EXERCISE SOLUTIONS
for
COMPUTATION AND PROBLEM SOLVING
IN UNDERGRADUATE PHYSICS

IDL • MATLAB • OCTAVE
PYTHON • MAXIMA • MAPLE
MATHEMATICA • FORTRAN • C
NUMERICAL RECIPES • LSODE • MUDPACK
\LaTeX • T\!\TeX

DAVID M. COOK
with assistance from
DANICA DRALUS, LU ’02
RYAN PETERSON, LU ’03
SCOTT KAMINSKI, LU ’04
MICHELLE MILNE, LU ’04
LAUREN KOST, LU ’05
CLAIRE WEISS, LU ’07
ERIK GARBACIK, LU ’08

Department of Physics
Lawrence University
711 E Boldt Way SPC 24
Appleton, Wisconsin 54911

Copyright © 2000–23 by David M. Cook

This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License (creativecommons.org/licenses/by-nc-sa/4.0/). Any use not permitted by this license requires authorization in writing from David M. Cook.
Preface

This document contains solutions to selected exercises from Computation and Problem Solving in Undergraduate Physics (CPSUP), a book that has grown from small beginnings in the 1990s to a flexible volume that provides an orientation to a subset of tools chosen from

- the general purpose programs IDL, MATLAB, OCTAVE, PYTHON, MAXIMA, MAPLE, and MATHEMATICA,
- the programming languages FORTRAN and C,
- FORTRAN NUMERICAL RECIPES,
- C NUMERICAL RECIPES,
- the FORTRAN procedure library LSODE,
- the UNIX drawing program TGIF, and
- the tool \LaTeX{} for preparation of technical documents.

In addition, chapters on ordinary differential equations, integration, and root finding provide examples of the use of the selected subset of tools for solving a wide variety of problems in physics. Problems from mechanics, electromagnetic theory, quantum mechanics, thermodynamics, statistical mechanics, relativity, and other subarea of physics are included.

This document admits the same flexibility in composition that characterizes CPSUP. Both CPSUP and this document can be configured to include all of the possibilities or only a selected subset of the options. Because there are 13 different components, each of which can be included or not, there are technically \(2^{13} = 8192\) versions of these items. To be sure, the vast majority of these possibilities makes no sense. Still, the number of versions is staggering. Creating documents with this degree of flexibility would be impossible without exploiting the elegant features of the ifthen package in \LaTeX{}, and I owe an immense debt to Donald Knuth, Leslie Lamport, and numerous others who have contributed to the development of that publishing system.

David M. Cook
Appleton, Wisconsin
19 February 2023
# Table of Contents

Preface iii

Table of Contents v

## 3 Introduction to MATLAB

3.1 Creating Vectors and Matrices ............................................. 1
3.2 Properties of `sort` ......................................................... 3
3.5 Summing Elements in a Vector ............................................ 5
3.6 Evaluating a Cross Product ................................................ 7
3.8 Eigenvectors/Values in a Single Matrix .................................. 8
3.9 Finding/Plotting Eigenvalues/Vectors ..................................... 9
3.10 Stark Effect for $n = 2$ and $n = 3$ ..................................... 14
3.11 Damped and Undamped Sine Waves ...................................... 17
3.13 Plotting the Potential of a Charged Disk ................................ 18
3.14 A Charging Capacitor ....................................................... 19
3.15 Plotting the Output of a Fabry-Perot Interferometer .................. 21
3.16 Plotting Relativistic Functions .......................................... 23
3.18 Plotting Transmission/Reflection of Thin Film ......................... 25
3.19 Plotting the Potential of Two Charged Disks ............................ 27
3.22 Plotting Relativistic Functions: Compton Effect ...................... 29
3.23 Field of a Moving Charge .................................................... 31
3.24 Plotting the Four-Slit Interference Pattern ............................. 35
3.25 Plotting the Planck Blackbody Radiation Curve ....................... 38
3.26 Plotting the On-Axis Field of a Solenoid ............................... 42
3.27 Plotting the On-Axis Field of a Pair of Current Loops ................. 48
3.28 Current in an LRC Circuit .................................................. 52
3.29 Cooking a Spherical Potato ................................................ 54
3.30 Gravitational Potential Energy in a Plane .............................. 59
3.31 Plotting Probability Densities in Hydrogen ............................. 62
3.32 Standing Waves in a Cube (2001 Version) ............................... 68
3.32 Standing waves in a cube (2002 Version) ............................... 74
3.37 Animation of a Vibrating String ......................................... 82
3.38 Animation of a User Function ............................................. 86
3.39 Animation of a Square Membrane ....................................... 90
3.40 Kaprekar Constant ......................................................... 92

## 9 Introduction to Programming in Psuedocode

9.2 Relationship between IF-THEN-ELSE and CASE .......................... 95
9.7 Tracking Both Extremes and their Positions ............................. 98
9.9 Mystery Procedure 1 .......................................................... 100

## 11 Solving ODEs numerically with MATLAB

95

103
11.17 Van der Pol Oscillator (MATLAB) ......................................................... 103
11.18 Large Amplitude Pendulum (MATLAB) .................................................. 107
11.19 Planet in non-inverse square gravity (MATLAB) ....................................... 112
11.21 Satellite with two suns (MATLAB) .......................................................... 114
11.22 Charge in Crossed Fields (MATLAB) ...................................................... 118
11.24 The Lorenz attractor (MATLAB) .............................................................. 123
11.27 Dynamics of a Chemical Reaction (MATLAB) .......................................... 126
11.28 The Predator-Prey Problem (MATLAB) ................................................... 129
11.31 The Elastic Swing (MATLAB) ................................................................. 132

13 Evaluating Integrals 141

numerically with MATLAB

13.18 Quantum Harmonic Oscillator Turning Points (MATLAB) ....................... 141
13.19 Maxwell-Boltzmann Distribution (MATLAB) ........................................... 143
13.20 Black Body Radiation (MATLAB) ............................................................ 145
13.21 Confidence Intervals for Gaussian (MATLAB) ......................................... 148
13.22 Earth Falling into Sun (MATLAB) ............................................................ 150
13.23 Properties of Lorentz Distribution (MATLAB) ......................................... 152
13.24 Electron inside Bohr Radius (MATLAB) ................................................... 154
13.25 Elliptic Integrals (MATLAB) ................................................................. 156
13.26 Large Amplitude Simple Pendulum (MATLAB) ...................................... 160
13.27 N-th Order Bessel Functions (MATLAB) ................................................ 163
13.28 The Bessel Function $J_1(x)$ (MATLAB) .................................................. 165
13.29 Off-Axis Potential of Circular Ring (MATLAB) ......................................... 167
13.30 Off-Axis Field of Current Loop (MATLAB) .............................................. 170

14 Finding Roots 175

numerically with MATLAB

14.12 Square Root by Newton’s Method (MATLAB) .......................................... 175
14.14 Natural Frequencies of Bar (MATLAB) ................................................... 178
14.17 Finite Depth Square Well (MATLAB) ..................................................... 184
14.18 Single-Slit Diffraction (MATLAB) ............................................................ 191
14.19 Roots of $J_0(x)$ (MATLAB) ................................................................. 194
14.20 Double-Welled Potential (MATLAB) ........................................................ 196
14.24 A Fluid Mechanics Problem (MATLAB) ................................................ 198

15 Introduction to Partial Differential Equations 201

numerically with MATLAB

15.2 A Vertically Hanging String ................................................................. 201
15.3 Second-Order ODEs Made Self-Adjoint ................................................... 205
15.7 FDM 1D Homogeneous Helmholtz Equation .............................................. 207
15.8 FEM 1D Homogeneous Helmholtz Equation ............................................. 211
15.9 1-D Inhomogeneous Helmholtz Equation ................................................. 217
15.12 Assembly for Three-Element Linear System .......................................... 224
15.20 FDM 2D Inhomogeneous Helmholtz Equation .......................................... 227
15.25 Finite Difference Formulas via Taylor Series ......................................... 228
15.26 Shape Functions for Two-Node Linear Element ..................................... 230

numerically with MATLAB

15.1 FDM 1D Inhomogeneous Equation (MATLAB) ........................................ 231
15.4 FDM 1D $u'' + ku = 0$ (MATLAB) .......................................................... 240
15.13 FEM 1D Thermal Equilibrium (MATLAB) ............................................... 246
15.23 FDM 2D Laplace Equation: Alternate Convergence Test (MATLAB) ......... 252
15.24 FEM 2D Poisson Equation (MATLAB) .................................................... 259
Chapter 3

Introduction to MATLAB

3.1 Creating Vectors and Matrices

Exercise: Write and test MATLAB statements to create (a) a five-element column vector, (b) an $8 \times 8$ unit matrix, and (c) a $10 \times 10$ matrix all of whose elements are zero except those on the main diagonal (which are all 2) and those on the diagonals just above and just below the main diagonal (which are all $-1$). Search for a route more efficient than laboriously setting each of the 100 elements in the $10 \times 10$ matrix individually.

Solution: (a) Creating a five element column vector of the form

$$\begin{pmatrix} v \\ w \\ x \\ y \\ z \end{pmatrix}$$

is equivalent to creating a $1 \times 5$ matrix. Within vectors, MATLAB uses semicolons to denote the end of a row. A five element column vector is then created with the statement

```matlab
>> h = [ 1; 2; 3; 4; 5 ]
```

```
h =
 1
 2
 3
 4
 5
```

(b) To create an $8 \times 8$ unit matrix, we invoke the function `eye(n)`. Here, the single argument $n$ denotes the dimension of one side of a unit matrix. Using this function with $n = 8$, we create the $8 \times 8$ unit matrix with the statement

```matlab
>> j = eye(8)
```

```
j =
 1 0 0 0 0 0 0 0
 0 1 0 0 0 0 0 0
 0 0 1 0 0 0 0 0
 0 0 0 1 0 0 0 0
 0 0 0 0 1 0 0 0
 0 0 0 0 0 1 0 0
 0 0 0 0 0 0 1 0
 0 0 0 0 0 0 0 1
```
(c) To construct the requested $10 \times 10$ matrix we utilize MATLAB’s \texttt{for} loops. First, we create a matrix on which the \texttt{for} loops can work. Since most of the values are zero, we tell MATLAB to create a $10 \times 10$ matrix of zeros using the function \texttt{zeros}. Then, we create \texttt{for} loops to set the elements in the main diagonal, subdiagonal and superdiagonal. The statements used and their outputs are

\begin{verbatim}
>> k = zeros(10,10);
>> for i = 1:10 k(i,i) = 2; end
>> for i = 1:9 k(i+1,i) = -1; end
>> for i = 1:9 k(i,i+1) = -1; end
>> k
k =
  2 -1  0  0  0  0  0  0  0  0
 -1  2 -1  0  0  0  0  0  0  0
  0 -1  2 -1  0  0  0  0  0  0
  0  0 -1  2 -1  0  0  0  0  0
  0  0  0 -1  2 -1  0  0  0  0
  0  0  0  0 -1  2 -1  0  0  0
  0  0  0  0  0 -1  2 -1  0  0
  0  0  0  0  0  0 -1  2 -1  0
  0  0  0  0  0  0  0 -1  2 -1

Create a $10 \times 10$ matrix of zeros.
Set elements in the main diagonal 2.
Set elements in the superdiagonal -1.
Set elements in the subdiagonal -1.
Show the matrix.
\end{verbatim}
3.2 Properties of sort

Exercise: Look up the function sort both in the on-line help and in the printed MATLAB manuals. Then, create a vector of your choice and test the use of sort, following the pattern illustrated in the documentation. Finally, write in your own words a brief description of what sort does.

Solution: Documentation on the function sort in the MATLAB User Guide is limited. The MATLAB command help sort, however, yields more information. The main purpose of the sort function is to sort items in vectors from lowest to highest. We begin by utilizing the sort function on a vector of six integers with the statements

```matlab
>> a = [ 4 2 8 13 0 6 ];
>> asort = sort(a)
asort = 0 2 4 6 8 13
```

As expected the items in the vector were sorted lowest to highest. When the output variable is a two-component vector, as in

```matlab
>> [ asort, index ] = sort(a)
asort = 0 2 4 6 8 13
index = 5 2 1 6 3 4
```

the function sort returns not only the sorted vector but also a vector whose elements indicate the position occupied in the original vector by each element in the sorted vector.

Complex elements are sorted on the basis of their absolute values and then, secondarily, on the angle at which the complex element lies in the complex plane. Thus, for example, the vector

```matlab
>> b = [ -3-3*i 3-3*i -3+3*i 3+3*i 4+i 2+3*i 6 ];
>> [ bsorted, index ] = sort(b)
bsorted = 2+3i 4+i -3-3i 3-3i 3+3i -3+3i 6
index = 6 5 1 2 4 3 7
```

Although it does not really make sense to sort a vector of complex numbers, let’s look at how MATLAB did sort them. MATLAB looks first at the absolute values of the numbers, which—with the numbers as in the sorted order—are

```matlab
>> abs( bsorted )
an = 3.6056 4.1231 4.2426 4.2426 4.2426 4.2426 6.0000
```

which appear in increasing order. The values in positions 3, 4, 5, and 6, however, all have the same absolute value. In that case, the second-level sorting examines the angles at which the values appear in the complex plane. We find those angles with the statement

```matlab
>> angle( bsorted )
an = 0.9828 0.2450 -2.3562 -0.7854 0.7854 2.3562 0
>> ans*180.0/pi
ans = 56.3099 14.0362 -135.0000 -45.0000 45.0000 135.0000 0
```

(Here, the first line of output gives the values in radians, and the second line of output gives the values in degrees.) In this output, the four values in the center are presented in increasing order, confirming the information in the MATLAB online help message. Apparently, angles are regarded as ranging from $-\pi$ to $\pi$, so the ordering of complex numbers of equal absolute value is as represented in Fig. E3.1.

We look next at a vector of strings created and sorted with the commands
Exercise 3.2

Figure E3.1: For complex numbers with equal absolute values, the `sort` function orders elements in quadrant III first, those in quadrant IV second, those in quadrant I third, and those in quadrant IV fourth.

>> c = ['Ryan' 'Danica' 'Teresa' 'Eric' 'Mark' 'Jim' ];
>> csorted = sort( c )
csorted = DEJMRTaaaaacceeiiikmnrrrsy

Since MATLAB treats strings as arrays of characters, when sorting a vector of strings, the strings are concatenated into a single string whose elements—individual characters—are then sorted in the order of their ASCII codes. In particular, all capital letters preceding all lowercase letters.¹

¹Evidently, starting in MATLAB Version R2017a, the function `sort` will properly alphabetize a list of strings if the strings are placed in the list with double quotation marks.
3.5 Summing Elements in a Vector

**Exercise:** (a) Describe and test a sequence of MATLAB statements that uses a `for/end` loop to evaluate \( \sum_i a_i \) when the values of \( a_i \) are supplied as the elements of the vector \( a \). In essence you will have to initialize a variable to zero and then, in the loop, successively add to that variable each of the elements \( a_i \) in turn. (b) Describe and test a sequence of MATLAB statements that uses the built-in function `sum` to achieve the same end. (c) Explore and then describe the behavior of the built-in function `cumsum` when it is applied to a vector. (d) Explore and describe the behavior of `sum` and `cumsum` when applied to a matrix rather than a vector.

**Solution:** (a) To explore how to add values from a vector successively to one another we begin by first creating our vector. Any arbitrary vector will do, but for simplicity we will form a vector of integers using MATLAB’s colon operator. Then, we need to initialize a variable, `asum`, to be used in a `for` loop. Finally, using a `for` loop, we can add successive elements of a vector. To carry this out in MATLAB we might use the statements:

```
>> a = 0:20
a = Columns 1 through 11
0 1 2 3 4 5 6 7 8 9 10
Columns 12 through 21
11 12 13 14 15 16 17 18 19 20
>> asum=0;
>> for i=1:21 asum = asum + a(i); end
>> asum
asum = 210
```

Create a vector of 21 integers (values 0–20) and display the vector. Initialize value to be used in `for` loop. Add together all elements in the vector. Print result.

(b) MATLAB also has a function that will add together all elements in a vector with a single statement. The MATLAB function `sum` accomplishes this task with the single statement:

```
>> sum(a)
ans = 210
```

The resulting value, displayed automatically, matches our expectation.

(c) When applied to a vector, the function `cumsum` returns another vector, each of whose elements is the cumulative sum of the elements of the original vector up to and including the element in whose position the returned value is located. Thus, with the vector `a` already defined, we have that:

```
>> cumsum(a)
ans = Columns 1 through 11
0 1 3 6 10 15 21 28 36 45 55
Columns 12 through 21
66 78 91 105 120 136 153 171 190 210
```

where, for example, element 4 in this result is:

\[ a(1) + a(2) + a(3) + a(4) = 0 + 1 + 2 + 3 = 6 \]

(d) To apply these functions to a matrix, we must first create a matrix, which—for simplicity—we will take to be the \( 4 \times 4 \) matrix created with the statement:

```
>> b = [ 0 1 2 3; 4 5 6 7; 8 9 10 11; 12 13 14 15]
b =
0 1 2 3
4 5 6 7
8 9 10 11
12 13 14 15
```

Then, we note the results of the statements: 
>> sum(b)
anst = 24  28  32  36
>> cumsum(b)
anst =  0  1  2  3
        4  6  8 10
        12 15 18 21
        24 28 32 36

Evidently, `sum(b)` has returned a vector whose elements are the sums of the columns in \( b \) while \( \text{cumsum}(b) \) has returned a matrix in which the element in each column is the cumulative sum of the elements in that column of \( b \) up to and including the element in the position of the element in question in the result.

Note, however, that both `sum` and `cumsum` have a second (optional) argument. If that argument is given the value 1, the results are the same as when the second argument is omitted.

>> sum(b,1)
anst =  24  28  32  36
>> cumsum(b,1)
anst =  0  1  2  3
        4  6  8 10
        12 15 18 21
        24 28 32 36

so the value 1 is the default value for this second argument. If, however, we give the second argument the value 2, then we find the results

>> sum(b,2)
anst =  6
        22
        38
        54
>> cumsum(b,2)
anst =  0  1  3  6
        4  9 15 22
        8 17 27 38
       12 25 39 54

This time, the sums and cumulative sums are evaluated across columns rather than down rows and, in the first case, a column vector rather than a row vector is generated.
3.6 Evaluating a Cross Product

**Exercise:** Write and test an MATLAB function that accepts two three-component vectors as input and returns a three-component vector containing the cross product of the two input vectors.

**Solution:** First, we create the pro-file

```matlab
function f = crossprod(x, y)
    f = [ x(2)*y(3)-x(3)*y(2), x(3)*y(1)-x(1)*y(3), x(1)*y(2)-x(2)*y(1) ];
end
```

in a text editor, storing it with the name `crossprod.m` in the default directory. This function requires the user to initialize a pair of vectors before using it. Here, we set two vectors \( x = [2 \ 3 \ 4] \) and \( y = [5 \ 6 \ 7] \) to test the MATLAB function. We invoke `crossprod` with the statements \(^2\)

```matlab
>> x = [2 3 4]; y = [5 6 7];
>> crossprod(x,y)
   crossprod = -3   6   -3
>> crossprod(y,x)
   crossprod =  3  -6   3
```

Computing the cross product by hand yields

\[
(3 * 7 - 4 * 6, 4 * 5 - 2 * 7, 2 * 6 - 3 * 5) = (-3, 6, -3)
\]

which checks out with the output from the MATLAB function. Further, the last statement verifies that exchanging the factors changes the sign of the result.

---

\(^2\)MATLAB will find the file defining the function in the default directory. We need not compile it explicitly.
3.8 Eigenvectors/Values in a Single Matrix

Exercise: Using `eig` for the main calculation, write and test a function that accepts a symmetric matrix as input and returns a single matrix, each column of which contains an eigenvalue as its first element and the associated eigenvector as its remaining elements.

Solution: To create a function that will produce the requested matrix of eigenvalues and eigenvectors we need to create a new M file to define a function that, essentially, invokes `eig` to find the eigenvalues and eigenvectors of the matrix submitted as its one argument but then rearranges the output to assemble it all into the desired single matrix. The coding for an appropriate M-file is

```matlab
function y = lueigen(x)
% LUEIGEN - Returns eigenvalues/vectors in a single matrix.
% The function lueigen uses the MATLAB routine eig to find the
% eigenvalues and eigenvectors of a symmetric matrix input as
% x but then rearranges the output from eig into a single
% matrix whose first row contains the eigenvalues and whose
% columns contain the eigenvectors.

[y,evals] = eig(x); % Find eigenvectors and eigenvalues
n = size(evals); % Find number of eigenvalues
n = n(1);
for i=1:n evs(i)=evals(i,i); end
y = [ evs; y ];

Now, we can use this procedure file within MATLAB to find eigenvalues and eigenvectors. We begin by creating a matrix `a` with the statement

```matlab
>> a = [ 1,3,0; 3,7,2; 0,2,4 ]
```

Notice that this is the same matrix used in the eigenvalue section of the chapter so we have some source for comparison. We then use our function to find the eigenvalues and the corresponding eigenvectors and print the results with the statements

```matlab
>> valsvecs = lueigen(a)
valsvecs =
 -0.3893  3.4469  8.9425
  0.8912 -0.3107  0.3305
-0.4127 -0.2534  0.8749
  0.1881  0.9161  0.3540
```

The resulting matrix confirms the results found in the eigenvalues and eigenvectors section in the chapter, each column representing an eigenvalue (in row one) and the corresponding eigenvectors beneath each of the eigenvalues.
### 3.9 Finding/Plotting Eigenvalues/Vectors

**Exercise:** Find the eigenvalues and eigenvectors for each of the following matrices:

(a) 
\[
\begin{pmatrix}
0 & 1 & 0 \\
1 & 0 & 1 \\
0 & 1 & 0
\end{pmatrix}
\]

(b) A 10 × 10 matrix with zeroes everywhere except that all elements on the main diagonal have the value 2.0 and all elements on the sub and superdiagonals have the value −1.0. Search for a route more efficient than laboriously setting each of the 100 elements in the 10 × 10 matrix individually.

(c) Identify the five eigenvectors in part (b) belonging to the lowest five eigenvalues and, for each, plot a graph whose vertical coordinate is the component of the eigenvector and whose horizontal coordinate is the component number. Actually, in the underlying physical context, it would be more appropriate to plot graphs of the result of augmenting these eigenvectors by placing an element 0.0 both before the first element and after the last element in the eigenvector. If, for example, the eigenvectors are in the columns of `evecs`, then the statement

\[
\gg \text{plot(} [ 0.0; \text{evecs(:,1); 0.0 } \text{])}
\]

would plot the requested graph for the first column of `evecs`—though you should try to improve the appearance of the plot by tampering with the scales, adding labels, .... In particular, the statements

\[
\gg x = 0.0 : 1.0/11.0 : 1.0 \\
\gg \text{plot(} x, [ 0.0; \text{evecs(:,1); 0.0 } \text{])}
\]

will produce a graph whose horizontal axis is more suitably labeled. In such a display, you should see something close to the lowest several modes of a vibrating string fixed at both ends!

(d) (Optional) Repeat parts (b) and (c) but with a similarly constructed matrix that is 50 × 50.

**Solution:**

(a) To find the eigenvalues and eigenvectors of the matrix

\[
\begin{pmatrix}
0 & 1 & 0 \\
1 & 0 & 1 \\
0 & 1 & 0
\end{pmatrix}
\]

we must create it in MATLAB and then invoke `eig`. We use the statements

\[
\gg a = [[0,1,0];[1,0,1];[0,1,0]]; \\
\gg [\text{evecs,evals}]=\text{eig(a)}
\]

\[
\text{evecs} = 
\begin{pmatrix}
0.5000 & -0.7071 & 0.5000 \\
-0.7071 & 0.0000 & 0.7071 \\
0.5000 & 0.7071 & 0.5000
\end{pmatrix}
\]

\[
\text{evals} = 
\begin{pmatrix}
-1.4142 & 0 & 0 \\
0 & 0.0000 & 0 \\
0 & 0 & 1.4142
\end{pmatrix}
\]
where the eigenvectors are the columns of \texttt{evecs} and the eigenvalues are the diagonal elements of \texttt{evals}.

(b) To find the eigenvalues and eigenvectors of the described $10 \times 10$ matrix, we must first create it. We could, of course, enter each value individually as we did in part (a), but that is cumbersome for a matrix of the size we want here. Instead, we create a $10 \times 10$ array of zeroes and then use \texttt{for} loops to set the non-zero elements to the desired values. The statements

```matlab
>> b=zeros(10);
>> for i=1:10 b(i,i)=2; end
>> for i=1:9 b(i+1,i)=-1; end
>> for i=1:9 b(i,i+1)=-1; end
>> b
b =
  2 -1  0  0  0  0  0  0  0  0
 -1  2 -1  0  0  0  0  0  0  0
  0 -1  2 -1  0  0  0  0  0  0
  0  0 -1  2 -1  0  0  0  0  0
  0  0  0 -1  2 -1  0  0  0  0
  0  0  0  0 -1  2 -1  0  0  0
  0  0  0  0  0 -1  2 -1  0  0
  0  0  0  0  0  0 -1  2 -1  0
  0  0  0  0  0  0  0 -1  2  0
  0  0  0  0  0  0  0  0 -1  2
```

create the matrix and display it so we can verify that we have created it correctly.

Once the matrix has been created, we can invoke \texttt{eig} to find the eigenvalues and eigenvectors. We use the statement

```matlab
>> eig(b)
```

The eigenvalues and eigenvectors will be displayed.
>> [evecs,evals] = eig(b)
evecs =
Columns 1 through 7
 0.1201  -0.2305  -0.3223   0.3879  -0.4221   0.4221   0.3879
 0.2305  -0.3879  -0.4221   0.3223  -1.2010  -1.2010  -0.3223
 0.3223  -0.4221  -0.2305   0.3879  -0.3879   0.3879  -0.4221
 0.3879  -0.3223   0.1201  -0.4221   0.2305   0.2305   0.4221
 0.4221  -0.1201   0.3879  -0.2305  -0.3223   0.3223  -0.2305
 0.4221   0.1201   0.3879   0.2305  -0.3223  -0.3223  -0.3223
 0.3879  -0.3223   0.1201   0.4221   0.2305  -0.2305   0.4221
 0.3223  -0.4221  -0.2305  -0.1201   0.3879  -0.3879  -0.1201
 0.1201  -0.3223  -0.3879  -0.3879  -0.3223   0.3879   0.3879

Columns 8 through 10
 0.3223  -0.2305   0.1201
-0.4221   0.3879  -0.2305
 0.2305  -0.4221   0.3223
 0.1201   0.3223  -0.3879
-0.3879  -0.1201   0.4221
 0.3879  -0.1201  -0.4221
-0.1201  -0.3223   0.3879
-0.2305  -0.4221  -0.3223
 0.4221   0.3879   0.2305
-0.3223  -0.3205  -0.1201
evals =
Columns 1 through 7
 0.0810   0   0   0   0   0   0
 0  0.3175   0   0   0   0   0
 0   0  0.6903   0   0   0   0
 0   0   0  1.1692   0   0   0
 0   0   0   1.7154   0   0
 0   0   0   0   2.2846   0
 0   0   0   0   0   2.8238
 0   0   0   0   0   0   0
 0   0   0   0   0   0   0
 0   0   0   0   0   0   0

Columns 8 through 10
 0   0   0
 0   0   0
 0   0   0
 0   0   0
 0   0   0
 0   0   0
 0   0   0
 0   0   0
 0   0   0
 0   0   0
 0   0   0
 3.3097   0   0
 0   3.6825   0
 0   0   3.9190

where, once again, the eigenvectors are the columns of evecs and the eigenvalues are the diagonal elements of evals. Note that the eigenvalues are presented in increasing order.

(c) In part (b), we printed the eigenvalues of our 10 × 10 matrix. Because they have been presented in increasing order, the lowest six eigenvalues are in positions 1–6 along the diagonal of evals and
the eigenvectors themselves are in columns 1–6 of \texttt{evecs}. We plot each of these eigenvectors in turn with the statements

\begin{verbatim}
>> x=linspace(0.0,1.0,12);
>> for i=1:6
    subplot(2,3,i)
    plot(x,[0.0; evecs(:,i); 0.0],'Color','black','LineWidth',2)
    axis([0,1,-1,1])
    xlabel('Component number')
    ylabel('Eigenvector Component')
    title(['Mode ', int2str(i)],'FontSize',14)
end
\end{verbatim}

The results are shown in Fig. E3.2.

(d) The process for a $50 \times 50$ matrix is similar to that described in part (c). We create the matrix, find its eigenvalues and eigenvectors, and display the eigenvalues with the statements

\begin{verbatim}
>> for j=1:50 c(j,j)=2; end
>> for j=1:49 c(j+1,j)=-1; end
>> for j=1:49 c(j,j+1)=-1; end
>> [evecs,evals]=eig(c);
>> x=linspace(0.0,1.0,52);
>> for i=1:6
    subplot(2,3,i)
    plot(x,[0.0; evecs(:,i); 0.0],'Color','black','LineWidth',2)
    axis([0,1,-1,1])
    xlabel('Component number')
    ylabel('Eigenvector Component')
    title(['Mode ', int2str(i)],'FontSize',14)
end
\end{verbatim}
Figure E3.3: The lowest six eigenvectors for the $50 \times 50$ matrix.

