vector, 144
 codomain, 16
 commensurable numbers, 46
 complete space, 306
 complex exponential function, 85
 complex Fourier series, 267
 complex numbers, 18
 component functions, 2
`cond` (MATLAB COMMAND), 161
 conclusion of an implication, 294
 conditional sentence
 condition number, 155, 156, 161
 conjecture, 6
 connectives, 293
 continuity, 45
 contrapositive, 5, 298
 convergent sequence, 306
 convex, 302
 cosine function, 72
 origin of word, 94
 countably infinite, 10
 counterexample, 296
 crossover, 186
 cubic spline, 143, 204
 curvature, 207
 cycle, p -cycle, 34
 cyclic frequency, 79
- D**
 data point, 13
 data sets, 10
 data value, 13
 degree measure of an angle, 74
 degrees of freedom, 61
 deMorgan's laws, 294
 dense, 41
 designing filters, 267
 determinant, 155
 deterministic, 108
`dfs` (MATLAB FUNCTION), 223
 digital filters, 244
 digital signal, 19
 direct proof, 297
 discrete-domain data, 10
 discrete-domain function, 17
 discrete Fourier series, 215, 222
 discrete Fourier transform, 233, 234
 discrete orthogonal functions, 164, 165
 discrete signal, 19
 divides, 54
 domain of a function, 16
 double-angle formulas, 80
`dscorth` (MATLAB FUNCTION), 171
- E**
 echelon (row echelon form), 63
 echoing (MATLAB), 153
 Economics Algorithm, 130
 eigenvalue, 252
 eigenvector, 252
 element, 11

- ellipsis, '...', 13
- entry, 11, 13
- equality of lists, 14
- equivalence, 294
- Euclidean norm, 144
- event, 109
- expression, 291
- extending a function, 20
- F
- fast Fourier transform, 107, 233
- filter coefficients, 244
 - from a given transfer function, 268
- filters, 244
- findfil** (MATLAB FUNCTION), 274
- fit, 8
- fitness, 186
- Fourier series (continuous), 81
- frequency of sinusoids, 79
- Frobenius norm, 159
- functions, 16
- function notation, 16
- fundamental period, 41
- G
- generation, 187
- genetic** (MATLAB FUNCTION), 188
- genetic algorithm, 186
- Gibbs' phenomenon, 83
- gradient, 178
- grad measure of an angle, 74
- Gram-Schmidt orthogonalization procedure, 166, 309
- H
- half-angle formulas, 80
- half-open interval, 18
- Hamming, R.W., 9
- harmonics, 82
- Hilbert space, 307
- horizontal orientation for lists, 10
- hypothesis (plural 'hypotheses'), 2
- hypothesis of an implication, 294
- I
- imaginary part of a complex number, 87
- implications, 291 improving filters, 267
- increasing order, 10
- induction argument, 29
- infimum, 309
- initialization, 186
- inner product, 165
- inner product space, 306
- input, 10
- insight, 9
- integers, 18
- interpolation, 142
- interval notation, 18
- invertibility, 304
- irrational numbers, 47
- iterative techniques, 173
- J
- K
- Kalman filter, 107
- kernel, 303
- knots, 204
- known unknown, 4
- L
- Lanczos smoothing, 271
- least common multiple, 54
- least positive period, 41
- least-squares approximation, 145
- length of a list, 12
- linear combination, 301
- linear dependence on a parameter, 174
- linearizing technique, 173
- linearly dependent set, 303
- linearly independent set, 303
- linear system, 60
- linear transformation, 303
- list, 11
- listed, 10
- local random behavior, 119
- logic, 5, 291
- M
- maps to, 15
- Martingale Algorithm, 124
- MATLAB, 20
- matrix laboratory, 20

- mean (average), 7
- mean-square error, 85
- member, 11
- metric space, 305
- minimax approximation, 145
- minimizing vector theorem, 309
- mutation, 186
- mutually orthogonal functions, 165
- N
- natural cubic spline, 207
- negations, 293 neighbor, 108
- nonlin, 182
- noise, 230
- nonlin (MATLAB FUNCTION), 180
- nonlinear least squares approximation, 173
- nonrec (MATLAB FUNCTION), 249
- nonrecursive filter, 245
- norm, 143, 305
 - norm (MATLAB COMMAND), 157
 - 2-norm, 157
- notation, 10
- O
- objective function, 186
- open interval, 18
- operations on lists, 13
- 'or' statement, 293
- orientation (vertical or horizontal), 10
- orthogonal functions, 154
- orthogonality properties of sine and cosine, 84
- orthogonal matrix, 161
- orthogonal projections, 308
- orthogonal vectors, 308
- oscillation, 79
 - in polynomial interpolation, 203
- output, 10
- output list, 12
- P
- peak, 108
- period (of transfer function), 258
- periodic function, definition, 26
- periodicity, 26
- periodogram, 215, 222
- pervfft (MATLAB FUNCTION), 241
- phase shift, 74
- piecewise continuous periodic function, 81
- point spread, 131
- Pointwise Convergence Theorem, 82
- polyfit (MATLAB COMMAND), 211
- polyval (MATLAB COMMAND), 211
- prstop (MATLAB FUNCTION), 139
- proving a sentence, 296
- Q
- quantifiers, 295
- R
- radian frequency, 79
- radian measure of an angle, 74
- rand, MATLAB COMMAND, 119
- ramps, 231
- random behavior, 108
- random sample, 111
- random walk, 2
- range of a function, 16, 303
- rank, 304
- rational numbers, 12, 18
- rcond (MATLAB COMMAND), 161
- real numbers, 17
- real part of a complex number, 87
- real-valued function of a real variable, 17
- recursive filters, 245
- redundancy, 233, 234
- relative error, 156
- relatively prime, 34
 - uniqueness of decomposition, 38
- reproduction, 186
- reshaping techniques, 7, 33
- S
- Sampling Theorem, 93
- scaling periodic functions, 50
- scaling of a list, 14
- Schwarz Inequality, 307
- sensitivity of solutions, 155
- sentence, 291
- series number, 126
- shares, 131

sigma factors, 272
 signal, 19, 103
 sine function, 72
 origin of word, 94
 singular values, 159, 312
 sinusoid, 74
 size, 143
 smoothing filters, 246, 260
 span, 302
 splines, 203
spline (MATLAB COMMAND), 211
 statement, 291 stochastic process, 2
 stock market data, 124
 strings, 186
symtolic (MATLAB FUNCTION), 275
 subspace, 302
T
 time, 10
 time list, 11
 time series, 19, 103
tpstest (MATLAB FUNCTION), 122
transfct (MATLAB FUNCTION), 264
 transfer function, 252, 258
 transpose, 145
 trapezoid rule, 245
 triangle approach to trigonometry, 94
 trigonometric identities, 80
 trough, 108
 truncated Fourier series, 271
 truth set, 295
 turning point test, 108
U
 uniform convergence, 84
 Uniform Convergence Theorem, 84
 uniform time list, 14
 universal set, 292
V
 variable, 292 vector space, 143, 300
 \mathbf{R}^∞ , 252
 vertical orientation for lists, 10
W
 Weierstrass Approximation Theorem, 9