```
end
>> subplot(1,1,1)
```

Again, the eigenvalues have been presented in increasing order. The resulting figure is shown in Figure E3.3. Note how closely these figures resemble the lowest modes of a vibrating string.
3.10 Stark Effect for \( n = 2 \) and \( n = 3 \)

**Exercise:** When a (weak) constant external electric field of magnitude \( F \)—we reserve \( E \) for energy in this exercise—is imposed on a hydrogen atom, the energies of the states with principal quantum number \( n \) shift from the energies given by the Bohr model by amounts determined by the eigenvalues of the matrix whose elements are \( \langle nlm|eF\hat{\bar{z}}|nl'm' \rangle \), where \( l, m, l', \) and \( m' \) range over all possible values of those quantum numbers allowed by the particular value of \( n \). If the states by which the rows and columns are labeled are ordered \( |2, 0, 0 \rangle, |2, 1, -1 \rangle, |2, 1, 0 \rangle, \) and \( |2, 1, 1 \rangle \), then the matrix for the state \( n = 2 \) is

\[
3e\alpha_0 F \begin{pmatrix}
0 & 0 & -1 & 0 \\
0 & 0 & 0 & 0 \\
-1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
\end{pmatrix}
\]

where \( e \) is the magnitude of the charge on the electron and \( \alpha_0 \) is the Bohr radius. Similarly, if the states by which the rows and columns are labeled are ordered \( |3, 2, 2 \rangle, |3, 1, 1 \rangle, |3, 2, 1 \rangle, |3, 0, 0 \rangle, |3, 1, 0 \rangle, |3, 2, 0 \rangle, |3, 1, -1 \rangle, |3, 2, -1 \rangle, \) and \( |3, 2, -2 \rangle \), then the matrix for the state \( n = 3 \) is

\[
3e\alpha_0 F \begin{pmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & -\frac{9}{2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & -3\sqrt{6} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & -\frac{9}{3}\sqrt{3} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & -\frac{9}{2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & -\frac{9}{2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\end{pmatrix}
\]

Find the eigenvalues and eigenvectors of these matrices. The eigenvalues give the energy shifts for the Stark effect for \( n = 2 \) and \( n = 3 \) and the eigenvectors give the linear combinations of the base states (i.e., the states in the absence of the external field) out of which the states in the presence of the field emerge as the field is turned on.

**Solution:** When \( n = 2 \), the matrix whose eigenvalues give the energy shifts for the Stark effect in hydrogen is created simply in MATLAB with the statements

```matlab
>> d = zeros(4,4); Create a 4 \times 4 matrix of zeros.
>> d(3,1) = -1; Set the element in column 3, row 1 = -1.
>> d(1,3) = -1 Set the element in column 1, row 3 = -1.

d =

0 0 -1 0
0 1 0 0
-1 0 0 0
0 0 0 0
```

We then find and display its eigenvalues and eigenvectors with the statements

```matlab
>> [ev, eval] = eig(d)
evecs =

-0.7071 0 0 -0.7071
0 1.0000 0 0
-0.7071 0 0 0.7071
0 0 1.0000 0
evals =

-1 0 0 0
0 0 0 0
0 0 0 0
0 0 0 1
```

Find the eigenvalues and eigenvectors of these matrices. The eigenvalues give the energy shifts for the Stark effect for \( n = 2 \) and \( n = 3 \) and the eigenvectors give the linear combinations of the base states (i.e., the states in the absence of the external field) out of which the states in the presence of the field emerge as the field is turned on.
Exercise 3.10  

(Note, incidentally, that 0.707107 = 1/\sqrt{2}.) Evidently, two states (\(\ket{2,0,-1}\), and \(\ket{2,1,1}\)) are not affected by the perturbation and two states,

\[
\frac{1}{\sqrt{2}} \left( \ket{2,0,0} \pm \ket{2,1,0} \right)
\]

are shifted, one up in energy by one unit (i.e., by 3eqaF) and the other down by one unit. The degeneracy is partly but not completely lifted by the application of the perturbation.

For the \(n = 3\) state, the method is identical but requires a bit more computational time. We start by creating our matrix with the statements

```matlab
>> f = zeros(9,9);
>> f(2,3) = -9/2; f(3,2) = -9/2;
>> f(4,5) = -3*sqrt(6); f(5,4) = -3*sqrt(6);
>> f(5,6) = -9/sqrt(3); f(6,5) = -9/sqrt(3);
>> f(7,8) = -9/2; f(8,7) = -9/2
f =
 0 0 0 0 0 0 0 0 0
 0 0 -4.5000 0 0 0 0 0 0
 0 -4.5000 0 0 0 0 0 0 0
 0 0 0 0 -7.3485 0 0 0 0
 0 0 0 -7.3485 0 -5.1962 0 0 0
 0 0 0 0 -5.1962 0 0 0 0
 0 0 0 0 0 0 0 -4.5000 0
 0 0 0 0 0 0 -4.5000 0 0
 0 0 0 0 0 0 0 0 0
```

We complete the task using the powerful `eig` function for the \(n = 3\) state. Incidentally, the `eig` function does not require the matrix to be symmetric or real to work. Again, we use the single statement

```matlab
>> [evecs, evals]=eig(f)
```

```
evecs =
 0 0 0 1.0000 0 0 0 0 0
 0 -0.7071 0 0 0 0 0 0 0
 0 -0.7071 0 0 0 0 0 0 0
 0.5774 0 0 0 0 0.5774 0 0 -0.5774
 0.7071 0 0 0 0 0.0000 0 0 0.7071
 0.4082 0 0 0 0 -0.8165 0 0 -0.4082
 0 0 -0.7071 0 0 0 -0.7071 0 0
 0 0 -0.7071 0 0 0 0.7071 0 0
 0 0 0 0 1.0000 0 0 0 0

evals =
-9.0000 0 0 0 0 0 0 0 0
 0 -4.5000 0 0 0 0 0 0 0
 0 0 -4.5000 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 4.5000 0 0
 0 0 0 0 0 0 0 4.5000 0
 0 0 0 0 0 0 0 0 9.0000
```

(The 0.0000 value in the eigenvalues is 0 and has been represented as a floating point decimal due to internal roundoff errors within the computations.) Again, recognize 0.7071 as \(\sqrt{2}/2\), 0.5774 as \(\sqrt{3}/3\), 0.4082 as \(\sqrt{6}/6\), and 0.8165 as \(\sqrt{6}/3\).
These eigenvalues and eigenvectors represent the energy shift and the linear combinations of the base states for the $n = 3$ state. Note that the groupings of the nine states in the eigenvectors are not as thoroughly jumbled as one might fear. The eigenvalues show that there is still some degeneracy within the system. Evidently, three of the nine states—those in columns 4, 5, and 6—are not shifted by the perturbation, two—those in columns 7 and 8—are shifted up by 4.5 units, two—those in columns 2 and 3—are shifted down by 4.5 units, one—in column 9—is shifted up by 9.0 units, and one—in column 1—is shifted down by 9.0 units. Given the order of the states in the statement of the exercise, we conclude from the eigenvectors that the states affected in various ways by the perturbation are given by the equations

<table>
<thead>
<tr>
<th>Energy Shift</th>
<th>Column</th>
<th>State(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>+9.0</td>
<td>9</td>
<td>(-0.58</td>
</tr>
<tr>
<td>+4.5</td>
<td>8</td>
<td>(-0.71</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>(-0.71</td>
</tr>
<tr>
<td>0</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>0.58</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>-4.5</td>
<td>2</td>
<td>(-0.71</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>(-0.71</td>
</tr>
<tr>
<td>-9.0</td>
<td>1</td>
<td>0.58</td>
</tr>
</tbody>
</table>
3.11 Damped and Undamped Sine Waves

**Exercise:** Create and test a sequence of MATLAB statements that will produce a graph that is similar to Fig.3.6 except that the axes are drawn along the lines $y = 0$ and $x = 0$.

**Solution:** We’ll create the requested graph using the commands

```matlab
>> dx = 20.0/100.0;
>> x = [0.0:dx:20];
>> sine = sin(x);
>> dampsine = exp(-x/10.0) .* sin(x);
>> plot( x, sine, 'color','black', 'linewidth',2 )
>> set( gca, 'fontsize', 16 )
>> title( 'Damped and Undamped Sine Waves', 'fontsize',20 )
>> hold on
>> plot( x, dampsine, 'color','black', 'linewidth',2 )
>> set( gca, 'xaxislocation', 'origin' )
>> box( 'off' )
>> hold off
```

The graph is shown in Fig. E3.4.
Exercise 3.13  Plotting the Potential of a Charged Disk

Exercise: Consider a circular disk of radius $a$ lying in the $xy$ plane with its center at the origin. If the disk carries a uniform charge on its surface, the electrostatic potential at the point $(0, 0, z)$ on the axis of the disk is given by

$$V(z) = E_0 \left[ \sqrt{a^2 + z^2} - |z| \right]$$

where $E_0$ is a constant. Obtain a graph of $V(z)/(E_0a)$ versus $z/a$.

Solution: A circular disk carrying a uniform surface charge lies with its center at the origin. The electrostatic potential at the point $(0, 0, z)$ on the axis of the disk is given by

$$V(z) = E_0 \left[ \sqrt{a^2 + z^2} - |z| \right]$$

where $a$ is the radius of the disk and $E_0$ is a constant. Introducing the dimensionless distance $d = z/a$, we have

$$V(z) = E_0 a \left[ \sqrt{1 + d^2} - |d| \right]$$

To make a graph of $V(z)/(E_0a)$, we divide the range $-10 \leq d \leq 10$ into 200 segments, calculate $V$, and plot the results with the statements

```matlab
>> zovera=[-10.0:0.1:10.0];
>> vbar=sqrt(1+zovera.^2)-abs(zovera);
>> plot(zovera,vbar,'Color','black','LineWidth',2)
>> title('On-Axis Electrostatic Potential','FontSize',14)
>> xlabel('{\it z}/\ita','FontSize',12)
>> ylabel('{\it V}(\itz)/({\it E}_0\ita)','FontSize',12)
```

Recall that MATLAB recognizes that `zovera` is an array and creates the corresponding array for `vbar` without explicit instruction. The graph resulting from these commands is shown in Fig. E3.5.

Figure E3.5: On axis electrostatic potential of a uniformly charged circular disk.
Exercise 3.14

3.14 A Charging Capacitor

**Exercise:** The voltage drop across an initially uncharged capacitor in a series RC circuit that is connected at time $t = 0$ to a battery is given by the expression

$$
\frac{V(t)}{V_0} = 1 - e^{-t/RC}
$$

Obtain a family of graphs showing $V(t) = V_0$ versus $t$ for various values of RC, and write a paragraph describing these graphs.

**Solution:** Let’s take five arbitrary values for $RC$, say the integers 1–5 for ease. Utilizing MATLAB’s ability to plot multiple graphs in the same window, then, we can show the time-variation of voltages for different values of $RC$ by executing the statements

```matlab
>> dt = 20.0/100.0;
>> t = [0.0 : dt : 20.0];
>> V1 = 1-exp(-t/1.0); V2 = 1-exp(-t/2.0); V3 = 1-exp(-t/3.0);
>> V4 = 1-exp(-t/4.0); V5 = 1-exp(-t/5.0);
>> plot( t, V1, 'color','black', 'linewidth',2 )
>> title( 'V(t)/V_0 versus t for various RC', 'fontsize',20 )
>> xlabel( 't', 'fontsize',16 )
>> ylabel( 'V(t)/V_0', 'fontsize',16 )
>> axis( [0.0,20.0, 0.0,1.5] )
>> hold on
>> plot( t, V2, 'color','black', 'linewidth',2, 'linestyle','--' )
>> plot( t, V3, 'color','black', 'linewidth',2, 'linestyle',':' )
>> plot( t, V4, 'color','black', 'linewidth',2, 'linestyle','-.' )
>> plot( t, V5, 'color','black', 'linewidth',2 )
>> text( 0.0, 1.1, 'RC = 1' )
>> text( 8.0, 0.7, 'RC = 5' )
```

The resulting graph is shown in Fig. E3.6. As the graph shows, having a larger value of $RC$ (whether through a larger resistance or a bigger capacitor) will lead to a longer time for the charge to build up. Also, note that even though the value of $RC$ can change, the value of $V(t)/V_0$ will always approach 1.0 asymptotically.
Figure E3.6: A graph showing voltage versus time for various values of $RC$ in an RC series circuit.
Exercise 3.15

3.15 Plotting the Output of a Fabry-Perot Interferometer

Exercise: In a Fabry-Perot interferometer, a very large number of waves, each out of phase with the previous one by an amount \( \delta \) and reduced in amplitude by a factor \( r, 0 \leq r < 1 \), interfere. The resulting intensity is proportional to the expression

\[
I(\delta) = \frac{1}{1 - 2r \cos \delta + r^2}
\]

Obtain graphs of \( I(\delta) \) versus \( \delta \) for various values of \( r \), and write a paragraph describing these graphs.

Solution: To evaluate the expression

\[
I(\delta) = \frac{1}{1 - 2r \cos \delta + r^2}
\]

for the intensity in the interference pattern of a Fabry-Perot interferometer, we first need to create a vector of values of \( \delta \). We elect to evaluate the function over the interval \(-3\pi \leq \delta \leq 3\pi \) and then to display it with the horizontal axis labeled in multiples of \( \pi \). We begin by telling MATLAB to form a vector of values for \( \delta \). Then we create the \texttt{for} loop to look at how the figures for a selected set of values of \( r \). This process is carried out with the statements

\begin{verbatim}
>> del = [-3.0 : 0.01 : 3.0];
>> delta = pi*del;
>> for j = 1:4
    r = (j-1)*0.3;
    I = 1./(1-2*r.*cos(delta)+r.^2);
    subplot(2,2,j);
    plot(del,I,'Color','black','LineWidth',2);
    axis([-3, 3, 0, 10]);
    xlabel('\delta', 'FontSize', 16);
    ylabel('{\itI}(\delta)','FontSize',16);
end
\end{verbatim}

Within the \texttt{for} loop the \texttt{subplot} command incorporates four graphs in a single figure. The plots created using this routine are included in Fig. E3.7.

As we might have expected, the graphs have peaks and valleys. As dictated by the periodicity of the cosine function, the period of \( I(\delta) \) is \( 2\pi \). When \( r = 0 \) the intensity is constant for all \( \delta \). As \( r \) gets larger the intensity varies more dramatically across the period. For example, if \( r = 0.9 \) (as in the final plot in Fig. E3.7), the intensity near \( \delta = 2\pi n \) (\( n \) denoting any integer) is more than ten-times the intensity at the points midway between the peaks. The greater \( r \) the more dramatic this intensity disparity is. In the limit as \( r \) approaches 1 the intensity at \( 2\pi n \) approaches infinity as the denominator in the original intensity equation approaches 0.
Figure E3.7: Showing $I(\delta)$ versus $\delta$ for values of $r = 0.0$ (upper left), 0.3 (upper right), 0.6 (lower left), and 0.9 (lower right). The horizontal axis is labeled in units of $\pi$. 
3.17 Plotting Relativistic Functions

Exercise: According to the special theory of relativity, the mass \( m \), the momentum \( p \), and the kinetic energy \( K \) of a particle moving with speed \( v \) are given in terms of the rest mass \( m_0 \) and the speed of light \( c \) by the equations

\[
m = \frac{m_0}{\sqrt{1 - \beta^2}}; \quad p = \frac{m_0 v}{\sqrt{1 - \beta^2}}; \quad K = \frac{m_0 c^2}{\sqrt{1 - \beta^2}} - m_0 c^2
\]

where \( \beta = v/c \). Obtain graphs of \( m/m_0 \), \( p/m_0 c \), and \( K/m_0 c^2 \) versus \( \beta \), superimposing on each a graph of the corresponding non-relativistic expression, and write a paragraph describing these graphs.

Solution: To compare relativistic and non-relativistic expressions of mass, momentum, and energy we must look at how these expressions change as velocity gets closer and closer to \( c \). To begin we create a variable \( \beta \), representing \( v/c \) and set its value to range from 0 to 1. Then we create expressions for the relativistic and non-relativistic representations of mass, momentum and energy using the MATLAB statements

\[
\begin{align*}
\text{beta} &= 0.0 : 0.01 : 1.0; \\
movermo &= 1.0./sqrt(1.0-beta.^2); \\
povermoc &= beta./sqrt(1.0-beta.^2); \\
kovermoc2 &= movermo-1.0; \\
movermon &= repmat( 1.0, 1, 101 ); \\
povermocn &= beta; \\
kovermoc2n &= 0.5*beta.^2;
\end{align*}
\]

Then we plot the relativistic expressions and overlay the non-relativistic on each plot with the commands

\[
\begin{align*}
\text{subplot}(1,3,1) \\
\text{plot}( \text{beta}, \text{movermo}, 'Color', 'black', 'LineWidth', 2 ) \\
\text{set}( \text{gca}, 'YLim', [ 0.0, 6.0 ] ) \\
\text{title}( 'Mass', 'FontSize', 16 ) \\
\text{xlabel}( 'v/c', 'FontSize', 14 ) \\
\text{ylabel}( 'm/m_0', 'FontSize', 14 ) \\
\text{hold on} \\
\text{plot}( \text{beta}, \text{movermon}, 'Color', 'black', 'LineWidth', 2, 'LineStyle', '--') \\
\text{hold off} \\
\text{subplot}(1,3,2) \\
\text{plot}( \text{beta}, \text{povermoc}, 'Color', 'black', 'LineWidth', 2 ) \\
\text{set}( \text{gca}, 'YLim', [ 0.0, 6.0 ] ) \\
\text{title}( 'Momentum', 'FontSize', 16 ) \\
\text{xlabel}( 'v/c', 'FontSize', 14 ) \\
\text{ylabel}( 'p/m_0 c', 'FontSize', 14 ) \\
\text{hold on} \\
\text{plot}( \text{beta}, \text{povermoc}, 'Color', 'black', 'LineWidth', 2, 'LineStyle', '--') \\
\text{hold off} \\
\text{subplot}(1,3,3) \\
\text{plot}( \text{beta}, \text{kovermoc2}, 'Color', 'black', 'LineWidth', 2 ) \\
\text{set}( \text{gca}, 'YLim', [ 0.0, 6.0 ] ) \\
\text{title}( 'K Energy', 'FontSize', 16 ) \\
\text{xlabel}( 'v/c', 'FontSize', 14 ) \\
\text{ylabel}( 'K/m_0 c^2', 'FontSize', 14 ) \\
\text{hold on} \\
\text{plot}( \text{beta}, \text{kovermoc2}, 'Color', 'black', 'LineWidth', 2, 'LineStyle', '--')
\end{align*}
\]
Figure E3.8: Relativistic versus non-relativistic dependence of mass ratio, momentum ratio, and energy ratio on \( v/c \). Relativistic functions are plotted using solid lines and non-relativistic functions are plotted using dashed lines.

\[
\begin{align*}
\text{Mass} & \quad \text{Momentum} & \quad \text{K Energy} \\
\frac{m}{m_0} & \quad \frac{p}{m_0 c} & \quad \frac{K}{m_0 c^2}
\end{align*}
\]

\[\begin{array}{c}
0 & 0.5 & 1 \\
0 & 0.5 & 1 \\
0 & 0.5 & 1
\end{array}\]

\[\begin{array}{c}
0 & 0.5 & 1 \\
0 & 0.5 & 1 \\
0 & 0.5 & 1
\end{array}\]

>> hold off

The result is shown in Figure E3.8.

The non-relativistic plots are close approximations to the relativistic plots for small values \( \beta \). But, for larger values of \( \beta \), the differences are more and more substantial as the relativistic values diverge toward infinity. From these plots one can gain insight as to why there had been no need for the theory of relativity until the beginning of the twentieth century. The effects of the theory are really not seen until one reaches speeds of around \( 0.2c \). It is certainly remarkable that Einstein was able to create this seemingly counterintuitive theory of relativity so early and with so few experimental results upon which to test his theory.
### 3.18 Plotting Transmission/Reflection of Thin Film

**Exercise:** In a vacuum, the transmission and reflection coefficients $T$ and $R$ of a dielectric film of thickness $d$ and index of refraction $n$ are given by the equations

$$T = \frac{4n^2}{4n^2 + (n^2 - 1)^2 \sin^2(\kappa d)}; \quad R = \frac{(n^2 - 1)^2 \sin^2(\kappa d)}{4n^2 + (n^2 - 1)^2 \sin^2(\kappa d)}$$

where $\kappa = \frac{2\pi n}{\lambda}$ and $\lambda$ is the wavelength of the wave in vacuum. Obtain graphs of $T$ and $R$ versus $\lambda/d$ for various values of $n$ and write a paragraph describing these graphs. **Warning:** Don’t try plotting too close to $\lambda = 0$ since the function $\sin(\kappa d)$ gives trouble at that point.

**Solution:** The coefficients of transmission and reflection of a dielectric film of thickness $d$ and index of refraction $n$ are given by the equations

$$T = \frac{4n^2}{4n^2 + (n^2 - 1)^2 \sin^2(2\pi n/(\lambda/d))}; \quad R = \frac{(n^2 - 1)^2 \sin^2(2\pi n/(\lambda/d))}{4n^2 + (n^2 - 1)^2 \sin^2(2\pi n/(\lambda/d))}$$

We want to look at what happens to the transmission and reflection coefficients as the ratio $\lambda/d$ changes for various indices of refraction, $n$. First, we create a vector of values for $\lambda/d$ at which to evaluate the transmission and reflection coefficients. Then, through the use of a for loop, we look at what happens to $T$ and $R$ as we change the indices of refraction. This task is accomplished with the statements

```matlab
>> lambdaoverd = [ 0.5:0.02:6.0 ];
```

Set vector to evaluate from 0.5 to 6.0 with 0.02 as the step size.

```matlab
>> for n = 1:4
    part1 = (n^2-1)^2*... (sin(2*pi*n./lambdaoverd)).^2;
    part2 = 4*n^2;
    denom = part1+part2;
    T = part2./denom;
    subplot(2,2,n);
    plot(lambdaoverd, T, 'k--')
    axis([0,6,-0.5,1.5]);
    hold on
    plot(lambdaoverd, R, 'k')
    hold off
    xlabel('\lambda/{\it d}');
    ylabel('{\it T} and {\it R}');
end
```

Here, the subplot command is used to write four graphs to a single figure. The resulting plots are included in Figure E3.9.

As $\lambda/d$ increases, the graphs indicate that the reflection coefficient oscillates between 0 and 0.5 plus some value, $a$, and the transmission coefficient oscillates between 1 and $0.5 - a$. The two graphs are reflections of one another over the axis-line $y = 0.5$. This confirms that $R + T = 1$. Also, as the ratio $\lambda/d$ becomes larger the oscillations occur less frequently and the sine wave oscillation stretches out. Further, as the index of refraction increases, the oscillations stretch out at a slower rate.
Exercise 3.18

Figure E3.9: Showing the values of the transmission (dashed line) and reflection (solid line) coefficients versus $\lambda/d$ of a dielectric film of thickness $d$. $\lambda$ is the wavelength of the wave in a vacuum. The graphs depicted represent indices of refraction of $n = 1$ (upper left), 2 (upper right), 3 (lower left), and 4 (lower right).
3.19 Plotting the Potential of Two Charged Disks

**Exercise:** Consider two circular disks, each of radius \( R \), located with their centers on the \( z \) axis such that their planes are parallel to the \( xy \) plane. Let the first disk have its center at the point \((0,0,b/2)\) and the second at the point \((0,0,-b/2)\) so that the disks are separated by a distance \( b \) \((b > 0)\) and the origin is halfway between them. If the top disk carries a uniform, constant charge density \( \sigma \) and the bottom disk carries a uniform, constant charge density \(-\sigma\), the electrostatic potential at the point \((0,0,z)\) is given by

\[
V(z) = \frac{\sigma}{2\epsilon_0} \left[ \sqrt{R^2 + \left( z - \frac{b}{2} \right)^2} - \left| z - \frac{b}{2} \right| - \sqrt{R^2 + \left( z + \frac{b}{2} \right)^2} + \left| z + \frac{b}{2} \right| \right]
\]

Obtain graphs of \( V(z)/\left(\sigma R/2\epsilon_0\right) \) versus \( z/R \) for various values of \( b/R \) and write a paragraph describing these graphs.

**Solution:** A graphical representation of the physical system explored in this exercise is included in Fig. E3.10. Starting with the expression

\[
V(z) = \frac{\sigma}{2\epsilon_0} \left[ \sqrt{R^2 + \left( z - \frac{b}{2} \right)^2} - \left| z - \frac{b}{2} \right| - \sqrt{R^2 + \left( z + \frac{b}{2} \right)^2} + \left| z + \frac{b}{2} \right| \right]
\]

we factor an \( R \) out of each term on the right to find that

\[
\frac{V(z)}{\sigma R/2\epsilon_0} = \left[ \sqrt{1 + \left( \frac{z}{R} - \frac{b}{2R} \right)^2} - \left| \frac{z}{R} - \frac{b}{2R} \right| - \sqrt{1 + \left( \frac{z}{R} + \frac{b}{2R} \right)^2} + \left| \frac{z}{R} + \frac{b}{2R} \right| \right]
\]

Then, if we introduce the dimensionless variables \( \overline{z} = z/R, \overline{b} = b/R \), and \( \overline{V} = V(z)/(\sigma R/2\epsilon_0) \), we

Figure E3.10: The setup of problem 3.18.
Exercise 3.19

Figure E3.11: Showing plots of $V(z)/(\sigma R/2\epsilon_0)$ versus $z/R$ for values of $b/R = 0.1$ (upper left), 0.4 (upper middle), 0.9 (upper right), 1.6 (lower left), 2.5 (lower middle), and 3.6 (lower right).

We then evaluate this expression as a function of $z/R$ for various values of $b/R$ with the statements

```matlab
>> zbar = [ -4.0:0.1:4.0 ];
> bbar=i^2*0.1;
> Vbar=sqrt(1+(zbar-bbar/2).^2)...
> -abs(zbar-bbar/2)...
> -sqrt(1+(zbar+bbar/2).^2)...
> +abs(zbar+bbar/2);

Setup vector to evaluate $z/R$ from -4.0 to 4.0 with 0.1 as the step size.
Assign values for $b/R$ (0.1, 0.4, 0.9, 1.6, 2.5 and 3.6).
Evaluate the dimensionless equation.
```

The resulting plots produced with this method are included in Fig. E3.11.

The graphs show that the dimensionless electrostatic potential of the system for small $b/R$ is approximately linear from $z/R = 0$ to $z/R = \pm b/2R$ (inside the charged plates). Outside the plates, the potential falls off to approach zero as one moves away from the system. For larger $b/R$, the potential outside the two charged plates still falls off as expected. However, between the plates, the potential plot begins to look like a cubic function rather than a linear one. Regardless, for all $b/R$, the dimensionless potential is maximized at $z/R = b/2R$ and is minimized at $z/R = -b/2R$. The larger the ratio of $b/R$ the larger the dimensionless electrostatic potential can be.
### Exercise 3.22

**Exercise:** When a photon of initial energy $E_0$ undergoes Compton scattering from an atom of mass $m$ and is scattered by an angle $\theta$, the energy of the photon is reduced to

$$E(\theta) = \frac{E_0}{1 + \xi (1 + \cos \theta)}$$

where $\xi = E_0 / mc^2$. Obtain both Cartesian and polar graphs of $E(\theta)/E_0$ versus $\theta$, $-\pi \leq \theta \leq \pi$, for several values of $\xi$, and write a paragraph describing these graphs.

**Solution:**

To graph the function

$$\frac{E(\theta)}{E_0} = \frac{1}{1 + \xi (1 + \cos \theta)}$$

over the interval $-\pi \leq \theta \leq \pi$, we begin by establishing a vector with values of the independent variable. For convenience in labeling the axes, we elect to generate an array both of values ranging from $-1$ to $+1$ and values ranging from $-\pi$ to $+\pi$ by invoking the statements

```matlab
>> x = [-1.0 : 0.02 : 1.0];
>> theta = pi*x;
```

Then, we set a value for the parameter $\xi$, evaluate the function, and plot the values with the statements

```matlab
>> xi = 0.5;
>> E = 1.0./(1.0+xi.*(1.0+cos(theta)));
>> plot( x, E, 'color', 'black', 'linewidth', 3.0 )
>> hold on
```

The resulting graph is combined with others in Fig. E3.12. With this graph alone, we are led to explore other values of the parameter $\xi$ and to construct a composite graph them all with the statements

```matlab
>> xi = [ 0.1, 0.2, 0.3, 0.75, 1.0, 2.0, 4.0 ];
>> for i = 1:7
    E = 1.0./(1.0+xi(i).*(1.0+cos(theta)));
    plot( x, E, 'color', 'black', 'linewidth', 3.0 )
end
```

Whatever the ratio of photon energy to atomic rest energy, the photon energy is reduced by the largest fraction for forward scattering ($\theta = 0$) and is not reduced at all for backscattering ($\theta = +\pi$ or $-\pi$). Further, the greater the photon energy, the greater its percentage loss of energy at any scattering angle.

We can also create polar graphs of this function with the statements

```matlab
>> xi = 0.5;
>> E = 1.0./(1.0+xi.*(1.0+cos(theta)));
>> h1 = polar( theta, E );
```
Figure E3.12: $E(\theta)/E_0$ versus $\theta$ for various values of $\xi = v/c$. 

This output is shown in Fig. E3.13. The conclusions we draw are the same as those drawn at the end of the previous paragraph.
3.23 Field of a Moving Charge

Exercise: A charged particle moves along the z-axis with speed \( v \). When the particle passes through the origin, the magnitude of the electric field produced by the particle is given by the expression

\[
E(\theta) = \frac{q}{4\pi\varepsilon_0 r^2} \frac{1 - \beta^2}{(1 + \beta^2 \sin^2(\theta))^{3/2}}
\]

where \( \theta \) is the polar angle of the observation point, \( r \) is the radial coordinate at that point, and \( \beta = v/c \). (\( c \) is the speed of light.) Obtain graphs of \( E(\theta)/E(0) \) versus \( \theta \) and, to reveal some of the behavior more visibly, of \( E(\theta)/E(0) \) versus \( \theta \) for various values of \( \beta \) on the interval \( \pi \leq \theta \leq \pi \), and write a paragraph describing these graphs.

Solution: Since we are asked to obtain (dimensionless) graphs of \( E/(q/4\pi\varepsilon_0 r^2) \) versus values of \( \theta \), and we know that \( \beta < 1 \) always (since \( v \leq c \)), we can simply write a sequence of MAPLE statements that will take a bunch of \( \beta \)'s and output graphs of the E-fields using the code

```maple
>> dtheta = 2.0*pi/100.0;
>> theta = dtheta*[0:1:100] - pi;
>> thetapl = 2.0/100.0 * [0:1:100] - 1;
>> beta = [ 0.2, 0.4, 0.6, 0.8, 0.99];
>> E = (1 - beta(1)^2) ./ (1 + beta(1)^2*sin(theta).^2).^1.5;
>> plot( thetapl, E, 'color','black', 'linewidth',2 )
>> hold on
>> for i = 2:5
    E = (1 - beta(i)^2) ./ (1 + beta(i)^2*sin(theta).^2).^1.5;
    plot( thetapl, E, 'color','black', 'linewidth',2 )
end
>> title( 'E Field of a Moving Particle', 'fontsize',20 )
>> xlabel( '\\theta/\pi', 'fontsize', 16 )
>> ylabel( 'E/(q/4\pi \varepsilon_0 r^2)', 'fontsize',16 )
>> text( 0.35, 0.85, '\beta=0.2', 'fontsize',14 )
>> text( 0.35, 0.75, '\beta=0.4', 'fontsize',14 )
>> text( 0.35, 0.50, '\beta=0.6', 'fontsize',14 )
>> text( 0.35, 0.25, '\beta=0.8', 'fontsize',14 )
>> text( 0.35, 0.05, '\beta=0.99', 'fontsize',14 )
>> hold off
```

When executed, will produce the graph shown in Fig. E3.14. The statements above initializing \( \theta \) and \( \theta_{\text{tapl}} \), while appearing quite arbitrary, are actually used for scaling. As you can see, our `plot` command is invoked with `\theta_{\text{tapl}}` as the first argument, while \( E \) is calculated using `\theta`. This will allow the plots to fill the entire graph, since MAPLE tends to use "nice" numbers for its default scaling.

Most interesting to note is that as \( v \) increases, the electric field observed becomes weaker and, as \( v \) moves away from \( v \approx c/2 \) the perturbations in \( E \) become weaker and weaker.

Now, we will do another plot, this time of the equation

\[
\frac{E(\theta)}{E(0)} = \frac{1}{(1 + \beta^2 \sin^2(\theta))^{3/2}}
\]

where \( E(0) \) is

\[
E(0) = \frac{q}{4\pi\varepsilon_0 r^2}(1 - \beta^2)
\]

which is the electric field evaluated at \( \theta \) equal to zero. We used the statements
Figure E3.14: Various values of $E/(q/4\pi\epsilon_0r^2)$ vs $\theta$ for different $\beta$

$$E = \frac{1}{(1 + \beta^2 \sin^2 \theta)^{1.5}};$$

plot( thetapl, E, 'color','black', 'linewidth',2 )
>> hold on
>> for i = 2:5
    E =1.0 ./ (1 + beta(i)^2*sin(theta).^2).^1.5;
    plot( thetapl, E, 'color','black', 'linewidth',2 )
end
>> title( 'E(\theta)/E(0) versus \theta', 'fontsize',20 )
>> axis( [-1.0,1.0,0.0,1.0])
>> xlabel( '\theta/\pi', 'fontsize', 16 )
>> ylabel( 'E(\theta)/E(0)', 'fontsize',16 )
>> text( 0.4, 0.9, '\beta=0.2', 'fontsize',14 )
>> text( 0.4, 0.75, '\beta=0.4', 'fontsize',14 )
>> text( 0.4, 0.59, '\beta=0.6', 'fontsize',14 )
>> text( 0.4, 0.43, '\beta=0.8', 'fontsize',14 )
>> text( 0.4, 0.30, '\beta=0.99', 'fontsize',14 )
>> hold off

to create the graph shown in Fig. E3.15. The troughs still occur at $\theta = \pi/2$ and $\theta = -\pi/2$, and even though the absolute magnitude of the electric field decreases as $\beta$ increases, the proportion between $E(\theta)$ and $E(0)$ increases dramatically. This type of phenomenon is similar to the “headlight” effect.

A similar plot, though in polar coordinates to better show the relationship between $E(\theta)/E(0)$ and $\theta$, can be viewed in Fig. E3.16. The polar plot can be produced with the coding

$$E = \frac{1}{(1 + \beta^2 \sin^2 \theta)^{1.5}};$$

>> h1 = polar( theta, E )
>> set( gca, 'fontsize',16)
>> set( h1, 'color','black', 'linewidth',2 )
>> hold on
Exercise 3.23

Figure E3.15: Various values of $E(\theta)/E(0)$ vs $\theta$ for different $\beta$

$$E(\theta)/E(0) \text{ versus } \theta$$

>> for i = 2:5
    E =1.0 ./ (1 + beta(i)^2*sin(theta).^2).^1.5;
    h1 = polar( theta, E )
    set( h1, 'color','black', 'linewidth',2 )
end
>> title( 'Polar graph of $E(\theta)/E(0)$ versus $\theta$')
Figure E3.16: Various values of $E(\theta)/E(0)$ vs $\theta$ for different $\beta$ in polar coordinates. The innermost solid line corresponds to the value $\beta = 0.99$ and the outermost solid line corresponds to the value $\beta = 0.2$. 

Polar graph of $E(\theta)/E(0)$ versus $\theta$
3.24 Plotting the Four-Slit Interference Pattern

**Exercise:** The intensity of the interference pattern produced by four slits illuminated by light of wavelength $\lambda$ when each slit is separated from the next by a distance $a$ is given by

$$I(\delta) = \cos^2 \delta (1 + \cos \delta)$$

where $\delta = (2\pi a \sin \theta)/\lambda$. Obtain both Cartesian and polar graphs of $I$ versus $\theta$ on the interval $-\pi/2 \leq \theta \leq \pi/2$ for various values of $\lambda/a$, and write a paragraph describing these graphs.

**Solution:** When illuminated by a wave of wavelength $\lambda$, an array of four slits, each a distance $a$ from its nearest neighbors, generates interesting interference patterns. To explore these patterns computationally, we begin by forming a vector of values for $\theta$ ranging from $-\pi/2$ to $\pi/2$ and, so the plotting can be done on a horizontal scale labeled in multiples of $\pi$, an identically sized vector of values ranging from $-0.5$ to $+0.5$. To that end, we invoke the statements

```matlab
>> dt = 1.0/1000.0;
>> xaxis = [-0.5 : dt : 0.5];
>> theta = pi * xaxis;
```

where, because we anticipate a function that varies rapidly in the interval $-\pi/2 \leq \theta \leq \pi/2$, we put what might seem to be an unusually large number of points in the interval. Then, we form a vector of values for $\lambda/a$ that we want to look at. Here, we create an appropriate vector explicitly with the statement

```matlab
>> lambdaa = [ 0.1, 0.25, 0.5, 1.0, 2.0, 5.0 ];
```

Finally, we structure a `for` loop to define $\delta$ and $I$, and plot $I$ versus $\theta$ (in units of $\pi$) for each value of $\lambda/a$ in the vector `lambdaa`. In Cartesian coordinates these operations are accomplished with the statements

```matlab
>> for i = 1:6
    delta=2.0*pi*sin(theta)/lambdaa(i);
    I=(cos(delta)).^2.0.*(1.0+cos(delta));
    subplot(2,3,i)
    plot(xaxis,I, 'Color', 'black')
    title(lambdaa(i))
end
```

The resulting plots are shown in Fig. E3.17. One can see from these plots that increasing the ratio $\lambda/a$ reduces the number of principal maxima and widens those peaks. The plots also show that there are two secondary maxima between consecutive principal maxima.

Another way to look at the equation is in polar form. To do this we need only change the plotting statement in the previous loop. Thus, the loop

```matlab
>> for i = 1:6
    delta=2.0*pi*sin(theta)/lambdaa(i);
    I=(cos(delta)).^2.0.*(1.0+cos(delta));
    subplot(2,3,i)
    hdl=polar(theta,I);
    set(hdl, 'Color', 'black', 'LineWidth', 1.75)
    title(lambdaa(i), 'FontSize', 16)
end
```

provides an interesting polar representation of the intensity patterns, shown in Fig. E3.18. These graphs don’t show the secondary maximum as clearly as we might have hoped.
Figure E3.17: Cartesian plots of the intensity versus angle of a four-slit interference pattern for various values of $\lambda/a$. The $x$-axis contains values for the angle ($\theta$) and the $y$-axis contains values for the intensity ($I$).
Figure E3.18: Polar plots of the intensity versus angle of a four-slit interference pattern for various values of $\lambda/a$. 
Exercise 3.25  Plotting the Planck Blackbody Radiation Curve

Exercise: The Planck radiation law gives the expression

\[ u(\lambda, T) = \frac{8\pi c h}{\lambda^5} \frac{1}{e^{c h / (\lambda k T)} - 1} \]

for the distribution of energy in the radiation emitted by a black body. Here, \( c \) is the speed of light, \( h \) is Planck’s constant, \( k \) is Boltzmann’s constant, \( \lambda \) is the wavelength of the radiation, and \( T \) is the absolute temperature. Using appropriate dimensionless units, plot this function (a) as a function of \( \lambda \) for several \( T \) and (b) as a surface over the \( \lambda T \)-plane. Write a paragraph about the way the peak changes in position, height, and width as \( T \) changes. Hint: Choose a reference wavelength \( \lambda_0 \) arbitrarily and recast the expression in terms of the dimensionless variable \( \Lambda = \lambda / \lambda_0 \). Then, note that \( T_0 = ch / (\lambda_0 k) \) has the dimensions of temperature and re-express the temperature \( T \) in terms of the dimensionless quantity \( \tau = T / T_0 \). (You might find it informative to evaluate \( T_0 \) for \( \lambda_0 = 550 \text{ nm} \).) With these changes, the expression to be plotted can be recast in the form

\[ \frac{u(\lambda, T)}{8\pi c h / \lambda_0^5} = \frac{1}{\Lambda^5} \frac{1}{e^{1/\Lambda^\tau} - 1} \]

and the question now becomes one of plotting this quantity using the dimensionless variables \( \Lambda \) and \( \tau \).

Solution: This exercise is best begun by casting the expression

\[ u(\lambda, T) = \frac{8\pi c h}{\lambda^5} \frac{1}{e^{c h / (\lambda k T)} - 1} \]

in a dimensionless form. Rather than exploiting the dimensionless casting based on the arbitrarily selected wavelength suggested in the statement of the exercise, we elect instead to choose an arbitrary reference temperature \( T_0 \) and introduce the dimensionless temperature \( T = T / T_0 \) so that \( T = T_0 T \). Then, note that

\[ \frac{c h}{k T} = \frac{c h}{k T_0 T} \]

from which we infer that the quantity \( c h / k T_0 \) might make a suitable unit for the measurement of \( \lambda \). Checking the units of this quantity, we find

units of \( \frac{c h}{k T_0} = (\text{m/s}) \times (\text{J s}) / (\text{J/K}) \times \text{K} = \text{m} \)

(Here, \( J = \text{Joule} \) and \( K = \text{Kelvin} \).) Thus, we choose a reference length \( \lambda_0 = c h / k T_0 \) and introduce the dimensionless wavelength \( \tilde{\lambda} \) so that \( \lambda = \lambda_0 \tilde{\lambda} \). With this notation, we then have that

\[ u(\lambda, T) = \frac{8\pi c h}{\lambda_0^5 \tilde{\lambda}} \frac{1}{e^{1/\tilde{\lambda}^\tau} - 1} \]

As the final step in this preparation, we then elect to plot

\[ \overline{u} = \frac{u}{8\pi c h / \lambda_0^5} = \frac{1}{\tilde{\lambda}^5} \frac{1}{e^{1/\tilde{\lambda}^\tau} - 1} \]

Note before we go on that, if the reference temperature is 1000 K, then the reference length is

\[ \lambda_0 = \frac{c h}{k T_0} = \frac{(3 \times 10^8) \text{ m/s} \times (6.6 \times 10^{-34}) \text{ J s}}{(1.38 \times 10^{-23}) \text{ J/K} \times 10^3 \text{ K}} = 14.3 \times 10^{-6} \text{ m} = 14300 \text{ nm} \]

A quick scaling of this result reveals that for \( T_0 = 2500 \text{ K} \) (approximately the temperature of a tungsten light filament), \( \lambda_0 = 5720 \text{ nm} \) and for \( T_0 = 6000 \text{ K} \) (approximately the temperature of the surface of the sun), \( \lambda_0 = 2380 \text{ nm} \).
One of the advantages of casting things in a dimensionless form is that we don’t have many powers of ten (positive or negative) to keep track of. Further, it is usually the case that the significant values of dimensionless parameters are on the order of 1.0. Consequently, a sensible first point of exploration here would be to take $T = 1.0$ and allow $\lambda$ to range over the interval $0.0 \leq \lambda \leq 10.0$. This we do with the statements

```plaintext
>> T = 1.0           ; Drop the overbars
>> lambda = [0.0 : 0.05 : 10.0] ; Interval from 0.0 to 10.0, steps 0.01

>> u = (1.0 ./ lambda.^5) ./ ( exp(1.0 ./ (lambda.*T)) - 1.0 )

>> plot( lambda, u, 'Color', 'black', 'LineWidth', 3 )
```

The resulting graph, shown in Fig. E3.19, reveals that the interesting things happen more in the range $0 < \lambda < 2.0$, so we change the scale on the horizontal axis to run from 0.0 to 2.0 in steps of 0.02 and repeat the process with the statements

```plaintext
>> T = 1.0;
>> lambda = [0.0 : 0.02 : 2.0];
>> u = (1.0 ./ lambda.^5) ./ ( exp(1.0 ./ (lambda.*T)) - 1.0 );
>> plot( lambda, u, 'Color', 'black', 'LineWidth', 3 )
>> text(0.2, 22.0, 'T = 1.0', 'FontSize', 16 )
```

Then, with the similar statements,

```plaintext
>> T = 0.5;
```
Figure E3.20: Second graph of \( u \) versus \( \lambda \), this time for the several indicated values of \( T \). These graphs are drawn to a better scale than the first graph.

\[
\begin{align*}
\text{>> } & \text{u = (1.0 ./ lambda.^5) ./ (exp(1.0 ./ (lambda.*T)) - 1.0);} \\
\text{>> } & \text{hold on} \\
\text{>> } & \text{plot( lambda, u, 'Color', 'black', 'LineWidth', 3) } \\
\text{>> } & \text{text(1.0, 2.0, 'T = 0.5', 'FontSize', 16)} \\
\text{we add a graph for } & \text{T = 0.5, and quickly conclude that the next graph should be at a value of } T \text{ between } T = 1.0 \text{ and } T = 0.5. \text{ We choose } T = 0.75 \text{ and add its graph with the statements} \\
\text{>> } & \text{T = 0.75;} \\
\text{>> } & \text{u = (1.0 ./ lambda.^5) ./ (exp(1.0 ./ (lambda.*T)) - 1.0);} \\
\text{>> } & \text{plot( lambda, u, 'Color', 'black', 'LineWidht', 3) } \\
\text{>> } & \text{text(0.7, 5.0, 'T = 0.75', 'FontSize', 16)} \\
\text{Finally, we put in the graph for } & \text{T = 0.9 with the statements} \\
\text{>> } & \text{T = 0.9;} \\
\text{>> } & \text{u = (1.0 ./ lambda.^5) ./ (exp(1.0 ./ (lambda.*T)) - 1.0);} \\
\text{>> } & \text{plot( lambda, u, 'Color', 'black', 'LineWidht', 3) } \\
\text{>> } & \text{text(0.5, 13.0, 'T = 0.9', 'FontSize', 16)} \\
\text{>> } & \text{hold off}
\end{align*}
\]

From these graphs (see Fig. E3.20), we note that, as \( T \) decreases from the reference temperature, \( T_0 \) (whatever we chose), the peak becomes broader and lower, and the value of \( \lambda \) at which it occurs moves to longer wavelengths. At \( T = T_0 \), the position of the peak is approximately at \( \lambda = 0.2 \lambda_0 \). Thus, we conclude that
Figure E3.21: A surface graph of the function $u(\lambda, T)$ over the $\lambda T$ plane.

Another way to get a good idea of how a system of three variables behaves is with a surface plot. To do this MATLAB needs to evaluate a function over a grid of points. The MATLAB function `meshgrid` is used to make this grid of points and the commands `mesh` and `surf` are used to make a meshed and smoothed surface for a function that has been evaluated over this grid of points. For this problem, we create a meshed surface with the statements

Finally, we draw the requested surface plot over the range $0.4 \leq T \leq 1.0$ and $0.0 \leq \lambda \leq 1.0$ with the IDL commands

The resulting figure is shown in Fig. E3.21. Note that we have put more points in the interval on $\lambda$ than in the interval on $T$ because the function varies more rapidly with $\lambda$ than with $T$. 

peak at 1000 K $0.2(14300)$ nm $= 2860$ nm
peak at 2500 K $0.2(5720)$ nm $= 1144$ nm
peak at 6000 K $0.2(2380)$ nm $= 476$ nm
3.26 Plotting the On-Axis Field of a Solenoid

Exercise:  A solenoid of length $L$ and circular cross-section of radius $a$ lies with its axis along the $z$ axis and its center at the origin. When the solenoid carries a current, the magnetic field at the point $(0,0,z)$ on the axis of the solenoid is given by

$$B(z) = \frac{1}{2} B_0 \left[ \frac{z + L/2}{\sqrt{a^2 + (z + L/2)^2}} - \frac{z - L/2}{\sqrt{a^2 + (z - L/2)^2}} \right]$$

where $B_0$ is the magnetic field at the center when $a \ll L$, i.e., when the solenoid is effectively infinite in length. Plot graphs showing $B(z)/B_0$ as a function of $z/L$ for various values of $a/L$, (b) as a surface over the $(z/L)(a/L)$ plane, and (c) as a contour over the $(z/L)(a/L)$ plane. Write a paragraph describing these graphs.

Solution:  Choosing $L$ as the unit of length, we introduce $\bar{z} = z/L$, $\bar{\sigma} = a/L$, and $\bar{B} = B/B_0$ to find that we can recast the given expression for the magnetic field in the form

$$\bar{B} = \frac{1}{2} \frac{\bar{z} + 1/2}{\sqrt{\bar{\sigma}^2 + (\bar{z} + 1/2)^2}} - \frac{\bar{z} - 1/2}{\sqrt{\bar{\sigma}^2 + (\bar{z} - 1/2)^2}}$$

In MATLAB, we will drop the over bars. Then, recognizing that the interesting region for this magnetic field will be inside and just outside of the solenoid (i.e., the region $-2.0 \leq \bar{z} \leq 2.0$, say), we might, as an exploratory pass, calculate the values of $\bar{B}$ as a function of $\bar{z}$ for $\bar{\sigma} = 1$ and plot the result quickly with the MATLAB statements

```matlab
>> inc=4.0/1000.0;
>> z=[-2.0:inc:2.0];
>> a=1.0;
>> B=0.5*(z+0.5)./sqrt(a^2+(z+0.5).^2) - 0.5*(z-0.5)./sqrt(a^2 + (z-0.5).^2);
>> plot(z,B, 'Color', 'black', 'LineWidth', 2)
```

Note that we have written 0.5 rather than 1.0/2.0 for one half so that MATLAB doesn’t recompute its value every time the expression is evaluated.

The graph resulting from the above statements is shown in Fig. E3.22. It suggests that we have the scaling about right and that the field is strong inside the solenoid but falls fairly quickly as we move away from the center of the solenoid at $\bar{z} = 0$ towards the top or bottom edges of the solenoid located at $\bar{z} = \pm 0.5$.

On the basis of this graph (and some further exploration that reveals that the peak at $\bar{z} = 0$ is highest for the smallest value of $a$, we then produce the final graph showing $\bar{B}$ as a function of $\bar{z}$ with the statements

```matlab
>> a = 0.05;
>> B1=0.5*(z+0.5)./sqrt(a^2+(z+0.5).^2) - 0.5*(z-0.5)./sqrt(a^2 + (z-0.5).^2);
>> a = 0.1;
>> B2=0.5*(z+0.5)./sqrt(a^2+(z+0.5).^2) - 0.5*(z-0.5)./sqrt(a^2 + (z-0.5).^2);
>> a = 0.5;
>> B3=0.5*(z+0.5)./sqrt(a^2+(z+0.5).^2) - 0.5*(z-0.5)./sqrt(a^2 + (z-0.5).^2);
>> a = 1.0;
>> B4=0.5*(z+0.5)./sqrt(a^2+(z+0.5).^2) - 0.5*(z-0.5)./sqrt(a^2 + (z-0.5).^2);
>> a = 2.0;
>> B5=0.5*(z+0.5)./sqrt(a^2+(z+0.5).^2) - 0.5*(z-0.5)./sqrt(a^2 + (z-0.5).^2);
>> plot(z,B1,'Color','black', 'LineWidth',2 )
```
Exercise 3.26

>> title('Magnetic Field of a Solenoid', 'FontSize', 20)
>> xlabel('z/L','FontSize',14)
>> ylabel('B/B0','FontSize',14)
>> hold on;
>> plot(z,B2,'Color','black', 'LineWidth',2)
>> plot(z,B3,'Color','black', 'LineWidth',2)
>> plot(z,B4,'Color','black', 'LineWidth',2)
>> plot(z,B5,'Color','black', 'LineWidth',2)

and use the statements

>> plot([0.5,0.5], [0.0,1.0],'LineStyle', '--', 'Color','black')
>> plot([-0.5,-0.5], [0.0,1.0],'LineStyle', '--', 'Color','black')
>> hold off;

to add to the graph two vertical lines at the position \( \bar{z} = \pm 0.5 \) of the top and bottom edges of the solenoid.

The graph resulting from these statements is shown in Fig. E3.23. Note that, when the solenoid is long and slender (\( a \ll L, \pi \ll 1 \)), the curve—the highest curve—has a fairly flat top and the field remains close to the value at the center of the solenoid for a significant distance along the axis. As the solenoid becomes “squatter”, the region of nearly constant magnetic field becomes smaller and, at the other extreme (\( a \gg L, \pi \gg 1 \)), the magnetic field varies more like that of a single loop, to which the solenoid reduces when its radius is distinctly larger than its length.

In the above, we avoided a trouble spot. Now let’s address that spot. Suppose we set \( a = 0 \), i.e., \( L = \infty \). We can readily evaluate \( \overline{B} \) with the statements

>> a=0.0;
>> B=0.5*(z+0.5)./sqrt(a^2+(z+0.5).^2) - 0.5*(z-0.5)./sqrt(a^2 +(z-0.5).^2)

Figure E3.22: A first pass at a graph of the on-axis magnetic field of a solenoid.
Figure E3.23: Magnetic field of solenoid for $a/L = 0.05$ (highest curve), 0.1, 0.5, 1.0, and 2.0 (lowest curve).

but we are presented with a warning message that the program was asked to divide by zero. Because we did not put a semicolon at the end of the statement $B = \ldots$, MATLAB printed 1001 numeres which are not here reproduced. Interestingly, with two exceptions [B(376) and B(626)], the numbers are all either 0 or 1, 0 outside the solenoid (which region, of course, fails to exist if $L = \infty$) and 1 inside the solenoid. The problems with evaluation come at the values $z = \pm 0.5$, where one or the other of the terms giving $B$ becomes the indeterminate quantity 0/0. The printout of the numbers contains the value NaN—not a number—at two points. Since the plot command knows to ignore the value NaN, we can simply plot the graph for this case and again mark the position of the ends of the solenoid with the statements

```matlab
>> plot(z,B,'Color', 'black', 'LineWidth', 6)
>> hold on
>> plot([0.5,0.5], [0.0,1.0], 'LineStyle','--', 'Color', 'black', 'LineWidth', 2)
>> plot([-0.5,-0.5], [0.0,1.0], 'LineStyle','--', 'Color', 'black', 'LineWidth', 2)
>> hold off
```

finding the result in Fig. E3.24. This graph reveals that, for the infinitely long solenoid, the on-axis magnetic field is absolutely uniform at the value $B_0$ throughout its interior.

Now, let's produce the requested surface plot. We adopt the ranges $0.0 \leq a \leq 2.0$ and $-2.0 \leq z \leq 2.0$, create the necessary arrays of the independent variables with the statement

```matlab
>> ainc=1.0/26.0;
>> zinc=4.0/101.0;
>> [a,z]=meshgrid(0.0:ainc:2.0,-2.0:zinc:2.0);
```

where we pick a “funny” value for ainc and zinc so that we don’t end up evaluating $B$ precisely at $z = \pm 0.5$, thereby avoiding the problem exhibited in the previous paragraph. Then, we calculate $B$ and produce a mesh surface graph, a contour map, and a shaded surface with the statements
Figure E3.24: Magnetic field of solenoid for $a/L = 0.0$.

```matlab
>> B=0.5*(z+0.5)./sqrt(a.^2+(z+0.5).^2) - 0.5*(z-0.5)./sqrt(a.^2 + (z-0.5).^2);
>> mesh(a,z,B, 'EdgeColor','black')
>> [C,H]=contour( a, z, B, 'k');
>> clabel( C, H, 'manual' );
>> colormap( gray )
>> surf( a, z, B )
>> shading interp
```

These three figures are shown in Figs. E3.25, E3.26, and E3.27. The ‘manual’ argument to clabel allows us to choose where to put the contour labels, and the command colormap selects a gray scale (which is better for printing in the solution manual) rather than the default coloring for the shaded surface. You should look at the colored version on the screen of your workstation.
Figure E3.25: Mesh surface plot of $B$ over the $(z/L)$-$(a/L)$ plane.

Figure E3.26: Contour map of $B$ over the $(z/L)$-$(a/L)$ plane.
Figure E3.27: Shaded surface display of $B$ over the $(z/L)-(a/L)$ plane.
3.27 Plotting the On-Axis Field of a Pair of Current Loops

**Exercise:** Consider two circular current loops, each of radius \( a \) and lying with its center on and its plane perpendicular to the \( z \) axis. The first loop is centered at the point \((0,0,b)\) and the second loop is centered at the point \((0,0,-b)\). The axial component of the magnetic field at the point \((0,0,z)\) is given by the equation

\[
B(z) = \frac{1}{2} B_0 \left( \frac{a^2 + b^2}{a^2 + (z+b)^2} \right)^{3/2} + \frac{1}{2} B_0 \left( \frac{a^2 + b^2}{a^2 + (z-b)^2} \right)^{3/2}
\]

where \( B_0 \) is the magnetic field at the origin. Plot graphs showing \( B(z)/B_0 \) as a function of \( z/a \) for various values of \( b/a \), (b) as a surface over the \((z/a)(b/a)\) plane, and (c) as a contour over the \((z/a)(b/a)\) plane. Write a paragraph describing these graphs.

**Solution:** The setup for this problem is shown as Fig. E3.28. The axial component of the magnetic field is represented by

\[
B(z) = \frac{1}{2} B_0 (a^2 + b^2)^{3/2} \left[ \frac{1}{a^2 + (z+b)^2} + \frac{1}{a^2 + (z-b)^2} \right]
\]

To begin, we recast this equation in dimensionless form to eliminate a variable. Start by multiplying the \((z \pm b)^2\) expression by \(a^2/a^2\) and then factoring out a \(1/a^3\) from the expression in the larger braces. This yields

\[
B(z) = \frac{1}{2} B_0 \frac{(a^2 + b^2)^{3/2}}{a^3} \left[ \frac{1}{1 + \left( \frac{z+b}{a} \right)^2} + \frac{1}{1 + \left( \frac{z-b}{a} \right)^2} \right]
\]
Then, multiply the $b^2$ term in the front by $a^2/a^2$ and factor out an $a^3$. Finally, divide both sides by $B_0$ to get the dimensionless equation

$$\frac{B(z)}{B_0} = \frac{1}{2} \left[ 1 + \left( \frac{b}{a} \right)^2 \right]^{3/2} \left[ \frac{1}{1 + \left( \frac{z}{a} + \frac{b}{a} \right)^2} \right]^{3/2} + \left[ \frac{1}{1 + \left( \frac{z}{a} - \frac{b}{a} \right)^2} \right]^{3/2}.$$

Now, we are ready to invoke MATLAB to help us with the problem.

(a) We first draw graphs of $B(z)/B_0$ versus $z/a$ for various values of $b/a$ by using a for loop. Within MATLAB, the statements used to create the graphs shown in Fig. E3.29 are

```matlab
>> zovera = [-4.0 : 0.25 : 4.0]; % Set values for z/a.
>> bovera = [0.25, 0.5, 0.75, 1.0]; % Set values for b/a.
>> for i = 1:4
    Bbar = ...
    (1/2)*(1+(bovera(i)).^2).^(1.5).*...
    (1./(1+(zovera+bovera(i)).^2).^(1.5)... 
    +1./(1+(zovera-bovera(i)).^2).^(1.5));
    subplot(2,2,i); % Select sub window.
    plot( zovera, Bbar, 'Color','black',...
    'LineWidth',2 ); % Plot data.
    axis([-4.0,4,0,2]); % Standardize, label axes.
    xlabel('{\itz}/{\ita}');
    ylabel('{\itB}({\itz})/{\itB_0}');
    title( ['{\itb}/{\ita} = ', num2str(bovera(i),2), ' ', 'FontSize',12 ])
end
```

The dimensionless magnetic field at $z = 0$ will always be 1. (we are evaluating $B$ compared to $B$ at $z = 0$.) Notice the flatness of the peak for $b/a = 0.5$. This is the Helmhotz configuration and is often used in laboratories to create a uniform magnetic field. For values of $b/a < 1.0$ the maximum magnetic field is at the center of the two loops with intensity $B(z)/B_0 = 1$. For larger values of $b/a$ the maximum dimensionless magnetic field can be found in the center of either ring. It appears we could create a very large magnetic field at the center of one of the loops by placing two very small loops far away from one another. However, we must be careful in interpreting the graphs. We have evaluated the dimensionless magnetic field, not the actual magnetic field. We are looking at the ratio of the $B$ field at an arbitrary point on the axis to the $B$ field at the origin. As the separation of the loops increases, the magnetic field at the origin gets smaller. This change more than counteracts the effect seen in the graphs and, as the rings move farther and farther apart, the magnetic field at their respective centers gets smaller and smaller. (Two loops directly on top of one another necessarily create the largest $B$ field.) As we expect, the magnetic field far from the loops approaches zero.

(b) To create a surface showing $B(z)/B_0$ over the $(z/a)(b/a)$ plane, we must form a grid of points for $b/a$ and $z/a$ at which to evaluate the dimensionless magnetic field. From there, the surface is created with the MATLAB `mesh` command. We create the surface with the statements

```matlab
>> [zovera, bovera] = meshgrid([-4.0:0.1:4.0, 0.0:0.2:1.0]);
>> Bbar = (1/2)*(1+(bovera).^2).^(1.5).*...
    (1./(1+(zovera+bovera).^2).^(1.5)... 
    +1./(1+(zovera-bovera).^2).^(1.5));
>> mesh( zovera, bovera, Bbar, 'EdgeColor','k' )
>> xlabel('\{\itz\}/\{\ita\}','FontSize',16);
>> ylabel('\{\itB\}/\{\itB_0\}', 'FontSize',16);
```
The resulting surface (Fig. E3.30) confirms the conclusions drawn from part (a).

(c) Lastly, to make a contour we use MATLAB’s `contour` command and plot the results with the statements

```matlab
>> contour( zovera, bovera, Bbar, 10, 'k' )
>> xlabel('{\itz}/{\ita}','FontSize',16);
>> ylabel('{\itb}/{\ita}','FontSize',16);
>> ylabel('{\itb}/{\ita}','FontSize',16);
```

The resulting graph is shown in Fig. E3.31. The shape of the contour lines gives a good indication of the shape of the peak.
Figure E3.30: Mesh surface display of $B/B_0$ over the $(z/a)(b/a)$ plane.

Figure E3.31: Contour lines showing $B/B_0$ in the $(z/a)(b/a)$ plane.
3.28 Current in an LRC Circuit

**Exercise:** In an LRC circuit of resonant frequency $\omega_0$, the current $I$ is given as a function of frequency $\omega$ by

$$I = \frac{I_0}{\sqrt{1 + Q^2 \left(\frac{\bar{\omega}}{\omega} - 1\right)^2}}$$

where $\bar{\omega} = \omega/\omega_0$. Plot $I = I_0$ (a) as a function of $\bar{\omega}$ for various values of the quality factor $Q$ and (b) as a surface over the $\bar{\omega}Q$-plane. Write a paragraph describing these graphs.

**Solution:** (a) Fig. E3.32 shows the current-versus-frequency plots for values of $Q$ ranging from 1.0 to 15.0. These plots were obtained in MATLAB with the commands,

```matlab
>> w = [0.0:0.05:5.0];
>> Q = [1.0, 3.0, 6.0, 10.0, 15.0];
>> I = 1.0 ./ sqrt( 1 + Q(1)^2.*(w-(1./w)).^2);
>> plot( w, I, 'color','black', 'linewidth',2)
>> title('Current in an LRC circuit', 'fontsize',20)
>> xlabel( '\omega', 'fontsize',16 )
>> ylabel( 'I(\omega) / I_0', 'fontsize',16 )
>> set( gca, 'fontsize',14 )
>> hold on
>> for i = [2:5]
       I = 1.0 ./ sqrt( 1 + Q(i)^2.*(w-(1./w)).^2);
       plot( w, I, 'color','black', 'linewidth',2)
>> end
>> text( 3.0, 0.4, 'Q = 1.0', 'fontsize',14 )
>> text( 2.5, 0.2, 'Q = 3.0', 'fontsize',14 )
>> text( 0.7, 0.05, 'Q = 15.0', 'fontsize',14 )
>> hold off
```

Figure E3.32: Current in an LRC circuit for values of $Q = 1.0, 3.0, 6.0, 10.0, 15.0$. 

![Current in an LRC circuit](image-url)
Exercise 3.28

Figure E3.33: A surface plot over the \( \bar{\omega}Q \) plane.

(b) Taking this a step further, we can plot \( I \) against a continuum of \( \bar{\omega} \)'s and \( Q \)'s. The MATLAB commands,

\[
\begin{align*}
\text{>> } & [w, Q] = \text{meshgrid}(0.0:0.05:5.0, 1.0:2.0:15.0); \\
\text{>> } & I = 1.0 ./ \sqrt{1.0 + Q.^2.*(w-(1./w)).^2}; \\
\text{>> } & \text{mesh( w, Q, I, 'edgecolor', 'black' )} \\
\text{>> } & \text{axis( [0.0,5.0,0.0,16.0] )} \\
\text{>> } & \text{xlabel( '\omega', 'fontsize',16 )} \\
\text{>> } & \text{ylabel( 'Q', 'fontsize',16 )} \\
\text{>> } & \text{zlabel( 'I / I_0', 'fontsize',16 )} \\
\text{>> } & \text{set( gca, 'fontsize',14 )} \\
\text{>> } & \text{view(60,40)}
\end{align*}
\]

were used to create the graph shown in Fig. E3.33. The `view` command is used to display the surface at a more optimal viewing angle. Note that it took a bit of experimentation to determine suitable azimuth and elevation.

The most interesting aspect of these plots is the peak at \( \bar{\omega} = 1 \), when \( \omega = \omega_0 \). At higher \( Q \) values the peak is much sharper (hence a higher “quality factor”: the circuit is only conductive at a very specific frequency, namely the resonant frequency). Notice also that the high-end roll-off is much more gradual than the low-end roll-off, which you may remember from some electronics labs.
3.29 Cooking a Spherical Potato

**Exercise:** A spherical potato of radius $a$ is taken from the refrigerator at $0 \, ^\circ C$ and placed in an oven at $u_0 = 200 \, ^\circ C$. The temperature $u(r, t)$ at a point a distance $r$ from the center of the potato at time $t$ is given by

$$
\frac{u(r, t)}{u_0} = 1 - 2 \sum_{n=1}^{\infty} \frac{J_0(\beta_n r/a)}{\beta_n J_1(\beta_n)} e^{-\kappa \beta_n^2 t/(ca^2)}
$$

where $\kappa$ is the thermal conductivity of the potato, $c$ is its heat capacity per unit volume, $J_0(x)$ and $J_1(x)$ are the zeroth- and first-order spherical Bessel functions, and $\beta_n$ is the $n$-th root of $J_0(x)$, i.e., $J_0(\beta_n) = 0$. Obtain graphs of $u(r, t)/u_0$ as a function of $r/a$ for various values of $t$. Obtain also a graph of the temperature $u(0, t)$ at the center of the potato as a function of $t$ and determine how long it takes the potato to bake if, by being baked, one means that the temperature at the center has risen to $175 \, ^\circ C$, i.e., to a value such that $u(0, t)/u_0 = 0.875$. **Hints:** (1) Note that

$$
J_0(x) = \frac{\sin x}{x} \quad \text{and} \quad J_1(x) = \frac{\sin x}{x^2} - \frac{\cos x}{x}
$$

Thus, the $n$-th root of $J_0(x)$ is $\beta_n = n\pi$. (2) Express times in units of $ca^2/\kappa$ but then, taking the radius of the potato to be $a = 0.05 \, m$ and taking $\kappa$ and $c$ for the potato to be those of water [$\kappa = 0.63 \, J/(m \, s \, K)$, and $c = 4.2 \times 10^6 \, J/(K \, m^3)$], determine the unit in which your answers are expressed, both in seconds and in hours. (3) Experiment a bit, but note that the exponential factor decays more rapidly as $n$ increases, so truncation of the infinite series at some point is probably in order.

**Solution:** The temperature of a spherical potato which started at $0 \, ^\circ C$ and was placed in an oven at $u_0 = 200 \, ^\circ C$ at a distance $r$ from the center of the potato is given by

$$
\frac{u(r, t)}{u_0} = 1 - 2 \sum_{n=1}^{\infty} \frac{J_0(\beta_n r/a)}{\beta_n J_1(\beta_n)} e^{-\kappa \beta_n^2 t/(ca^2)}
$$

where $\kappa$ is the thermal conductivity of the potato, $c$ its heat capacity, $J_0(x)$ and $J_1(x)$ the spherical Bessel functions, and $\beta_n$ the $n$-th root of $J_0(x)$.

If we express time in units of $ca^2/\kappa$ and distance in units of $a$ and substitute $n\pi$ for $\beta_n$, we find that the equation for the temperature becomes

$$
\frac{u(\bar{r}, \bar{t})}{u_0} = 1 - 2 \sum_{n=1}^{\infty} (-1)^{n+1} \frac{\sin(n \pi \bar{r})}{n \pi \bar{r}} e^{-n^2 \pi^2 \bar{t}}
$$

where $\bar{r} = r/a$ and $\bar{t} = t/(ca^2/\kappa)$. Unfortunately, at the special point $r = 0$, the summand in this expression becomes indeterminate, which will cause a problem for the numerical evaluation (even though the limit of the expression as $r \to 0$ is perfectly reasonable). Since

$$
\lim_{r \to 0} \frac{\sin(n \pi \bar{r})}{n \pi \bar{r}} = 1
$$

we can deduce the simpler expression

$$
\frac{u(0, \bar{t})}{u_0} = 1 - 2 \sum_{n=1}^{\infty} (-1)^{n+1} e^{-n^2 \pi^2 \bar{t}}
$$

for the temperature at $r = 0$; we shall use this expression to deal with that special point.

Now, dropping the bars in the coding, we can evaluate this expression in MATLAB as follows. First, we invoke the statements
to create values of \( r \) that avoid the point \( r = 0 \), define the vector \textit{including} the zero value for plotting purposes, and establish a value for \( t \). Then, for \textit{each} of 100 values of \( r \) (index \( i \), we sum (index \( n \)) over however many values of \( n \) we choose to include in the sum. If we truncate the sum with \( n = 20 \), we would then evaluate \( u(r) \), add the temperature at \( r = 0 \), and save the results in a unique variable with the statements

\[
\begin{align*}
\text{>> for } i &= 1:100 \\
& \quad \text{for } n = 1:20 \\
& \quad \quad \text{tmp} = (-1) \cdot ((n+1) \cdot \sin(n \cdot \pi \cdot r(i)) \cdot \exp(-n \cdot 2 \cdot \pi \cdot 2 \cdot t)/(n \cdot \pi \cdot r(i))) \\
& \quad \quad u(i) = u(i) - 2.0 \cdot \text{tmp} \\
& \quad \text{end} \\
& \text{end} \\
\text{>> u0} = 1.0; \\
\text{>> for } n = 1:20 \ u0 = u0 - 2.0 \cdot ((-1) \cdot ((n+1) \cdot \exp(-n \cdot 2 \cdot \pi \cdot 2 \cdot t)); \text{end}; \\
\text{>> u01} = [ u0, u ];
\end{align*}
\]

Similar statements (not here recorded) will produce arrays \( u05 \), \( u10 \), \( u15 \), \( u20 \), and \( u25 \) containing the radial temperature distribution at \( t = 0.05, 0.10, 0.15, 0.20, \) and \( 0.25 \).³

With those quantities calculated, we can then plot a composite graph and label each time with the statements

\[
\begin{align*}
\text{>> plot( rplot, u01, 'Color', 'black', 'LineWidth', 3 );} \\
\text{>> xlabel('r/a', 'FontSize',16 );} \\
\text{>> ylabel('u/u0', 'FontSize', 16 );} \\
\text{>> hold on} \\
\text{>> plot( rplot, u05, 'Color', 'black', 'LineWidth', 3 );} \\
\text{>> plot( rplot, u10, 'Color', 'black', 'LineWidth', 3 );} \\
\text{>> plot( rplot, u15, 'Color', 'black', 'LineWidth', 3 );} \\
\text{>> plot( rplot, u20, 'Color', 'black', 'LineWidth', 3 );} \\
\text{>> plot( rplot, u25, 'Color', 'black', 'LineWidth', 3 );} \\
\text{>> text( 0.8, 0.1, '0.01' )} \\
\text{>> text( 0.5, 0.15, '0.05' )} \\
\text{>> text( 0.3, 0.3, '0.10' )} \\
\text{>> text( 0.2, 0.5, '0.15' )} \\
\text{>> text( 0.15, 0.68, '0.20' )} \\
\text{>> text( 0.1, 0.78, '0.25' )} \\
\text{>> hold off}
\end{align*}
\]

The graph resulting from the above operations is shown in Fig. E3.34. You can see how the potato heats up from the outside in as time passes.

The exercise also asks for a graph of the temperature as a function of time at the center of the potato. We evaluate the points for plotting that graph as follows. Given the results in the above, we infer that examining times over the interval \( 0 \leq t \leq 0.5 \) would be appropriate, though we must avoid \( t = 0 \) (because the sum doesn’t converge at that time). We set the time scale with the statements

³We can safely ignore occasional messages informing us of floating point underflow. Negative exponentials go quickly to zero and, at some points in these evaluations, the exponential assumes a value smaller than the smallest value that can be represented in the computer. Despite the message, MATLAB actually treats the value as zero, which is quite correct.
Exercise 3.29

Figure E3.34: The temperature of the potato at several times. Remember that time is in units of $ca^2/\kappa$ (approximately 4 hrs 38 mins), distance is in units of $a$ (the radius of the potato), and temperature is in units of $u_0$ ($200^\circ$C).

$$t = [0.005 : 0.005 : 0.5 ];$$

Then, for each time (index $i$), we again take 20 terms in the sum (index $n$), we evaluate the temperatures and plot the desired graph with the statements

```matlab
>> for i = 1:100
    u(i) = 1.0;
    for n = 1:20
        u(i) = u(i) - 2.0*(-1)^(n+1)*exp(-n^2*pi^2*t(i));
    end
end
>> plot( [0.0, t] , [0.0, u], 'Color', 'black', 'LineWidth', 3 );
>> grid on
>> xlabel( 't/(ca^2\kappa)' )
>> ylabel( 'u(0)/u_0' )
```

Finally, to locate the point at which the temperature at the center has reached $0.875u_0$, we draw a (dashed) horizontal line at $u(0)/u_0 = 0.875$ and, with a bit of trial and error, discover that a vertical line at $t/(ca^2/\kappa) = 0.28$ comes close to intersecting the solid curve at the point $u(0)/u_0 = 0.875$. The statements adding these two lines are

```matlab
>> hold on
>> plot( [0.0,0.5], [0.875, 0.875], '--' )
>> plot( [0.28, 0.28], [0.0, 1.0], '--' )
>> hold off
```

The final plot is shown in Fig. E3.35.
To produce a more refined estimate of the critical time, we could adjust the previous graph to show a smaller portion of the interval with the statements

```matlab
>> axis([0.27,0.29, 0.86,0.9])
>> plot([0.281, 0.281], [0.86,0.90], '--')
```

from which graph—shown in Fig. E3.36—we conclude that $t = 0.281$ is a better estimate of the critical time.

We see from this final graph that the center of the potato has reached 85% of $u_0$ at $t = .281$. Remembering that $t$ is in units of $ca^2/\kappa$ and assuming that the values of $c$ and $\kappa$ are about the same as those for water, ($\kappa = 0.63 \text{ J/(s m K)}$ and $c = 4.2 \times 10^6 \text{ J/(K m^3)}$), we find that $t$ in more conventional time units is about 1 hour and 18 minutes, so this potato takes a very long time to cook. The cooking time would be shorter if we set the oven temperature higher, so we would not have to wait until the temperature at the center had come so close to equilibrium.
Figure E3.36: A magnified view of the temperature at the center near the critical cooking time.
3.30 Gravitational Potential Energy in a Plane

Exercise: At a particular time, a planet of mass $M$ is located at the origin in the $xy$ plane and a moon of mass $M/3$ is located at a point a distance $R$ from the planet on the $x$ axis. The gravitational potential energy of a spaceship of mass $m$ at the point $(x, y, z)$ is then given by

$$V(x, y, z) = -\frac{GmM}{\sqrt{x^2 + y^2 + z^2}} - \frac{GmM/3}{\sqrt{(x-R)^2 + y^2 + z^2}}$$

Using MATLAB, obtain surface plots and contour maps of this potential energy in the $xy$ plane (i.e., the plane $z = 0$) and in the planes $z = 0.1R$ and $z = 0.5R$. Suggestion: Recast the function in dimensionless form by measuring $x$, $y$, and $z$ in units of $R$ and $V(x, y, z)$ in units of $GmM/R$.

Solution: If we introduce the variables $\bar{x} = x/R$, $\bar{y} = y/R$, and $\bar{z} = z/R$, or $x = R\bar{x}$, $y = R\bar{y}$, $z = R\bar{z}$, we can recast the potential energy of the spaceship as

$$V(x, y, z) = -\frac{GmM}{\sqrt{x^2 + y^2 + z^2}} - \frac{GmM/3}{\sqrt{(x-R)^2 + y^2 + z^2}}$$

$$\Rightarrow V = \frac{V}{GmM/R} = \frac{1}{\sqrt{\bar{x}^2 + \bar{y}^2 + \bar{z}^2}} - \frac{1/3}{\sqrt{(\bar{x} - 1)^2 + \bar{y}^2 + \bar{z}^2}}$$

To generate plots of $V$ as a function of $x$ and $y$ (we drop the bars from here on) for various values of $z$, we choose—at least as a first pass—to plot over the interval $-2.0 \leq x, y \leq 2.0$, and we begin by creating a grid of values of $x$ and $y$ with the MATLAB statement

$$[x, y] = \text{meshgrid}(-2.0:0.16:2.0, -2.0:0.16:2.0);$$

Then, we evaluate the function with the statements

$$z=0.0; \quad \text{den1 = sqrt(x.^2+y.^2+z.^2); den2 = sqrt((x-1).^2+y.^2+z.^2);} \quad \text{V = - 1./den1 - 0.3333333./den2;}$$

Finally, we request some guidance on the range of potential energies that appear, generate a surface plot of the potential energy in the plane $z = 0.0$, set the values at which equipotential contours should be drawn, set a vector that will specify the labeling of contours,

$$[\text{max(max(V))}, \text{min(min(V))}] \quad \text{ans = -0.4460 -9.1999} \quad \text{mesh(x, y, V, 'edgecolor', 'black')} \quad \text{set(gca, 'fontsize', 14)} \quad \text{title('z = 0.0', 'fontsize', 16)}$$

$$\text{lvls = [-9.0, -6.0, -4.0, -3.0, -2.0, -1.5, -1.0, -0.75, -0.5];} \quad \text{contour(x, y, V, lvls, 'k', 'linewidth', 2)} \quad \text{set(gca, 'fontsize', 14)} \quad \text{title('z = 0.0', 'fontsize', 16)}$$

\footnote{Note that the choice of the increment 0.16 divides the intervals with points that do not include evaluation of the potential energy either at the point $(x, y) = (0, 0)$ or the point $(x, y) = (1, 0)$ so we will not be troubled by the divergences of the potential energy at those points.}
Some trial and error was involved in determining appropriate values at which to draw the contour lines. The resulting graphs are shown in Fig. E3.37 and Fig. E3.38.

Producing graphs for other values of \( z \) involves a similar process. For \( z/R = 0.1 \), we would invoke the statements

\[
\begin{align*}
\text{>> } & z=0.1; \\
\text{>> } & \text{den1 = sqrt(x.^2+y.^2+z^2); den2 = sqrt((x-1).^2+y.^2+z^2);} \\
\text{>> } & \text{V = - 1/den1 - 0.3333333/den2;} \\
\text{>> } & \text{[ max(max(V)), min(min(V)) ]} \\
\text{ans} = & \text{ -0.4457 -6.9815} \\
\text{>> } & \text{mesh( x, y, V, 'edgecolor', 'black' )} \\
\text{>> } & \text{set( gca, 'fontsize', 14 )} \\
\text{>> } & \text{title('z = 0.1', 'fontsize', 16 )} \\
\text{>> } & \text{lvls = [-6.5, -4.5, -3.5, -2.5, -1.5, -1.0, -0.75, -0.5];} \\
\text{>> } & \text{contour( x, y, V, lvls, 'k', 'linewidth', 2 )} \\
\text{>> } & \text{set( gca, 'fontsize', 14 )} \\
\text{>> } & \text{title('z = 0.1', 'fontsize', 16)} \\
\end{align*}
\]

(We need not reinvoke \texttt{meshgrid}, since its output has not been destroyed.) The resulting graphs are shown in Fig. E3.39 and Fig. E3.40.

Finally, for \( z/R = 0.5 \), we would invoke the statements

\[
\begin{align*}
\text{>> } & z=0.5; \\
\text{>> } & \text{den1 = sqrt(x.^2+y.^2+z^2); den2 = sqrt((x-1).^2+y.^2+z^2);} \\
\text{>> } & \text{V = - 1/den1 - 0.3333333/den2;} \\
\text{>> } & \text{[ max(max(V)), min(min(V)) ]} \\
\text{ans} = & \text{ -0.4397 -2.2681} \\
\text{>> } & \text{mesh( x, y, V, 'edgecolor', 'black' )} \\
\text{>> } & \text{set( gca, 'fontsize', 14 )} \\
\text{>> } & \text{title('z = 0.5', 'fontsize', 16 )} \\
\text{>> } & \text{lvls = [-2.2, -1.8, -1.4, -1.0, -0.8, -0.6, -0.5, -0.45];} \\
\text{>> } & \text{contour( x, y, V, lvls, 'k', 'linewidth', 2 )} \\
\text{>> } & \text{set( gca, 'fontsize', 14 )} \\
\text{>> } & \text{title('z = 0.5', 'fontsize', 16)} \\
\end{align*}
\]

The resulting graphs are shown in Fig. E3.41 and Fig. E3.42.
Figure E3.37: Surface plot of potential energy in the plane $z = 0.0$.

Figure E3.38: Contour map of potential energy in the plane $z = 0$.

Figure E3.39: Surface plot of potential energy in the plane $z = 0.1R$.

Figure E3.40: Contour map of potential energy in the plane $z = 0.1R$.

Figure E3.41: Surface plot of potential energy in the plane $z = 0.5R$.

Figure E3.42: Contour map of potential energy in the plane $z = 0.5R$. 
3.31 Plotting Probability Densities in Hydrogen

**Exercise:** Following the pattern illustrated in Section 3.14, explore at least one of the three-dimensional scalar fields

\[ p_{3,1,0}(x, y, z) = \frac{8}{(27)^2 \pi} \rho^2 \left( 1 - \frac{\rho}{6} \right)^2 e^{-2\rho/3} \cos^2 \theta \]

\[ p_{3,1,1}(x, y, z) = \frac{4}{(27)^2 \pi} \rho^2 \left( 1 - \frac{\rho}{6} \right)^2 e^{-2\rho/3} \left( 1 - \cos^2 \theta \right) \]

\[ p_{3,2,1}(x, y, z) = \frac{3}{(27)^3 \pi} \rho^4 e^{-2\rho/3} \cos^2 \theta (1 - \cos^2 \theta) \]

\[ p_{3,2,2}(x, y, z) = \frac{3}{4(27)^3 \pi} \rho^4 e^{-2\rho/3} (1 - \cos^2 \theta)^2 \]

giving the probability density for the hydrogen states \((n, l, m) = (3, 1, 0), (n, l, m) = (3, 1, 1), (n, l, m) = (3, 2, 1),\) and \((n, l, m) = (3, 2, 2).\) These fields are expressed in dimensionless form, where \(\rho\) is the radial coordinate in units of the Bohr radius. In terms of the Cartesian coordinates \(x, y, z,\) \(\rho = \sqrt{x^2 + y^2 + z^2}\) and \(\cos \theta = z/\rho.\) **Hint:** To avoid divisions by zero, recast the expressions in terms of \((x, y, z)\) explicitly before evaluating any of them numerically.

**Solution:** To analyze the probability density for various hydrogen states, we might invoke a variety of techniques. We could fix one variable and make mesh surfaces or contour plots showing the function as a function of the other two variables for that fixed value of the chosen variable. We could also look at more general slices taken with MATLAB’s `slice` function or examine isosurfaces drawn in the three-dimensional space of the independent variables. Here, we opt for the last method for a compact representation of the probability densities. Realize that there are, of course, many ways that these probability densities might be represented; this approach is merely a small sample of how one might look at these densities. To avoid problems with division by zero, we elect first to recognize that, if evaluated as \(z^2/\rho^2,\) the factors of \(\cos^2 \theta\) appearing in the several probability densities will generate numerical difficulties when \(\rho = 0.\) We can avoid these problems by noting that

\[ \rho^2 \cos^2 \theta = \frac{\rho^2 z^2}{\rho^2} = z^2 \quad ; \quad \rho^2 (1 - \cos^2 \theta) = \rho^2 \left( 1 - \frac{z^2}{\rho^2} \right) = x^2 + y^2 \]

Thus, we find that

\[ p_{3,1,0} = \frac{8}{(27)^2 \pi} z^2 \left( 1 - \frac{\rho}{6} \right)^2 e^{-2\rho/3} \]

\[ p_{3,1,1} = \frac{4}{(27)^2 \pi} (x^2 + y^2) \left( 1 - \frac{\rho}{6} \right)^2 e^{-2\rho/3} \]

\[ p_{3,2,1} = \frac{3}{(27)^3 \pi} (x^2 + y^2) z^2 e^{-2\rho/3} \]

\[ P_{3,2,2} = \frac{3}{4(27)^3 \pi} (x^2 + y^2)^2 e^{-2\rho/3} \]

With these recastings, we begin by creating the three-dimensional grid containing values ranging from \(-15 \leq x, y, z \leq 15\) and then evaluating the four probability densities with the statements

\[ \text{inc}=30.0/50.0; \]
\[ [x,y,z]=\text{meshgrid}(-15.0:\text{inc}:15.0, -15.0:\text{inc}:15.0, -15.0:\text{inc}:15.0); \]
\[ \text{rhos}=x.\text{^}2+y.\text{^}2+z.\text{^}2; \]
\[ \text{rho}=	ext{sqrt}(	ext{rhos}); \]
\[ \text{xspys} = x.\text{^}2 + y.\text{^}2; \]
Exercise 3.31

>> zs = z.^2;
>> fct = (1 - rho/6.0).^2;
>> expf = exp(-2.0*rho/3.0);
>> p310 = (8.0/(27.0^2*pi))*zs.*fct.*expf;
>> p311 = (4.0/(27.0^2*pi))*xspys.*fct.*expf;
>> p321 = (3.0/(27.0^3*pi))*xspys.*zs.*expf;
>> p322 = (3.0/(4.0*27.0^3*pi))*xspys.^2.*expf;

To guide our selection of values, we find out the maximum probability in each array with the statement

>> [max(max(max(p310))), max(max(max(p311))), ...
   max(max(max(p321))), max(max(max(p322)))]
ans = 0.0017 0.0008 0.0003 0.0003

While these results are more dramatic when displayed in color on a computer screen, we adopt the gray-scale colormap for the sake of printing. Following the pattern illustrated in the text, we next use the routines isosurface and patch by executing the statements

>> colormap gray;
>> poly310 = isosurface( x, y, z, p310, 1.4e-4);
>> hpatch = patch( poly310 );
>> axis([-15.0,15.0, -15.0,15.0, -15.0,15.0] );
>> camlight left; camlight right; lighting phong
>> grid on

Here, we have found the points at which the probability density is \(1.4 \times 10^{-4}\) and displayed the resulting image. The resulting isosurface is shown in Figure E3.43.

We can, of course, also produce similar surfaces for the other three states. In each case, we show only one isosurface. Others can readily be drawn. There is no substitute for exploring these functions interactively at the computer keyboard. To generate a representative isosurface for the state \(3,1,1\), we might execute the statements

>> clf
>> poly311 = isosurface( x, y, z, p311, 0.75e-4);
>> hpatch = patch( poly311 );
>> axis([-15.0,15.0, -15.0,15.0, -15.0,15.0] );
>> camlight left; camlight right; lighting phong
>> grid on

The result is shown in Fig. E3.44.

Similarly, for the states \((3,2,1)\) and \((3,2,2)\), we would execute the statements

>> clf
>> poly321 = isosurface( x, y, z, p321, 0.5e-4);
>> hpatch = patch( poly321 );
>> axis([-15.0,15.0, -15.0,15.0, -15.0,15.0] );
>> camlight left; camlight right; lighting phong
>> grid on
>> clf

5 Omitting the statement colormap gray will select the default color map and produce a more vivid display on the screen.
Exercise 3.31

Figure E3.43: An isosurface across the probability density of $1.4 \times 10^{-4}$ for the $p_{3,1,0}$ state.

$$\text{>> poly322 = isosurface( x, y, z, p322, 0.3e-4);}$$
$$\text{>> hpatch = patch( poly322 );}$$
$$\text{>> axis( [-15.0,15.0, -15.0,15.0, -15.0,15.0] );}$$
$$\text{>> camlight left; camlight right; lighting phong}$$
$$\text{>> grid on}$$

The resulting isosurfaces are shown in Fig. E3.45 and Fig. E3.46.

Since all of these probability distributions are invariant to rotation about the $z$ axis, we might also find a contour map in a plane containing the $z$ axis to be useful. Following the pattern in the text, we would construct such a map for the state $(3,1,0)$ with the statements\(^6\)

$$\text{>> for i = 1:51}$$
$$\text{\hspace{1em} for j = 1:51}$$
$$\text{\hspace{2em} p310c(i,j) = p310(25,j,i);}$$
$$\text{\hspace{1em} end}$$
$$\text{\hspace{1em} end}$$
$$\text{>> sc = [ -15.0 : inc : 15.0 ];}$$
$$\text{>> subplot( 2, 2, 1 )}$$
$$\text{>> contour( sc, sc, p310c, 30, 'k' )}$$
$$\text{>> axis square}$$
$$\text{>> title( '(3,1,0)' )}$$

Here, the $x$ index is set to 25 so as to pick the $yz$ plane. To produce displays for $p_{3,1,1}$, $p_{3,2,1}$, and $p_{3,2,2}$, we would use the similar statements

$$\text{>> for i = 1:51}$$
$$\text{\hspace{1em} for j = 1:51}$$

\(^6\)The indices are reversed in the interior of the double loop so that, in the end, the $z$ axis will turn out to be the vertical axis in the figure.
Figure E3.44: An isosurface across the probability density of $0.75 \times 10^{-4}$ for the $p_{3,1,1}$ state.

The results are shown in Fig. E3.47. In all four cases, the isosurfaces explored earlier in this solution can be obtained by rotating the contour map about a vertical line through the center of the map.
Figure E3.45: An isosurface across the probability density of $0.5 \times 10^{-4}$ for the $p_{3,2,1}$ state.

Figure E3.46: An isosurface across the probability density of $0.3 \times 10^{-4}$ for the $p_{3,2,2}$ state.
Figure E3.47: Contour maps for the four states explored in this exercise.
3.32 Standing Waves in a Cube (2001 Version)

Exercise: The (gauge) pressure \( p(x, y, z, t) \) inside a cubical box located in the region \( 0 \leq x, y, z \leq a \) is given by

\[
p(x, y, z, t) = A \sin \frac{l \pi x}{a} \sin \frac{m \pi y}{a} \sin \frac{n \pi z}{a} \cos \omega_{lmn} t
\]

where \( l, m, \) and \( n \) are positive integers. Obtain several presentations of the pressure distribution inside this box at \( t = 0 \) and at \( t = \pi/\omega_{lmn} \) for several different values of \( l, m, \) and \( n \).

Solution: The pressure inside a cubical box of side \( a \) at some time \( t \) can be recast in dimensionless form as

\[
\frac{p(x, y, z, t)}{A} = \sin \frac{l \pi x}{a} \sin \frac{m \pi y}{a} \sin \frac{n \pi z}{a} \cos \omega_{lmn} t
\]

(E3.32.1)

Knowing that \( 0 \leq x, y, z \leq a \) we can look at the equation as a function of the dimensionless variables \( x/a, y/a, \) and \( z/a \) running from 0 to 1. In the examination of this equation we will look at slices taken on each of the \( xy, xz, \) and \( yz \) planes as well as isosurfaces to get a feel for what is happening inside the box. First, we need to create a three-dimensional matrix over which the function can be evaluated. Recognize that at \( t = 0 \) the cosine term will be 1 and at \( t = \pi/\omega_{lmn} \) the cosine term will be \(-1\). We need to define and evaluate our equation for various values of \( l, m, \) and \( n \). Looking at the equation we realize that a state of \( l = 1, m = 1, n = 2 \) and a state of \( l = 2, m = 1, \) and \( n = 1 \) will be mere rotations of the same figure. Thus, if we allow \( l, m, \) and \( n \) to range from 1 to 3 there are really only 10 unique states with the others being rotations of one already created. Cross sections and isosurfaces of the first three states can be created with the commands shown in Table 3.1.

The resulting plots are contained in Figure E3.48. For the next three states of \( l, m, \) and \( n, \) use the same commands with the exception of \( l \) and \( m \) being assigned a value of 2 rather than 1. This produces the plots in Figure E3.49. Then, assigning \( l \) and \( m \) to 3 and reexecuting the same commands we produce Figure E3.50. The final figure for the \( l = 1, m = 2, n = 3 \) state at \( t = 0, \) is

---

Figure E3.48: Cross-sectional surfaces and isosurfaces of the pressure inside a cubical box at \( t = 0 \) for \( l = 1, m = 1, \) and \( n = 1 \) (column 1), \( n = 2 \) (column 2), and \( n = 3 \) (column 3). Isosurfaces are drawn at \( p/A = 0.5. \)
Table 3.1: Statements for standing waves.

```
>> inc = 0.025;
>> [xovera,yovera,zovera]=meshgrid(0:inc:1,0:inc:1,0:inc:1);
>> cosfactor = 1;
>> l = 1;
>> m = 1;
>> part1 = sin(l*pi.*xovera);
>> part2 = sin(m*pi.*yovera);
>> pbarpart = part1.*part2.*cosfactor;
>> for n = 1:3;
    pbar = sin(n*pi.*zovera).*pbarpart;
    subplot(2,3,n);
    slice(xovera,yovera,zovera,...
        pbar,0.5,0.5,0.5);
    title('Cross Section');
    xlabel('{\it x}/{\it a}');
    ylabel('{\it y}/{\it a}');
    set(h,'FaceColor','red','EdgeColor','none');
    camlight left;
    camlight right;
end
```

Create a step size of 0.25.

Create a 3-D matrix of values for \(x/a, y/a,\) and \(z/a\) all ranging from 0 to 1.

Evaluate at \(t = 0.\)

Set \(l = 1.\)

Set \(m = 1.\)

Evaluate Parts of the equation.

Begin for loop for \(n.\)

Put the parts together and evaluate for the dimensionless pressure.

Setup plotting format for six graphs.

Take 2-D slices in the \(xy, xz,\) and \(yz\) planes at 0.5 (center of box).

Title plot.

Label \(x.\)

Label \(y.\)

Label \(z.\)

Add colorbar for scale.

Use gray colormap.

Make axes square.

Shading plots.

Plot isosurfaces below cross sections.

Find isosurface for a value of 0.5.

End loop.

created by eliminating the for loop and changing the subplot routine to contain two figures rather than six. This figure is represented as Figure E3.51.

To create the same representations at \(t = \pi/\omega_{lmn},\) we need only change the cosfactor to be \(-1\) and reevaluate the same commands. This produces Figures E3.52, E3.53, E3.54, and E3.55.

Going a step further, we can get a good idea of the isosurfaces by creating an animation of how the pressure changes over time. To facilitate producing this display, we write a function file to look at an isosurface from \(t = 0\) to \(t = \pi/\omega_{lmn}\) with parameters \(L, M, N, ISO,\) and \(FRAMES.\) This file is named pmovie.m and contains the statements

```
function fct = pmovie(L, M, N, ISO, FRAMES)
% PMOVIE - creates a movie showing how the pressure changes
% within a cubical box over time. The parameters L, M,
% and N are positive integers representing the state that
```
Figure E3.49: Cross-sectional surfaces and isosurfaces of the pressure inside a cubical box at $t = 0$ for $l = 2$, $m = 2$, and $n = 1$ (column 1), $n = 2$ (column 2), and $n = 3$ (column 3). Isosurface drawn at $p/A = 0.5$.

Figure E3.50: Cross-sectional surfaces and isosurfaces of the pressure inside a cubical box at $t = 0$ for $l = 3$, $m = 3$, and $n = 1$ (column 1), $n = 2$ (column 2), and $n = 3$ (column 3). Isosurface drawn at $p/A = 0.5$. 
Figure E3.51: Cross-sectional surface and isosurface of the pressure inside a cubical box at $t = 0$ for $l = 1$, $m = 2$, and $n = 3$ state. Isosurface drawn at $p/A = 0.5$.

Figure E3.52: Cross-sectional surfaces and isosurfaces of the pressure inside a cubical box at $t = \frac{\pi}{\omega_{lmn}}$ for $l = 1$, $m = 1$, and $n = 1$ (column 1), $n = 2$ (column 2), and $n = 3$ (column 3). Isosurface drawn at $p/A = 0.5$. The isosurface in the first column is absent because, for the $(1, 1, 1)$ mode at this time, the pressure in the box is everywhere negative.
Figure E3.53: Cross-sectional surfaces and isosurfaces of the pressure inside a cubical box at \( t = \pi / \omega_{lmn} \) for \( l = 2, m = 2, \) and \( n = 1 \) (column 1), \( n = 2 \) (column 2), and \( n = 3 \) (column 3). Isosurface drawn at \( p/A = 0.5 \).

Figure E3.54: Cross-sectional surfaces and isosurfaces of the pressure inside a cubical box at \( t = \pi / \omega_{lmn} \) for \( l = 3, m = 3, \) and \( n = 1 \) (column 1), \( n = 2 \) (column 2), and \( n = 3 \) (column 3). Isosurface drawn at \( p/A = 0.5 \).
Figure E3.55: Cross-sectional surface and isosurface of the pressure inside a cubical box at $t = \pi/\omega_{lmn}$ for $l = 1$, $m = 2$, and $n = 3$ state. Isosurface drawn at $p/A = 0.5$.

% the box is in. The ISO parameter represents what isosurface you want to look at and the FRAMES parameter represents how many frames will be produced.
inc = 0.025;
[xovera,yovera,zovera] = meshgrid(0:inc:1,0:inc:1,0:inc:1); % Creates matrix
part1 = sin(L*pi.*xovera); % Evaluates parts
part2 = sin(M*pi.*yovera);
part3 = sin(N*pi.*zovera);
bigpart = part1.*part2.*part3 % Puts parts together
for i = 1:FRAMES % Begins for loop
    cosfactor = i*2.0/FRAMES-1; % Makes time vector
    pbar = bigpart.*cosfactor; % Evaluates pbar.
poly = isosurface(xovera,yovera,zovera,pbar,ISO); % Makes isosurface
    h = patch(poly); % Displays isosurface
    axis([0,1,0,1,0,1]); % Uniform axes
    set(h,'FaceColor','red','EdgeColor','none'); % Red surface
    camlight left;
    A(i) = getframe; % Get a frame
    clf; % Clear figure
end % End loop
movie(A) % Run movie

Then, within MATLAB, we can execute the single command

```
>> pmovie(l, m, n, iso, frames)
```

to run the movie. (It may take a couple minutes before the movie actually starts.) A similar file could be created to look at how the slices in the figures change over time.
3.32 Standing waves in a cube (2002 Version)

Exercise: The (gauge) pressure \( p(x, y, z, t) \) inside a cubical box located in the region \( 0 \leq x, y, z \leq a \) is given by

\[
p(x, y, z, t) = A \sin \frac{l \pi x}{a} \sin \frac{m \pi y}{a} \sin \frac{n \pi z}{a} \cos \omega_{lmn} t
\]

where \( l, m, \) and \( n \) are positive integers. Obtain several presentations of the pressure distribution inside this box at \( t = 0 \) and at \( t = \pi/\omega_{lmn} \) for several different values of \( l, m, \) and \( n \).

Solution: The (gauge) pressure inside a cubical box of side \( a \) at some time \( t \) can be recast in dimensionless form as

\[
\bar{p}(x, y, z, t) = \sin(l \pi \bar{x}) \sin(m \pi \bar{y}) \sin(n \pi \bar{z}) \cos(\omega_{lmn} t)
\]

where \( l, m, \) and \( n \) are positive integers and \( \bar{p} = p/A, \bar{x} = x/a, \bar{y} = y/a, \) and \( \bar{z} = z/a \). At \( t = 0 \) this becomes

\[
\bar{p}(x, y, z, 0) = \sin(l \pi \bar{x}) \sin(m \pi \bar{y}) \sin(n \pi \bar{z})
\]

Similarly, at \( t = \pi/\omega_{lmn} \) it becomes

\[
\bar{p}(x, y, z, \pi/\omega_{lmn}) = -\sin(l \pi \bar{x}) \sin(m \pi \bar{y}) \sin(n \pi \bar{z})
\]

In these expressions, the independent variables range over the values \( 0 \leq \bar{x}, \bar{y}, \bar{z} \leq 1 \) and \( -1 \leq \bar{p} \leq 1 \). In the examination of this equation we will look at slices taken on each of the \( xy, xz, \) and \( yz \) planes as well as isosurfaces to reveal more completely what is happening inside the box. First, we need to create a three-dimensional matrix over which the function can be evaluated. Looking at the equation, we realize that a state of \( l = 1, m = 1, n = 2 \) and a state of \( l = 2, m = 1, n = 1 \) will be mere rotations of the same figure. Thus, if we allow \( l, m, \) and \( n \) to range from 1 to 3 there are really only 10 unique states with the others being rotations of one already created. We shall confine our explorations to the states \((l, m, n) = (1, 1, 1), (1, 3, 2), \) and \((2, 3, 2)\).

We begin by creating the grid of points over which the pressure will be evaluated for each of the states we examine. To do so, we invoke the statement

\[
\text{>> inc=1.0/50.0;}
\]
\[
\text{>> [x,y,z]=meshgrid(0.0:inc:1.0,0.0:inc:1.0,0.0:inc:1.0);} \]

where we have decided to divide each of the three coordinate axes into 50 segments, yielding arrays, each of which has 51\(^3\) = 132,651 elements and requiring (at double precision; eight bytes per value) 1.6MB of storage. (The choice of this size reflects a balance between use of memory and computation times on the one hand and smoothness of displays on the other hand.)

To enter the pressure for \( l = m = n = 1 \) and \( t = 0 \) and \( t = \pi/\omega_{111} \) into IDL, we execute the statements\(^7\)

\[
\text{>> p1110=sin(pi*x).*sin(pi*y).*sin(pi*z);} \]

Note, incidentally, that the pressure in this mode is everywhere positive at \( t = 0 \), i.e., the pressure is everywhere above the uniform pressure that would exist everywhere in the cube in the absence of the standing wave. Similarly, in this mode, the pressure is everywhere negative at \( t = \pi/\omega_{111} \).

We can look at images for these functions using several different techniques. Let us first look at the isosurfaces corresponding to the values \( p = 0.2, 0.4, 0.6, \) and \( 0.8 \). We generate the images with the statements

\[
\text{>> From this point on, we shall drop the bars on all variables.}
\]

\( ^7 \)From this point on, we shall drop the bars on all variables.
>> img11102 = isosurface( x, y, z, p1110, 0.2 );
>> img11104 = isosurface( x, y, z, p1110, 0.4 );
>> img11106 = isosurface( x, y, z, p1110, 0.6 );
>> img11108 = isosurface( x, y, z, p1110, 0.8 );

Then we display those images with the statements

```matlab
>> clf
>> subplot(2,2,1); hpatch = patch(img11102);
>> axis( [0.0,1.0, 0.0,1.0, 0.0,1.0 ] );
>> set(hpatch,'FaceColor','red','EdgeColor','none')
>> camlight left ; camlight right; lighting phong;
>> axis square; grid on;
>> title('p/A = 0.2');
```

```matlab
>> subplot(2,2,2); hpatch = patch(img11104);
>> axis( [0.0,1.0, 0.0,1.0, 0.0,1.0 ] );
>> set(hpatch,'FaceColor','red','EdgeColor','none')
>> camlight left ; camlight right; lighting phong;
>> axis square; grid on;
>> title('p/A = 0.4');
```

```matlab
>> subplot(2,2,3); hpatch = patch(img11106);
>> axis( [0.0,1.0, 0.0,1.0, 0.0,1.0 ] );
>> set(hpatch,'FaceColor','red','EdgeColor','none')
>> camlight left ; camlight right; lighting phong;
>> axis square; grid on;
>> title('p/A = 0.6');
```

```matlab
>> subplot(2,2,4); hpatch = patch(img11108);
>> axis( [0.0,1.0, 0.0,1.0, 0.0,1.0 ] );
>> set(hpatch,'FaceColor','red','EdgeColor','none')
>> camlight left ; camlight right; lighting phong;
>> axis square; grid on;
>> title('p/A = 0.8');
```

Here, we choose to set the keyword `FaceColor` to red so that the printed image, when converted to a gray scale, is not as dark as it would be if we selected the gray colormap. These four images are shown in the four panels of Fig. E3.56. The upper left panel is an isosurface out near the bounding walls of the cube, and the pressure there is low (and positive). As we move from upper right to upper left to lower left to lower right, the pressure associated with the isosurface increases and, clearly, we move towards the center of the cube as well. If we looked not at \( t = 0 \) but at \( t = \pi/\omega_{111} \), the isosurfaces would look the same but would correspond to negative rather than positive values of \( p \).

A somewhat more interesting and revealing display can be created on screen but not conveyed very well in print. Suppose we created more images and then displayed them in animation. We might then invoke the statements\(^8\)

```matlab
>> for i=1:39
    clf
    pre1110 = isosurface( x, y, z, p1110, 1.0-0.025*i);
    hpatch = patch( pre1110 );
```

\(^8\)We must avoid trying to find an isosurface at \( p = 1.0 \), since there is only one point at which that value occurs—the very center of the box—and a point doth not an isosurface make.
Figure E3.56: Isosurfaces for the state \((l,m,n) = (1,1,1)\) at \(t = 0\). The separate panels correspond to \(p = 0.2\) (upper left), \(p = 0.4\) (upper right), \(p = 0.6\) (lower left), and \(p = 0.8\) (lower right).

\[
\begin{align*}
\text{axis( [0.0,1.0, 0.0,1.0, 0.0,1.0 ] );} \\
\text{set(hpatch,'FaceColor','red','EdgeColor','none')} \\
\text{camlight left ; camlight right; lighting phong; } \\
\text{axis square; grid on; } \\
\text{M(i)=getframe; } \\
\text{end} \\
\text{>> movie( M, 15 )}
\end{align*}
\]

which—in the loop—generate several images and save them in the array \(M\) and then—after the loop—display the sequence 15 times. From the animation, we see that the pressure is greater in the center of the box and decreases as we move farther away.

Alternatively, we can produce surface plots of the pressure in a chosen cross section of the cube. For example, the statements

\[
\begin{align*}
\text{>> subplot(3,2,1); } \\
\text{>> mesh( x(:,:,11), y(:,:,11), p1110(:,:,11), 'EdgeColor', 'black' ); } \\
\text{>> axis( [0.0,1.0, 0.0,1.0, -1.0,1.0] ); } \\
\text{>> title('t=0, z=0.2'); } \\
\text{>> subplot(3,2,2); }
\end{align*}
\]
Figure E3.57: Surface plots of pressure in several planes parallel to the \( xy \) plane.

We can also produce contour plots showing the pressure in various planes. For example, the
Figure E3.58: Contour plots of the pressure in the plane $z = 0.5$ for the mode $(l, m, n) = (1, 1, 1)$.

Statements

```matlab
>> clf
>> contour( x(:,:11), y(:,:11), p1110(:,:,26), 12, 'k' )
>> axis square;
>> title('Contours at z=0.5, t=0')
>> contour( x(:,:11), y(:,:11), -p1110(:,:,26), 12, 'k--' )
>> axis square;
>> title('Contours at z=0.5, t=T/2')
```

produce displays showing the pressure in the plane $z = 0.5$ at $t = 0$ and at $t = T/2$, respectively. The results are shown in Fig. E3.58.

Similar statements will produce isosurfaces and cross sections for other values of $l$, $m$, and $n$. Fig. E3.59 is such an isosurface for $l = 1$, $m = 3$, and $n = 2$ at $p = 0.5$. This isosurface is created by executing the statements

```matlab
>> p1320 = sin(pi*x).*sin(3.0*pi*y).*sin(2.0*pi*z);
>> p1320i = isosurface( x, y, z, p1320, 0.5);
>> hpatch = patch( p1320i );
>> axis([0.0,1.0, 0.0,1.0, 0.0,1.0 ]);
>> set(hpatch,'FaceColor','red','EdgeColor','none')
>> camlight left ; camlight right; lighting phong;
>> axis square; grid on;
```

As with the $(1,1,1)$ mode, we can produce diagrams showing the variation of pressure in particular planes passed through the cube when it supports a standing wave in the $(1,3,2)$ mode. We could, for example, produce surface and contour plots of the pressure in the planes $z = 0.26$ (index = 14) and $z = 0.74$ (index 38) with the statements

```matlab
>> subplot(2,2,1);
```

9We do not pick the plane $z = 0.5$ because the pressure in the standing wave has a node at that value of $z$. The values $z = 0.25$ and $0.75$ would be better, but those value occur at indices 13.5 and 37.5, which we don’t have available to us.
Figure E3.59: \( l = 1, m = 3, n = 2, \) and \( t = 0, \) at \( p = 0.5 \)

The resulting figures are shown in Fig. E3.60.

Similar diagrams for the plane \( x = 0.5 \) would be produced by the statements

```matlab
>> [xx,yy] = meshgrid( 0.0:inc:1.0, 0.0:inc:1.0 );
>> for i=1:51
    for j=1:51
        p1320x(i,j) = p1320(i,26,j);
    end;
end;
>> mesh( xx, yy, p1320x, 'EdgeColor', 'black' );
>> xlabel('z', 'FontSize', 16); ylabel('y', 'FontSize', 16); zlabel('p', 'FontSize', 16 );
```

```matlab
>> mesh( x(:,14), y(:,14), p1320(:,14), 'EdgeColor', 'black' );
>> axis square;
>> title('z=0.26');
>> subplot(2,2,2);
>> mesh( x(:,38), y(:,38), p1320(:,38), 'EdgeColor', 'black' );
>> axis square;
>> title('z=0.74');
>> subplot(2,2,3);
>> contour( x(:,14), y(:,14), p1320(:,14), [0.2 0.4 0.6 0.8], 'k');
>> hold on;
>> contour( x(:,14), y(:,14), p1320(:,14), [-0.2 -0.4 -0.6 -0.8], 'k--');
>> axis square;
>> subplot(2,2,4);
>> contour( x(:,38), y(:,38), p1320(:,38), [0.2 0.4 0.6 0.8], 'k' );
>> hold on;
>> contour( x(:,38), y(:,38), p1320(:,38), [-0.2 -0.4 -0.6 -0.8], 'k--' );
>> axis square;
```
Figure E3.60: Surface and contour plots for pressure in the planes $z = 0.26$ and $z = 0.74$ for the mode $(l, m, n) = (1, 3, 2)$. Solid contour lines represent positive (gauge) pressures; dashed lines represent negative pressures. The figures show pressures at $t = 0$.

```
>> contour( xx, yy, p1320x, [0.0 0.2 0.4 0.6 0.8], 'k' );
>> hold on;
>> contour( xx, yy, p1320x, [-0.2 -0.4 -0.6 -0.8], 'k--' );
>> axis square;
>> xlabel('z', 'FontSize', 16); ylabel('y', 'FontSize', 16);

The resulting figures are shown in Fig. E3.61.

The isosurfaces for $t = \pi/\omega_{lmn}$ are “mirror images” of the ones at $t = 0$ for the same value of $l$, $m$, and $n$, which is what we would expect, since the negative sign would simply flip everything over the axes. We can see this characteristic in Fig. E3.62 where the left figure is drawn for $t = 0$ and the right figure is drawn for $t = T/2 = \pi/\omega_{lmn}$. We produce these figures with the statements

```
>> clf
>> p2220 = sin(2.0*pi*x).*sin(2.0*pi*y).*sin(2.0*pi*z);
>> p2220i = isosurface( x, y, z, p2220, 0.5);
>> hpatch = patch( p2220i );
>> axis( [0.0,1.0, 0.0,1.0, 0.0,1.0 ] );
>> set(hpatch,'FaceColor','red','EdgeColor','none')
>> camlight left ; camlight right; lighting phong;
>> axis square; grid on;
>> title('t=0');
```
Figure E3.61: Contour plots of the pressure in the plane $x = 0.5$ for the mode $(l, m, n) = (1, 3, 2)$.

Figure E3.62: $l = 2$, $m = 2$, and $n = 2$ at $p = 0.5$, where the left figure is drawn for $t = 0$ and the right figure is drawn for $t = T/2 = \pi/\omega_{lmn}$.

```matlab
>> clf
>> p2220i = isosurface( x, y, z, -p2220, 0.5);
>> hpatch = patch( p2220i );
>> axis( [0.0,1.0, 0.0,1.0, 0.0,1.0 ] );
>> set(hpatch,'FaceColor','red','EdgeColor','none')
>> camlight left ; camlight right; lighting phong;
>> axis square; grid on;
>> title('t=T/2');
### 3.37 Animation of a Vibrating String

**Exercise:** The transverse motion of a flexible string of length \( l \) lying nominally along the \( x \) axis and fixed at both ends can be expressed as the superposition

\[
y(x, t) = \sum_{n=1}^{\infty} A_n \sin \frac{n\pi x}{l} \cos \frac{2\pi nt}{T}
\]

of its normal modes of oscillation. Here, \( A_n \) is the amplitude of the \( n \)-th harmonic (and may be negative to convey a 180° phase shift relative to a mode with positive amplitude) and \( T \) is the period of the fundamental mode of oscillation. In particular, the shape of the string at time \( t = 0 \) is given by

\[
y(x, 0) = \sum_{n=1}^{\infty} A_n \sin \frac{n\pi x}{l}
\]

Measuring \( x \) in units of \( l \) and \( t \) in units of \( T \), generate animated displays by writing a procedure that will accept as input a vector giving the amplitudes of the first fifteen harmonics and produce a continuously running display showing the motion of the string when its initial shape is defined by those amplitudes. Test your program with a variety of sets of amplitudes, including but not limited to the first several harmonics by themselves. For example, when the string is pulled aside at its center and released from rest, the amplitude of the first several harmonics will be

\[
1.0, 0.0, -0.111111, 0.0, 0.04, 0.0, -0.0204082, 0.0, 0.0123457, 0.0, -0.00826446, 0.0, 0.00591716, 0.0, -0.00444444
\]

and, when it is pulled aside very near to one end, the amplitude of the \( n \)-th harmonic will be \( 1/n \).

**Solution:** Regarding \( x \) to be measured in units of \( l \) and \( t \) to be measured in units of \( T \), we recast the equations of issue in the forms

\[
y(x, t) = \sum_{n=1}^{\infty} A_n \sin \frac{n\pi x}{l} \cos \frac{2\pi nt}{T} \quad ; \quad y(x, 0) = \sum_{n=1}^{\infty} A_n \sin \frac{n\pi x}{l}
\]

We begin by defining the amplitudes with the statements

\[
A1 = [1.0, 0.0, -0.111111, 0.0, 0.04, 0.0, -0.0204082, 0.0, 0.0123457, 0.0, \ldots
-0.00826446, 0.0, 0.00591716, 0.0, -0.00444444];
\]

```matlab
for n = 1:15 A2(n) = 1.0/n; end
```

Here \( A1 \) contains the first 15 amplitudes for a string drawn aside at its center and \( A2 \) contains the amplitudes for a string drawn aside near one end.

Then, we define a function that accepts the amplitude \( A \) as its one argument and creates an animated display of the motion of the string. The MATLAB code shown in Table 3.2 achieves that objective.\(^\text{10}\) We store this function in a file named `animstring.m` in the default directory. Having defined \( A1 \) and \( A2 \) above, we finally execute the statements

\[
\text{animstring}(A1) \\
\text{animstring}(A2)
\]

to run each animation.

Of course, you must run this program on a computer to view the animation. To show still shots of several frames in the animation, we create the function `stringframes`—see Table 3.3—which we store in the file `stringframes.m` in the default directory and execute with the statements

\[^{10}\text{For some reason, the command `movie` does not seem to work inside a function. Consequently, we have instead invoked the command `imshow`, which—when executed—displays the indicated image. Note the argument selects a particular component of the structure of the image `M`.}\]
Table 3.2: Code for the function `animstring`.

```matlab
function fct = animstring( A )
    % ***** Create frames in the animation *****
    x = [0.0 : 0.025 : 1.0 ]; % Set range of x
    t = [0.0 : 0.025 : 1.0 ]; % Set range of t
    for i = 1:41 % For each time
        y = zeros(1,41); % Initialize y for shape
        for n = 1:15 % Sum harmonic contributions
            y = y + A(n)*sin(n*pi*x)*cos(2*pi*n*t(i));
        end
        plot( x, y, 'color','black, 'linewidth',3 ) % Plot shape
        set( gca, 'ylim', [-1.5,1.5] ) % Set y range
        M(i) = getframe; % Capture image
    end
    movie( M, 5 )
end
```

Table 3.3: Code for the function `stringframes`.

```matlab
function fct = stringframes( A )
    % ***** Create frames in the animation *****
    x = [0.0 : 0.025 : 1.0 ]; % Set range of x
    t = [0.0 : 0.025 : 1.0 ]; % Set range of t
    for i = 1:9 % For each time
        subplot(3,3,i) % Select subplot
        y = zeros(1,41); % Initialize y for shape
        for n = 1:15 % Sum harmonic contributions
            y = y + A(n)*sin(n*pi*x)*cos(2*pi*n*t(3*i-2));
        end
        plot( x, y, 'color','black', 'linewidth',3 ) % Plot shape
        set( gca, 'ylim', [-1.5,1.5] ) % Set y range
        title( ['t = ',num2str(t(3*i-2))], 'fontsize',16 ) % Add time in title
    end
end
```

`stringframes( A1 )`
`stringframes( A2 )`

to produce Fig. E3.63 and Fig. E3.64 showing nine frames in each animation.
Figure E3.63: A multi-plot showing the displacement of the string drawn aside at the center at the indicated times. In the time scale adopted, the full cycle would take 1 time unit.
Figure E3.64: A multi-plot showing the displacement of the string drawn aside near one end at the indicated times. In the time scale adopted, the full cycle would take 1 time unit. Note that, at $t = 0.6$, the pulse has been reflected and is traveling back to the left.
3.38 Animation of a User Function

**Exercise:** The displacement of a string supporting a transverse wave is given by \( u = f(x, t) \) where \( f(x, t) \) is a given function of \( x \) and \( t \). Develop a general procedure to animate this wave propagation (i.e., for showing a sequence of images created by graphing \( f(x, t) \) as a function of \( x \) for a succession of values of \( t \)) and write an appropriate M-file to accomplish this task. Try your procedure with the two functions

\[
\begin{align*}
    f(x, t) &= e^{-(x-t)^2} \\
    f(x, t) &= \sin(x) \cos(t)
\end{align*}
\]

but feel free to invent others of your own choosing. Here, we suppose that \( x \) and \( t \) have been cast in appropriate dimensionless units.

**Solution:** The essential data in this exercise consists of values of the function \( f(x, t) \) for a range of values of \( x \) and \( t \), which ranges will depend on the specific function to be explored. In general terms, we might set

\[
\begin{align*}
    \% \text{ Set range for } x, t \text{ and number of points.} \\
    \text{xr} &= \text{[xmin, xmax, xstpsz]; tr} = \text{[tmin, tmax, tstpsz];}
\end{align*}
\]

where \( \text{xmin, xmax, xstpsz, tmin, tmax, and tstpsz} \) should be floating point values. The beginnings of a procedure to produce the desired animation will then involve the coding

```matlab
function fct = f38( xr, tr, fnct )
\% ***** CREATE DATA FOR ANIMATION *****
\% Calculate values of t and x; determine size of each
x = [xr(1) : xr(3) : xr(2)]; t = [tr(1) : tr(3) : tr(2)];
ts = size(t); xs = size(x);
\% Evaluate function
for i = 1:xs(2) for j = 1:ts(2) f(i,j) = feval(fnct, x(i), t(j) ); end; end;
```

To facilitate animating different functions, we have supposed the function of interest is provided through the function \( \text{fnct} \), a definition for which must be available before the procedure \( \text{f38} \) is invoked; the entered value for this variable must be a string. Each row in the array \( f \) contains the values of \( f \) at several values of \( x \) for the value of \( t \) to which the row corresponds. From this point, following the procedure outlined in Section 3.16 in CPSUP, we produce the animation by adding the coding\(^{11}\)

```matlab
\% ***** CREATE ANIMATION *****
for j = 1:41 % For each of 41 images
    plot( x, f(:,j), 'color','black', 'linewidth',3 ); \% Create display
    ylim( [-1.0 1.0 ] ); \% Standardize y axis
    M(j) = getframe; % Read display as bit map
end
movie( M, 10 )
end
```

to complete the required procedure file. We store the completed file with the name \( \text{f38.m} \) in the default directory.

Before proceeding, we must also define the desired function. For the function \( f(x, t) = \sin(x) \cos(t) \), we might set

\(^{11}\)We accept the default axis range for \( x \) but standardize the axis range on \( y \).
and anticipate that we will scale the arguments of the sine and cosine functions so that sin(x) is actually evaluated at 41 points over the interval $-\pi \leq x \leq \pi$, i.e., over one cycle of the sine function, and cos(t) is actually evaluated at 31 points over the interval $0 \leq t \leq 2\pi$, i.e., over one cycle of the cosine function. With those assumptions, we would define the function $f_{38sc}$ with the statements

```matlab
function fct = f38sc( xx, tt )
    fct = sin(pi*xx)*cos(2.0*pi*tt);
end
```

and store it in the default directory with the name $f_{38sc}.pro$.

With the two files $f38.m$ and $f38sc.m$ available, we execute the procedure and generate the desired animation with the simple statements

```matlab
xr=[-1.0, 1.0, 0.05]; tr=[ 0.0, 1.0, 0.025];
f38( xr, tr, 'f38sc' )
```

Compilation of $f38$ and $f38sc$ will happen automatically each time the function is called.

Of course, you must run this program on a computer to view the animation. Fig. E3.65 shows nine frames in this animation. This figure was produced by replacing the statements in the block labeled ‘CREATE ANIMATION’ with the statements

```matlab
for j = 1:9
    subplot(3,3,j) % Specify subplot
    plot( x, f(:,5*j-4), 'color','black', 'linewidth',3 ); % Create display
    ylim( [-1.0 1.0 ] ); % Standardize y axis
    title( ['t = ',num2str(t(5*j-4))], 'fontsize',16 )
end
```

Alternatively, for the function $f(x,t) = e^{-(x-vt)^2}$, we would define the function $f_{38exp}$ with the statements

```matlab
function fct = f38exp( xx, tt )
    fct = exp(-(xx-tt)^2);
end
```

After a bit of exploration, we select suitable values for $xr$ and $tr$ and create the animation with the statements

```matlab
xr=[0.0,10.0,0.1]; tr=[0.0,10.0,0.2];
f38( xr, tr, 'f38exp')
```

Of course, you must run this program on a computer to view the animation. Fig. E3.66 shows nine frames in this animation. This figure was produced by replacing the statements in the block labeled ‘CREATE ANIMATION’ with the statements

```matlab
for j = 1:9
    subplot(3,3,j) % Specify subplot
    plot( x, f(:,5*j-4), 'color','black', 'linewidth',3 ); % Create display
    ylim( [-1.0 1.0 ] ); % Standardize y axis
    title( ['t = ',num2str(t(5*j-4))], 'fontsize',16 )
end
```
Figure E3.65: A multi-plot showing $f(x,t) = \sin(x) \cos(t)$ for a selected values of $t$. In the scaling adopted, the full cycle would take 1.0 time units.
Figure E3.66: A multi-plot showing \( f(x,t) = e^{-(x-t)^2} \) for a selected values of \( t \).
3.39 Animation of a Square Membrane

Exercise: The displacement of a square membrane extending over the region $0 \leq x, y \leq a$ when it is oscillating in its $m,n$ normal mode is given by

$$u(x,y,t) = A \sin\left(\frac{m\pi x}{a}\right) \sin\left(\frac{n\pi y}{a}\right) \cos(\omega_{mn} t)$$

where $\omega_{mn} = \frac{(c\pi/a)\sqrt{m^2 + n^2}}{c}$, $c$ being the speed of propagation of the waves in the membrane.

Write a function M-file to animate the motion of this drumhead for a user-selected mode (user-selected values of $m$ and $n$). Suggestions: (1) Express $x$ and $y$ in units of $a$ and $u$ in units of $A$ so the graphs you produce will show $u/A$ above the $(x/a)(y/a)$-plane. (2) Review the discussion in the last paragraph of Section 3.16.

Solution: Given the above equation, we can immediately begin writing an M–file that will animate this motion. One condition is that the user is to input the values of $m$ and $n$, so $m$ and $n$ must be included as arguments. Likewise, from looking at the hints, we can also substitute the following: $\bar{x} = x/a$, $\bar{y} = y/a$, and $\bar{u} = u/A$, which leads to the new (and plottable) equation

$$\bar{u}(\bar{x}, \bar{y}, t) = \sin(m\pi \bar{x}) \sin(n\pi \bar{y}) \cos(\omega_{mn} t)$$

in dimensionless form. Notice that, although $t$ still exists in the equation, the term $(c\pi/a)$ in $\omega_{mn}$ is in units of inverse time (frequency), so the units cancel out completely.

Although $u$ is given as a three-dimensional function, it is here convenient to think of it as a succession of $xy$-planes, one for each value of $t$. Thus, we can utilize meshgrid to feed us an array of $x$ and $y$ coordinates, evaluate the $xy$ factors in the function, and then for a range of values of $t$ calculate the value of the function in the $xy$-plane for each $t$ with the statements

```matlab
function f = f39(m, n) % Define procedure
    [ x, y ] = meshgrid( 0.0: 0.05 : 1.0, 0.0: 0.05 : 1.0 ); % Create grid
    u = sin(m*pi*x).*sin(n*pi*y); % Evaluate initial shape
    t = [0.0 : 0.025 : 1.0 ]; % Set times for solutions
    for i = 1:41 disp(:,:,i) = u*cos(2*pi*t(i)); end % Calculate shape for each t
    for i = 1:41
        mesh( x,y,disp(:,:,i), 'edgecolor','black' ); % Create display
        axis( [0.0 1.0 0.0 1.0 -1.0 1.0 ] ); % Standardize axes
        M(i) = getframe; % Read display as bit map
    end
    movie( M, 10 )
end
```

which follow the pattern illustrated in Section 3.16 in CPSUP.

While the animation can be viewed only on a computer screen, several frames from that animation for the $[2,3]$ mode are recorded in Fig. E3.67. For the sake of completeness, we note that this figure was created with the code

```matlab
m=2; n=3; % Set mode
[ x, y ] = meshgrid( 0.0: 0.05 : 1.0, 0.0: 0.05 : 1.0 ); % Create grid
u = sin(m*pi*x).*sin(n*pi*y); % Evaluate initial shape
for i = 1:41 disp(:,:,i) = u*cos(2*pi*t(i)); end % Calculate shape for each t
for i = 1:9
```

```matlab
  disp(:,:,i) = u*cos(2*pi*t(i));
end
```
Figure E3.67: An sequence of plots conveying the shape of the \([2, 3]\) mode of a square membrane at the indicated times. Given the adopted scaling, the full cycle would take 1.0 time units.

```matlab
subplot(3,3,i)
mesh( x,y,disp(:, :, 2*i), 'edgecolor','black')
title([‘t = ’, num2str(t(2*i))], ‘fontsize’,16);
set( gca, ‘fontsize’,10);
axis( [0.0 1.0 0.0 1.0 -1.0 1.0] )
end
```
3.40 Kaprekar Constant

Exercise: Write a program to (a) accept as input an arbitrary two- to four-digit integer with at least two different digits, (b) rearrange the four digits to create (i) the largest and (ii) the smallest integers from the four digits, (c) subtract the smallest number from the largest numbers, and (d) repeat steps (b)–(c) until the same number occurs twice in succession. As a guard against an infinite loop, limit the number of iterations to 10. You should find, probably with some surprise, that the end result is always 6174, known as a Kaprekar Constant, and that no more than seven iterations will be required. Note: If the input number or any result after you complete step (c) has only three digits, add a leading zero before moving to the next iteration. Note: If you code appropriately, the adding of zeroes may happen automatically. Optional: Explore whether numbers with three or five digits exhibit a similar property and, if they do, identify the corresponding Kaprekar Constant. Google “Kaprekar Constant” to locate several articles on this and similar interesting numbers.

Solution: We begin by defining two functions. The first,

function fct = digitss(n)
% Input to this function is a four digit integer.
% Output is a four component vector containing the digits of the input
% d1=floor(n/1000);
% d2 = floor((n-1000*d1)/100);
% d3 = floor((n-1000*d1-100*d2)/10);
% d4 = n-1000*d1-100*d2-10*d3;
% fct = [d1,d2,d3,d4];
endfunction

which is stored in the file digitss.m, accepting a four digit number and returns a vector whose four elements are the individual digits in the number. After setting MATLAB's default directory to the directory in which digitss.m is stored, we can invoke the function to generate sample results with the statements

tmp1 = digitss(4215)
tmp1 = 4 2 1 5
tmp2 = digitss(396)
tmp2 = 0 3 9 6
tmp3 = digitss(24)
tmp3 = 0 0 2 4

Note particularly that the coding adopted to extract the digits will automatically provide leading zeros if a two- or three-digit number happens to be provided. The second,

function fct = differ(tmp)
% Input to this function is a four-component vector of integers
% Output is the difference between the two numbers constructed by
% sorting the input array in increasing and decreasing order.
% n1 = sort(tmp);
% n2 = sort(tmp, 'descend' );
% na = 1000*n1(1) + 100*n1(2) + 10*n1(3) + n1(4); 
% nb = 1000*n2(1) + 100*n2(2) + 10*n2(3) + n2(4); 
% fct = abs(na-nb);
% fprintf( '%6d %6d %6d\n', na, nb, fct )
end

\footnote{The name \texttt{digits} is reserved in MATLAB.}
which is stored in the file differ.m, takes as input the four-component vector output by digitss, sorts the digits into ascending and descending order, reconstructs the two numbers called for in part (b), calculates the (absolute value of) the difference between those two numbers, displays the two numbers and their difference, and returns that difference as the next iterate. Providing the output from the above execution of digitss as input to differ.m, we test differ.m with the statements

\begin{verbatim}
n = differ(tmp1);
  1245  5421  4176
n = differ(tmp2);
  369  9630  9261
n = differ(tmp3);
  24  4200  4176
\end{verbatim}

Again, note that any leading zeroes are properly treated.

With these functions, we can now write a program that asks for a four-digit number and invokes digitss and differ in a while loop to convert the output of each pass through the loop into the next iterate. We introduce the variable n1 to save the current iterate to free n for storing the next iterate. To effect the desired control of the loop, we need outside the loop to initialize an iteration counter and assign a value to n1 that will prevent the loop from terminating immediately. The coding

\begin{verbatim}
n = input(' Enter number: ');  \% Input n
i=0;  \% initialize counter
n1 = 0;  \% Prevent premature termination
while i < 10 & n ~= n1  \% Save current iterate
  n1 = n;
  tmp= digitss(n1);  \% Extract digits
  n = differ(tmp);  \% Calculate next iterate
  i = i + 1;  \% Increment counter
end
\end{verbatim}

will achieve those objectives. We store this program with the name kap4.m. Here are a few sample runs of kap4:

\begin{verbatim}
kap4
Enter number: 4215
  1245  5421  4176
  1467  7641  6174
  1467  7641  6174
kap4
Enter number: 396
  369  9630  9261
  1269  9621  8352
  2358  8532  6174
  1467  7641  6174
kap4
Enter number: 24
  24  4200  4176
  1467  7641  6174
  1467  7641  6174
\end{verbatim}

All three of these examples generated the four-digit Kaprekar Constant 6174 in noticeably fewer than seven iterations.
Chapter 9

Introduction to Programming

9.2 Relationship between IF-THEN-ELSE and CASE

Exercise: Figure E9.1 shows three different alternative structures. Express each structure using (a) only CASE structures and (b) only IF-THEN-ELSE structures. In these figures, T, F, C, and B stand for true, false, condition, and block of statements, respectively. Use proper indentation as illustrated in the examples.

Solution: (a) In the pseudocode of Chapter 9, this figure translates directly into the nested IF-THEN-ELSE control structure

```plaintext
IF C1 THEN B1
   ELSE IF C2 THEN B2
      ELSE B3
   END_IF
END_IF
```

Equivalently, using the CASE structure, we would write

```plaintext
CASE
   OF C1 DO B1
   OF C2 DO B2
   ELSE OTHERS DO B3
END_CASE
```

(b) In the pseudocode of Chapter 9, this figure translates directly into the CASE structure

```plaintext
CASE
   OF C1 DO B1
   OF C2 DO B2
   OF C3 DO B3
   ELSE OTHERS DO B4
END_CASE
```

Equivalently, using the IF-THEN-ELSE structure, we would write
Figure E9.1: Figure for Exercise 9.2.

IF C1 THEN B1
    ELSE IF C2 THEN B2
        ELSE IF C3 THEN B3
            ELSE B4
                END_IF
                END_IF
            END_IF
        END_IF
    END_IF
END_IF

(c) In the pseudocode of Chapter 9, this figure translates directly into the IF-THEN-ELSE structure

IF C1 THEN IF C2 THEN B1
    ELSE B2
    END_IF
ELSE IF C3 THEN B3
    ELSE B4
    END_IF
END_IF
Equivalently, using the CASE structure, we would write

```
CASE
    OF C1 DO CASE
        OF C2 DO B1
        OF OTHERS DO B2
    END_CASE
    OF OTHERS DO CASE
        OF C3 DO B3
        OF OTHERS DO B4
    END_CASE
END_CASE
```

In Fig. E9.2, we redraw each of the diagrams in the statement of the exercise in the alternative form that leads more directly to a casting of the coding in the other of the two possible structures. Recognize that the two diagrams for each part of this exercise are simply topological rearrangements of each other.
9.7 Tracking Both Extremes and their Positions

Exercise: Basing your work on Algorithm (6) of Section 9.2, specifically

```
SENTINEL ← (agreed-upon special value)
READ (first) ITEM from list
EXTREME ← ITEM
LOOP
    READ (next) ITEM from list
    EXIT LOOP WHEN ITEM = SENTINEL
    IF ITEM and EXTREME are out of order
        THEN EXTREME ← ITEM
    END IF
END LOOP
WRITE "The extreme value is "; EXTREME
```

write an algorithm that will obtain words one at a time and ultimately report (1) the word that would appear last if the list were alphabetized, (2) the word that would appear first if the list were alphabetized, (3) the total number of words given, and (4) the position of each extreme word in the original list. Only one pass through the list is permitted.

Solution: Algorithm (6) in Section 9.2 involves scanning through a list of values, keeping track of the extreme value in the list and ultimately displaying that extreme value. This exercise asks that, in scanning the list, we keep track of the earliest value, its position in the list, the latest value, and its position in the list, and that we count the entries in the list as we go so we can report the total number of values entered. For the last item, we will need to count values as they are entered, not only to have the total count at the end but also to have the information along the way that will permit us to note the position of items in the list. In broad outline, we need to execute the statements shown in Table 9.1.

While this program will do the job, note that

- the algorithm will not recognize it if the ultimate earliest or latest entry occurs more than once in the list. If, say the earliest item occurs twice, which occurrence will this algorithm report? How would you modify the algorithm to report the other occurrence?

- the algorithm does not recognize that, if a particular item is earlier than the current earliest item, it is certainly not later than the current latest item. Thus, the separate IF-THEN structures in the loop could be combined into a nested structure in which the algorithm didn’t bother to make the second test if the first test happens to be satisfied.

- the incrementation of CNT, is positioned in the loop so that the sentinel is not counted as a valid entry. In broad terms, one must pay careful attention to the point in the sequence of instructions at which a counter is incremented; off-by-one errors are extremely common in counting operations. Generally speaking, counters should be incremented immediately after the task being counted has been completed and only if the completion represents a bona fide countable occurrence.
Table 9.1: Algorithm for Part (a) of Exercise 9.7.

```
SENTINEL$ <-- ???  ! Set sentinel to mark end of list
CNT% <-- 0        ! Initialize counter to count entries
READ ITEM$ from list ! Get first item
CNT% <-- CNT% + 1% ! Count item entered
EARLIEST$ <-- ITEM$ ! First item is earliest item
LATEST$ <-- ITEM$  ! First item is also latest item
EARLCNT% <-- CNT%  ! Note position of current earliest item
LATCNT% <-- CNT%   ! Note position of current latest item
LOOP
  READ ITEM$ from list ! Get next item
  EXIT_LOOP WHEN ITEM$ = SENTINEL$
  CNT% <-- CNT% + 1% ! Count newly entered item
  IF ITEM$ < EARLIEST$ THEN BEGIN_BLOCK ! Adjust earliest
    EARLIEST$ <-- ITEM$ ! if necessary
    EARLCNT% <-- CNT%
  END_BLOCK
  END_IF
  IF ITEM$ > LATEST$ THEN BEGIN_BLOCK ! Adjust latest
    LATEST$ <-- ITEM$ ! if necessary
    LATCNT% <-- CNT%
  END_BLOCK
END_LOOP
WRITE "The earliest item is"; EARLIEST$; "in position"; EARLCNT%
WRITE "The latest item is"; LATEST$; "in position"; LATCNT%
WRITE "The total number of items is"; CNT%
```
9.9 Mystery Procedure 1

**Exercise:** Suppose you have $N\%$ cards laid out in a row on a table. On each card is a single word. Determine the end result of applying the mystery procedure laid out in Table 9.2 to that array of cards and choose a suitable name for the procedure.

---

Table 9.2: Procedure for Exercise 9.9.

```
PROCEDURE ???????
  SCANEND% ← N%
  LOOP
    CARD% ← 1%
    Obtain word on card CARD% and store in WORD$
    LATEST_WORD$ ← WORD$
    LATEST_CARD% ← CARD%
    LOOP
      CARD% ← CARD% + 1%
      Obtain word on card CARD% and store in WORD$
      IF WORD$ occurs after LATEST_WORD$
      THEN BEGIN_BLOCK
        LATEST_WORD$ ← WORD$
        LATEST_CARD% ← CARD%
      END_BLOCK
      END_IF
      EXIT_LOOP WHEN CARD% = SCANEND%
    END_LOOP
    Exchange card LATEST_CARD% with card SCANEND%
    SCANEND% ← SCANEND% - 1%
    EXIT_LOOP WHEN SCANEND% = 1%
  END_LOOP
END_PROCEDURE
```

---

**Solution:** To determine the function of the procedure in Table 9.2, we begin by looking at what happens in the outermost loop, which begins by

1. setting the counter `CARD%` to 1, i.e., by setting our attention on the first card on the table,
2. fetching the word on that card and storing it in memory location `WORDS$`,
3. copying that word into a memory location labeled `LATEST_WORDS$`, and
4. recording the card number in a memory location labeled `LATEST_CARD%`.

These several actions prime the inner loop, in each pass through which we

1. move our sights to the next card on the table,
2. copy the word on that card into `WORDS$`,
3. adjust the value of `LATEST_WORDS$` and `LATEST_CARD%` to be the content and position of the newly examined card, but only if its contents `WORDS$` occurs later in the alphabet than the current contents of `LATEST_WORDS$`.

The loop continues until we have examined the `SCANEND%`-th card on the table, at which point `LATEST_WORDS$` contains the particular word on the cards from the first through the `SCANEND%`-th
that occurs latest in the alphabet and \texttt{LATEST\_CARD}\% contains the position on the table of the card containing that word.

Upon exit from the inner loop, we exchange the \texttt{SCANEND}\%-th card with the \texttt{LATEST\_CARD}\%-th card, thereby moving to the \texttt{SCANEND}\%-th position the card in the subgroup of cards from 1 thru \texttt{SCANEND}\%, i.e., we move the card in that subgroup that will be latest in the alphabet to the last position in the subgroup!

Then, we decrement \texttt{SCANEND}\% and go back to the beginning of the outermost loop.

Note that the outer loop starts with \texttt{SCANEND}\% set to the number of cards on the table. Thus, in the first pass through that outer loop, we move the card in the entire stack that is latest in the alphabet to the last position in the stack. Then we ignore that last card by decrementing the counter on the outer loop. At the end of the next pass through the outer loop, we move the “latest-in-the-alphabet” card in the first \(N\%-1\)% cards to the bottom of that stack. Then, we move the “latest-in-the-alphabet” card in the first \(N\%-2\)% cards to the bottom of that stack, ... Every time we move a card, we move the card displaced by the “latest-in-the-alphabet” card to a position earlier in the stack—so it will continue to be examined in each pass through the loops until such time as it, itself, becomes the “latest-in-the-alphabet” card, moves to the bottom, and ceases to be further examined.

By the time this mystery procedure has completed execution, the cards on the table have been arranged in alphabetic order.
Chapter 11

Solving ODEs

11.17 Van der Pol Oscillator (MATLAB)

Exercise: Explore the behavior of the Van der Pol oscillator described in dimensionless form by the equation

\[
\frac{d^2 \pi}{dt^2} = \frac{d\pi}{dt}(1 - \pi^2) - \pi
\]

obtaining graphs of position versus time, velocity versus time, and velocity versus position (the phase-plane trajectory), each for several different initial conditions. Convince yourself that the final, steady-state path in the phase plane is independent of the initial conditions.

Solution: To cast the Van der Pol equation as a pair of first-order equations, we introduce the correspondences \( \pi \mapsto x(1) \) and \( \frac{d\pi}{dt} \mapsto x(2) \). Further, we establish the correspondence \( \tau \mapsto t \). In this notation, the equations describing the behavior of the oscillator are

\[
\frac{dx(1)}{d\tau} = x(2) ; \quad \frac{dx(2)}{d\tau} = x(2)(1 - x(1)^2) - x(1)
\]

Thus, since there are no parameters in the equations, a suitable m-file defining these equations is

```matlab
function derivs = vandpol( t, x )
derivs = [ x(2); x(2)*(1 - x(1)^2) - x(1) ];
```

With this m-file stored with the name `vandpol.m` in the default directory, we find the solution to the default tolerance of \( 10^{-3} \) for several initial conditions with the statements

```matlab
>> ic1 = [1.0; 0.0];
>> [t1,x1] = ode23( 'vandpol', [0.0, 40.0], ic1 );
>> ic2 = [2.0; -2.0];
>> [t2,x2] = ode23( 'vandpol', [0.0, 40.0], ic2 );
>> ic3 = [0.1; 0.0];
>> [t3,x3] = ode23( 'vandpol', [0.0, 40.0], ic3 );
>> ic4 = [0.0; 0.5];
>> [t4,x4] = ode23( 'vandpol', [0.0, 40.0], ic4 );
```

We determine appropriate initial conditions and time intervals by trial and error, choosing a time interval so as to embrace several periods of the motion and choosing initial conditions so that we start at four widely different points in phase space.
Once the solutions are in hand (though, in truth, we made several preliminary plots along the way to this point), we plot the graphs in Fig. E11.1 of the displacement versus time with the statements

```matlab
>> subplot(2,2,1)
>> plot( t1, x1(:,1), 'Color', 'black', 'LineWidth', 3.0 )
>> xlabel( 'Time' ); ylabel( 'Position' ); title( 'x(0)=1.0, v(0)=0.0' )
>> subplot(2,2,2)
>> plot( t2, x2(:,1), 'Color', 'black', 'LineWidth', 3.0 )
>> xlabel( 'Time' ); ylabel( 'Position' ); title( 'x(0)=2.0, v(0)=-2.0' )
>> subplot(2,2,3)
>> plot( t3, x3(:,1), 'Color', 'black', 'LineWidth', 3.0 )
>> xlabel( 'Time' ); ylabel( 'Position' ); title( 'x(0)=0.1, v(0)=0.0' )
>> subplot(2,2,4)
>> plot( t4, x4(:,1), 'Color', 'black', 'LineWidth', 3.0 )
>> xlabel( 'Time' ); ylabel( 'Position' ); title( 'x(0)=0.0, v(0)=0.5' )
```

and the graphs in Fig. E11.2 of velocity versus displacement (the phase plots) with the statements

```matlab
>> subplot(2,2,1)
>> plot( x1(:,1),x1(:,2), 'Color', 'black', 'LineWidth', 3.0 )
>> xlabel( 'Position' ); ylabel( 'Velocity' ); title( 'x(0)=1.0, v(0)=0.0' )
>> subplot(2,2,2)
>> plot( x2(:,1),x2(:,2), 'Color', 'black', 'LineWidth', 3.0 )
>> xlabel( 'Position' ); ylabel( 'Velocity' ); title( 'x(0)=2.0, v(0)=-2.0' )
>> subplot(2,2,3)
>> plot( x3(:,1),x3(:,2), 'Color', 'black', 'LineWidth', 3.0 )
>> xlabel( 'Position' ); ylabel( 'Velocity' ); title( 'x(0)=0.1, v(0)=0.0' )
>> subplot(2,2,4)
>> plot( x4(:,1),x4(:,2), 'Color', 'black', 'LineWidth', 3.0 )
>> xlabel( 'Position' ); ylabel( 'Velocity' ); title( 'x(0)=0.0, v(0)=0.5' )
```

Note that, ultimately the trajectory in the phase plane settles down to a single path, regardless of the initial conditions and, because the same path (to the resolution of the graph, anyway) is traced repeatedly, the motion ultimately becomes periodic—though clearly not sinusoidal or simple harmonic (which would result in an elliptical path). The only difference between the steady state motions emerging from different initial conditions is a phase difference.

To make the point of identical phase-plane trajectories even more persuasively, let us plot all four solutions on top of one another with the statements

```matlab
>> subplot(1,1,1)
>> plot( x1(:,1),x1(:,2), 'Color', 'black', 'LineWidth', 3.0 )
>> xlabel( 'Position' ); ylabel( 'Velocity' )
>> hold on;
>> plot( x2(:,1),x2(:,2), 'Color', 'black', 'LineWidth', 3.0 )
>> plot( x3(:,1),x3(:,2), 'Color', 'black', 'LineWidth', 3.0 )
>> plot( x4(:,1),x4(:,2), 'Color', 'black', 'LineWidth', 3.0 )
```

finding the graph in Fig. E11.3.
Figure E11.1: Graphs of position versus time for the Van der Pol oscillator for the indicated initial conditions.

Figure E11.2: Phase-plane plots of velocity versus position for the Van der Pol oscillator for the indicated initial conditions.
Figure E11.3: Phase-plane plots of velocity versus position for the Van der Pol oscillator for the indicated initial conditions.
11.18 Large Amplitude Pendulum (MATLAB)

**Exercise:** The angular position \( \theta \) of a simple pendulum of length \( l \) satisfies the non-linear equation

\[
\frac{d^2\theta}{dt^2} + \frac{g}{l} \sin \theta = 0
\]

where \( \theta \) is measured in radians from the lowest point of the pendulum’s motion. Use numerical methods to study the motion of this pendulum when it is released from rest at each of several initial displacements, say 20°, 45°, 90°, 120°, 150°, 165°, and 178°. Look particularly at graphs of \( \theta \) versus \( t \), \( \frac{d\theta}{dt} \) versus \( t \), and \( \frac{d\theta}{dt} \) versus \( \theta \) (the phase plot). Obtain also a graph of period versus amplitude (initial displacement). Write several paragraphs describing your set up of the problem and presenting evidence for your discoveries. *Optional:* Try starting the pendulum at the bottom (0 initial angle) with several initial angular velocities. How large can the angular velocity be before the pendulum begins to swing over the top? *Suggestion:* Begin by introducing the dimensionless time \( \bar{t} = \sqrt{\frac{g}{l}} t \) so that the equation becomes \( \frac{d^2\theta}{d\bar{t}^2} + \sin \theta = 0 \).

**Solution:** Starting with the equation

\[
\frac{d^2\theta}{dt^2} + \frac{g}{l} \sin \theta = 0
\]

we begin by introducing the dimensionless time \( \bar{t} = \sqrt{\frac{g}{l}} t \), in terms of which the equation becomes

\[
\frac{d^2\theta}{d\bar{t}^2} + \sin \theta = 0
\]

As a pair of first-order equations, we would alternatively have the two equations

\[
\frac{d\theta}{dt} = \omega ; \quad \frac{d\omega}{dt} = -\sin \theta
\]

(We have dropped the bars.) To address this equation numerically using MATLAB, we adopt the correspondences \( \theta(t) \mapsto x(1) \) and \( \phi(t) \mapsto x(2) \), and then we create the function profile

```matlab
function derivs = sp0602( t, x )
    derivs = [x(2); -sin(x(1)) ];
end
```

and store it in the default directory with the name `sp0602.m`. Then we launch MATLAB set the default directory appropriately and, accepting the default tolerances and method, execute the following statements

```matlab
ic = [ 15.0*pi/180.0; 0.0 ];
stl = [' -k'; ' :k'; ' -k'; ' :k'; ' -k'; ' :k'; ' -k' ];
[t, x] = ode23( @sp0602, [0.0 10.0], ic );
plot( t, x(:,1), stl(1,:), 'linewidth',2 )
```

to set the initial conditions for release from rest at a displacement of 15°, generate the solution over the interval \( 0.0 < t < 10.0 \), and plot a graph of angular displacement as a function of time for that interval. (We elect not to display the graph produced by the statement in this paragraph.)

To produce a graph showing the behavior of this pendulum for several different amplitudes, we begin by setting a vector containing several different initial conditions (both in degrees and in radians) with the statements

```matlab
phi0 = [20.0, 45.0, 90.0, 120.0, 150.0, 165.0, 178.0];
phi0rad = pi*phi0/180.0;
```
In trial runs, we discover (1) that we need to run the solution over the time interval \(0.0 < t < 25.0\) to cover the entire period of the motion at the largest amplitude and (2) that the default tolerances and method are adequate at all amplitudes. Further, let’s set up so we can plot the angular displacement in degrees, which will range from \(-180^\circ\) to \(180^\circ\) by calculating and plotting the first line and establishing the axes with the statements\(^1\)

\[
\text{ic} = \begin{bmatrix} \phi0\text{rad}(1); 0.0 \end{bmatrix}; \\
\text{[t, x]} = \text{ode23}(\text{@sp0602}, [0.0 \text{ 25.0}], \text{ic}); \\
\text{plot}( t, \text{180.0*}x(:,1)/\pi, \text{stl}(1,:), \text{'linewidth'},2 ) \\
\text{axis}([0.0 \text{ 25.0} \text{ -200.0 200.0 }]) \\
\text{grid on} \\
\text{hold on}
\]

Then, we generate and plot each solution in turn within the loop

\[
\text{for } i = 2:7 \\
\quad \text{ic} = \begin{bmatrix} \phi0\text{rad}(i), 0.0 \end{bmatrix}; \\
\quad [t, x] = \text{ode23}(\text{@sp0602}, [0.0 \text{ 25.0}], \text{ic}); \\
\quad \text{plot}( t, \text{180.0*}x(:,1)/\pi, \text{stl}(i,:), \text{'linewidth'},2 ) \\
\text{end}
\]

and, finally, enlarge the labels on the tick marks and label the axes with the statements

\[
\text{set(gca, 'fontsize', 16)} \\
\text{xlabel('t', 'fontsize', 16)} \\
\text{ylabel('\theta', 'fontsize', 16)} \\
\text{hold off}
\]

The resulting graph is shown in Fig. E11.4. Note that the period gets longer as the amplitude increases and that the pendulum dwells longer and longer near its highest point as the amplitude increases.

To plot angular velocity versus time, reduce the number of initial amplitudes (to render the display clearer) and then plot \(x(:,2)\) rather than \(x(:,1)\) with the statements

\[
\phi0 = [\text{20.0, 90.0, 135.0, 165.0, 178.0}]; \\
\phi0\text{rad} = \pi\times\phi0/180.0; \\
\text{ic} = \begin{bmatrix} \phi0\text{rad}(1); 0.0 \end{bmatrix}; \\
\text{[t, x]} = \text{ode23}(\text{@sp0602}, [0.0 \text{ 25.0}], \text{ic}); \\
\text{plot}( t, x(:,2), \text{stl}(1,:), \text{'linewidth'},2 ) \\
\text{hold on} \\
\text{for } i = 2:5 \\
\quad \text{ic} = \begin{bmatrix} \phi0\text{rad}(i), 0.0 \end{bmatrix}; \\
\quad [t, x] = \text{ode23}(\text{@sp0602}, [0.0 \text{ 25.0}], \text{ic}); \\
\quad \text{plot}( t, x(:,2), \text{stl}(i,:), \text{'linewidth'},2 ) \\
\text{end} \\
\text{grid on} \\
\text{set(gca, 'fontsize', 16)} \\
\text{xlabel('t', 'fontsize', 16)} \\
\text{ylabel('\omega', 'fontsize', 16)}
\]

though it took a bit of exploration to determine an appropriate range for the vertical coordinate in these graphs. The resulting graph is shown in Fig. E11.5.

\(^1\)Note that each string in an array of strings apparently has to be of the same length.
Figure E11.4: Angular position as a function of time for amplitudes 20.0°, 45.0°, 90.0°, 120.0°, 150.0°, 165.0°, and 178.0°.

Next, for the original initial amplitudes, we would generate the phase-plane plots with the statements

\begin{verbatim}
phi0 = [20.0, 45.0, 90.0, 120.0, 150.0, 165.0, 178.0];
phi0rad = pi*phi0/180.0;
ic = [phi0rad(1); 0.0 ];
[t, x] = ode23( @sp0602, [0.0 25.0], ic );
plot( 180.0*x(:,1)/pi, x(:,2), stl(1,:), 'linewidth',2 )
grid on
hold on
for i=2:7
\end{verbatim}

Figure E11.5: Angular velocity as a function of time for amplitudes 20.0°, 90.0°, 135.0°, 165.0°, and 178.0°.
Exercise 11.18 (MATLAB)

Figure E11.6: Phase-plane plot for amplitudes 20.0°, 45.0°, 90.0°, 120.0°, 150.0°, 165.0°, and 178.0°.

To generate the phase-plane plot shown in Fig. E11.6.

Finally, we obtain a graph of period versus amplitude. There is no simple way of finding the period directly when we start with a solution for the differential equation. In essence, we must scan the angular position of the solution for each amplitude to find the time at which the angular position returns to its starting value. Alternatively (and this may be easier), we could scan the angular velocity of the solution for each amplitude to find the time at which the angular velocity returns to zero. Let us adopt the latter strategy. We will, however, need more initial amplitudes to generate an adequately smooth graph of period versus amplitude. Thus, let’s set a new vector of amplitudes, both in degrees and in radians, with the statements

```matlab
phi0 = [1 : 1 : 179 ];
phi0rad = pi*phi0/180.0;
```

Thus, we elect to determine the period for amplitudes starting at 1° and continuing in 1° increments to 178°. Then, within the loop

```matlab
for i = 1:179
    ic = [ phi0rad(i); 0.0 ];
    [t, x] = ode23( @sp0602, [0.0 25.0], ic );
    j=5;
    while (x(j,2) < 0.0) j=j+1; end
    period(i) = 2.0*( t(j) - x(j,2)*(t(j+1)-t(j))/(x(j+1,2)-x(j,2)) )
end
```
we

- set the next initial conditions (first statement in for loop),
- solve the equations for those conditions (lsode statement),
- search the velocity for the index at which the velocity first becomes positive (statement j=5 and while loop,
- interpolate to find the time at which the velocity assumes the value zero, calculating the period from that time and storing the resulting value in the proper place in the vector of periods (last statement in for loop.

Then, we plot the desired graph of period versus amplitude with the statement

```matlab
plot(phi0, period/period(1), 'color','black', 'linewidth',2)
grid on
axis( [0.0 200.0 0.0,4.0] );
set( gca, 'fontsize', 16 )
xlabel('Amplitude in Degrees', 'fontsize', 16 )
ylabel('Relative Period', 'fontsize', 16 )
```

Here, we have elected to plot period in units of the small amplitude period by dividing each period by the period for the smallest amplitude for which we computed the period. The graph is shown in Fig. E11.7.
11.19 Planet in non-inverse square gravity (MATLAB)

Exercise: Suppose that the “gravitational” force were not inverse square but instead depended on some other (negative) power of the radial coordinate. As laid out in Section 11.1.7, the dimensionless equations of motion then would be

\[
\frac{d^2\vec{r}}{dt^2} = -\frac{\vec{r}}{(\vec{r}^2 + \vec{y}^2)^b} ; \quad \frac{d^2\vec{y}}{dt^2} = -\frac{\vec{y}}{(\vec{r}^2 + \vec{y}^2)^b}
\]

Of course, the equations reduce to those for the inverse square force if we simply set \(b = 3/2\). For the planetary problem, find conditions that will generate a distinctly elliptical orbit for an attractive inverse square force \((b = 1.5)\). Then explore the effect on that orbit of distorting the force by changing the exponent in the denominator of the equations making \(b = 1.45, b = 1.55,\) or anything else you can think of, and write a paragraph or two describing the nature of the changes in some detail. Make sure your solutions are generated to an adequate accuracy to support your conclusions.

Solution: The m-file \textit{planet.m} deduced in Section 9.8.6 is designed to facilitate the exploration requested in this problem as well. With the correspondences \(\vec{r}_x \leftrightarrow x(1), \vec{r} \leftrightarrow x(2), \vec{y} \leftrightarrow x(3),\) and \(\vec{y}_y \leftrightarrow x(4),\) we there created the m-file

\[
\textit{function derivs = planet( t, x, flag, b ) }
\]

\[
temp = ( x(1)^2+x(2)^2 )^b;
\]

\[
derivs = [x(2); -x(1)/temp, x(4), -x(3)/temp ];
\]

to return the derivatives of the four dependent variables involved in the set of four first-order ODEs describing the planetary problem. We have already determined that the initial conditions

\[
\vec{r}(0) = 4.0 ; \quad \vec{r}_x(0) = 0.0 ; \quad \vec{y}(0) = 0.0 ; \quad \vec{y}_y(0) = 0.3
\]

lead to a distinctly elliptical orbit when \(b = 1.5\) (inverse square force). Making sure to track far enough into the future for the planet to orbit its sun several times (and taking several preliminary trials to find time appropriate time intervals), we track the motion using different values of \(b\) with the statements

\[
\text{>> ics = [ 4.0; 0.0; 0.0; 0.3 ];}
\]

\[
\text{>> b = 1.50;}
\]

\[
\text{>> [t1,x1] = ode45( 'planet', [0.0 150.0], ics, '', b );}
\]

\[
\text{>> b = 1.55;}
\]

\[
\text{>> [t2,x2] = ode45( 'planet', [0.0 150.0], ics, '', b );}
\]

\[
\text{>> b = 1.45;}
\]

\[
\text{>> [t3,x3] = ode45( 'planet', [0.0 150.0], ics, '', b );}
\]

\[
\text{>> b = 1.60;}
\]

\[
\text{>> [t4,x4] = ode45( 'planet', [0.0 150.0], ics, '', b );}
\]

Then, plotting in order of increasing \(b\) and establishing equal scales on the two axes, we plot the orbits with the statements

\[
\text{>> subplot(2,2,1)}
\]

\[
\text{>> plot( x3(:,1),x3(:,3), 'Color', 'black', 'LineWidth', 3.0 )}
\]

\[
\text{>> axis equal; title( 'b = 1.45' )}
\]

\[
\text{>> set( gca, 'XLim', [-4.0,4.0], 'YLim', [-4.0,4.0] )}
\]

\[
\text{>> subplot(2,2,2)}
\]

\[
\text{>> plot( x1(:,1),x1(:,3), 'Color', 'black', 'LineWidth', 3.0 )}
\]

\[
\text{>> axis equal; title( 'b = 1.50' )}
\]
Figure E11.8: Planetary orbits in non-inverse square force for indicated values of $b$.

we obtain the graphs shown in Fig. E11.8. The graph for $b = 1.5$ is—as expected—a closed elliptical orbit. When $b \neq 1.5$, the orbit precesses. The precession is clockwise when $b < 1.5$ and counterclockwise when $b > 1.5$. Further, the rate of precession increases as the $b$ departs further and further from the value 1.5. 
11.21 Satellite with two suns (MATLAB)

**Exercise:** Deduce the equations of motion for a space ship of mass $m$ coasting freely in the $xy$ plane under the gravitational influence of two suns, each of mass $M$ and located respectively at $(R, 0)$ and $(-R, 0)$. Then express the equations in dimensionless form and, creating all necessary files, thoroughly explore the motion of this space ship. In particular, you might search for an orbit that loops like a figure-eight around the two suns and/or you might see if your approach predicts what you would expect intuitively if you start the spaceship from rest at a point on the perpendicular bisector of the line joining the two suns. *Make sure your solutions are generated to an adequate accuracy.*

**Solution:**

As shown in the figure to the right, this problem focuses on the motion in the $xy$ plane of a satellite of mass $m$ in the field of two suns, each of mass $M$. The planet and the two suns are located at the points

$$r = x \hat{i} + y \hat{j}; \quad r_1 = R \hat{i}; \quad r_2 = -R \hat{i}$$

From the law of gravitation, the force experienced by the satellite then is given by

$$F = -GMm \frac{r - r_1}{|r - r_1|^3} - GMm \frac{r - r_2}{|r - r_2|^3}$$

$$= -GMm \frac{(x - R) \hat{i} + y \hat{j}}{[(x - R)^2 + y^2]^{3/2}} - GMm \frac{(x + R) \hat{i} + y \hat{j}}{[(x + R)^2 + y^2]^{3/2}}$$

$$= -\frac{GMm}{R^2} \frac{(x/R - 1) \hat{i} + (y/R) \hat{j}}{[(x/R - 1)^2 + (y/R)^2]^{3/2}} - \frac{GMm}{R^2} \frac{(x/R + 1) \hat{i} + (y/R) \hat{j}}{[(x/R + 1)^2 + (y/R)^2]^{3/2}}$$

and the equation of motion for the satellite then is

$$m \frac{d^2(x \hat{i} + y \hat{j})}{dt^2} = -\frac{GMm}{R^2} \frac{(x/R - 1) \hat{i} + (y/R) \hat{j}}{[(x/R - 1)^2 + (y/R)^2]^{3/2}} - \frac{GMm}{R^2} \frac{(x/R + 1) \hat{i} + (y/R) \hat{j}}{[(x/R + 1)^2 + (y/R)^2]^{3/2}}$$

Dividing this equation by $R$, we find next that

$$m \frac{d^2((x/R) \hat{i} + (y/R) \hat{j})}{dt^2} = -\frac{GMm}{R^3} \frac{(x/R - 1) \hat{i} + (y/R) \hat{j}}{[(x/R - 1)^2 + (y/R)^2]^{3/2}} - \frac{GMm}{R^3} \frac{(x/R + 1) \hat{i} + (y/R) \hat{j}}{[(x/R + 1)^2 + (y/R)^2]^{3/2}}$$

or, in terms of the dimensionless variables $\bar{x} = x/R$ and $\bar{y} = y/R$, that

$$\frac{d^2(\bar{x} \hat{i} + \bar{y} \hat{j})}{dt^2} = -\frac{GM}{R^3} \frac{(\bar{x} - 1) \hat{i} + \bar{y} \hat{j}}{[(\bar{x} - 1)^2 + \bar{y}^2]^{3/2}} - \frac{GM}{R^3} \frac{(\bar{x} + 1) \hat{i} + \bar{y} \hat{j}}{[(\bar{x} + 1)^2 + \bar{y}^2]^{3/2}}$$
Finally, we introduce the dimensionless time $\tilde{t} = \sqrt{\frac{GM}{R^3}} t$ to find that

$$\frac{d^2(\bar{x} \hat{i} + \bar{y} \hat{j})}{dt^2} = -\frac{(\bar{x} - 1) \hat{i} + \bar{y} \hat{j}}{[(\bar{x} - 1)^2 + \bar{y}^2]^{3/2}} - \frac{(\bar{x} + 1) \hat{i} + \bar{y} \hat{j}}{[(\bar{x} + 1)^2 + \bar{y}^2]^{3/2}}$$

or, in component form, that

$$\frac{d^2\bar{x}}{dt^2} = -\frac{\bar{x} - 1}{[(\bar{x} - 1)^2 + \bar{y}^2]^{3/2}} - \frac{\bar{x} + 1}{[(\bar{x} + 1)^2 + \bar{y}^2]^{3/2}}$$

and

$$\frac{d^2\bar{y}}{dt^2} = -\frac{\bar{y}}{[(\bar{x} - 1)^2 + \bar{y}^2]^{3/2}} - \frac{\bar{y}}{[(\bar{x} + 1)^2 + \bar{y}^2]^{3/2}}$$

Then, with the correspondences $\bar{x} \mapsto \text{pos}[0]$, $d\bar{x}/dt \mapsto \text{pos}[1]$, $\bar{y} \mapsto \text{pos}[2]$, and $d\bar{y}/dt \mapsto \text{pos}[3]$, we would construct the m-file

```matlab
function derivs = twosuns( t, pos )
tmp1 = 1.0 / ( (pos(1) - 1.0)^2 + pos(3)^2 )^1.5;
tmp2 = 1.0 / ( (pos(1) + 1.0)^2 + pos(3)^2 )^1.5;
xacc = -(pos(1) - 1.0)*tmp1 - (pos(1) + 1.0)*tmp2;
yacc = -pos(3) * (tmp1 + tmp2);
derivs = [ pos(2); xacc; pos(4); yacc ];
```

As a first test of the above analysis and the pro-file we have created, let us ask about the motion when the satellite is released from rest at a point on the $y$ axis. The statements

```matlab
>> ics = [0.0; 0.0; 2.0; 0.0 ];
>> [t1,x1] = ode23( 'twosuns', [0.0, 50.0], ics );
>> plot( x1(:,1), x1(:,3), 'Color', 'black', 'LineWidth', 3.0 );
>> plot( t1, x1(:,1), 'Color', 'black', 'LineWidth', 3.0 );
>> plot( t1, x1(:,3), 'Color', 'black', 'LineWidth', 3.0 );
```

produce three graphs (not shown) in which (1) the trajectory of the satellite is confined to the $y$ axis, (2) the $y$ coordinate oscillates through several cycles, staying in the interval $-2.0 \leq y \leq 2.0$, and (3) the $x$ coordinate stays fixed at $x = 0.0$. All of this is as expected and provides some evidence of the adequacy of the specified default tolerance.

Next, again as a test, we suppose that the satellite is started off quite some distance from the two suns on the $x$ axis and given a velocity perpendicular to that axis (i.e., a $y$ velocity only). After some exploration with the $y$ velocity, we invoke the statements

```matlab
>> ics = [20.0; 0.0; 0.0; 0.3 ];
>> [t1,x1] = ode23( 'twosuns', [0.0, 400.0], ics );
>> plot( x1(:,1), x1(:,3), 'Color', 'black', 'LineWidth', 3.0 );
```

which generates a graph (again, not shown) of a closed orbit around both suns. This result, too, is expected, since from a distance 10 times the separation of the two suns, the two suns look almost like a single sun of twice the mass, and the orbit should be close to the ellipse we would expect if there were only one sun. This outcome provides further justification of the adequacy of our approach and choice of tolerances.

Finally, let’s move the satellite in a bit closer and explore a few trajectories with different initial conditions:

1. Start on the $y$ axis with an initial $x$ velocity:

   Some exploration was needed before the initial conditions actually presented were identified.

2. If, as you explore this problem, the message ‘Beware of singularity’ appears, you should simply conclude that the adaptive algorithm failed to converge at some point after executing the maximum number of iterations, and the solution was terminated. The satellite probably collided with one of the suns, and a plot of the data that IDL does return will probably reveal as much.
Exercise 11.21 (MATLAB)

```matlab
>> ics = [ 0.0; 1.0; 2.0; 0.0 ];
>> [t1,x1] = ode23( 'twosuns', [0.0, 50.0], ics );
>> plot( x1(:,1), x1(:,3), 'Color', 'black', 'LineWidth', 3.0 );
```

The result is shown in (a) of Fig. E11.9. The orbit circles both suns and appears not to be heading for any very immediate disaster, either through collision with a sun or through escape from the suns altogether.

(b) Start at the origin with a velocity directed initially at some angle to the $x$ axis:

```matlab
>> ics = [ 0.0; 0.9*cos(pi/4.0); 0.0; 0.9*sin(pi/4) ];
>> [t2,x2] = ode23( 'twosuns', [0.0, 10.0], ics );
>> plot( x2(:,1), x2(:,3), 'Color', 'black', 'LineWidth', 3.0 );
```

The result is shown in (b) of Fig. E11.9. Here, we have generated an orbit that is moving around both suns in a figure-8-like pattern.

(c) Start at $(x,y) = (2.0,0.0)$ with a velocity directed towards positive $y$:

```matlab
>> ics = [ 2.0; 0.0; 0.0; 1.0 ];
>> [t3,x3] = ode23( 'twosuns', [0.0, 7.5], ics );
>> plot( x3(:,1), x3(:,3), 'Color', 'black', 'LineWidth', 3.0 );
```

The result is shown in (c) of Fig. E11.9. Here, we have a fairly quick collision with the sun on the left.

(d) Tampering with the initial velocity given in the previous example, we find ultimately that the statements

```matlab
>> ics = [ 2.0; 0.0; 0.0; 0.902 ];
>> [t4,x4] = ode23( 'twosuns', [0.0, 25.0], ics );
>> plot( x4(:,1), x4(:,3), 'Color', 'black', 'LineWidth', 3.0 );
```

produce an orbit that seems to be converging on a closed figure 8. The result is shown in (d) of Fig. E11.9.

In all of these cases, we have had to play a bit with the time interval, since `ode23` doesn’t seem to sense that the satellite has fallen into one of the suns and stop execution automatically.

As a further embellishment of this solution, one might look at energy conservation as an indicator of the adequacy of the solution.
Figure E11.9: Graphs for the four examples described in the solution.
11.22 Charge in Crossed Fields (MATLAB)

Exercise: Suppose a particle of charge $q$ and mass $m$ is injected into a region of space containing constant, crossed electric and magnetic fields $\mathbf{E} = E_x \hat{i}$ and $\mathbf{B} = B_z \hat{k}$. In vector form, the equation of motion for this particle is

$$m \frac{d^2 \mathbf{r}}{dt^2} = q \mathbf{E} + q \frac{d \mathbf{r}}{dt} \times \mathbf{B}$$

Verify the equations of motion

$$m \frac{d^2 x}{dt^2} = qE_x + qB_z \frac{dy}{dt} \quad ; \quad m \frac{d^2 y}{dt^2} = -qB_z \frac{dx}{dt} \quad ; \quad m \frac{d^2 z}{dt^2} = 0$$

for the specific fields of this exercise, express them in dimensionless form (note that $\omega = qB_z/m$ is a frequency and $E_x/B_z$ is a velocity), and thoroughly explore the behavior of the particle in this situation. Try to understand the motion intuitively. Hint: You should find that, in terms of an arbitrarily selected unit of length $\ell$, the equations involve a single parameter $qE_x/(m\omega^2 \ell)$, which may alternatively be written as $(E_x/B_z)/(\omega \ell)$ — the ratio of the velocity $E_x/B_z$ determined by the fields to the characteristic velocity implied by your choice of a length unit and the frequency $\omega$. Note that this exercise actually has more than one parameter, since the initial components of the velocity—probably expressed in units of $\omega \ell$—also influence the solution.

Solution: The first part of this exercise is to verify the second set of equations from the first set. We begin by substituting components for all the vectors in the basic equation, finding that

$$m \frac{d^2 (x \hat{i} + y \hat{j} + z \hat{k})}{dt^2} = qE_x \hat{i} + q \left( \frac{d(x \hat{i} + y \hat{j} + z \hat{k})}{dt} \right) \times B_z \hat{k}$$

$$= qE_x \hat{i} + qB_z \frac{d(-x \hat{j} + y \hat{i})}{dt}$$

The desired equations are the $x$, $y$, and $z$ components of this vector equation.

The second part of the exercise asks us to cast these individual equations in dimensionless form. Dividing all three equations by $m$, we obtain

$$\frac{d^2 x}{dt^2} = \frac{E_x}{m} + \omega \frac{dy}{dt} \quad ; \quad \frac{d^2 y}{dt^2} = -\omega \frac{dx}{dt} \quad ; \quad \frac{d^2 z}{dt^2} = 0$$

Now, setting $\bar{x} = x/\ell$, $\bar{y} = y/\ell$, $\bar{z} = z/\ell$, and $\bar{t} = \omega t$, we find that

$$\frac{d^2 \bar{x}}{dt^2} = \frac{E_x}{m\omega^2 \ell} + \frac{d\bar{y}}{dt} \quad ; \quad \frac{d^2 \bar{y}}{dt^2} = -\frac{d\bar{x}}{dt} \quad ; \quad \frac{d^2 \bar{z}}{dt^2} = 0$$

Hereafter, we drop the overbars and use the symbol $\omega$ to stand for $qE_x/m\omega^2 \ell$.

At this point, we recognize that the motion in the $z$-direction is motion with zero acceleration, which implies constant velocity. That motion is independent of the projection of the motion into the $xy$-plane. Consequently, we can focus on the $x$ and $y$ motion. To set up the function that will define the $x$ and $y$ equations for MATLAB’s procedure $\texttt{ode23}$, we begin by establishing the correspondences

$$x \rightarrow n(1), \quad y \rightarrow n(2), \quad \frac{dx}{dt} \rightarrow n(3), \quad \frac{dy}{dt} \rightarrow n(4)$$

in terms of which the equations become

$$\frac{dn(1)}{dt} = n(3), \quad \frac{dn(2)}{dt} = n(4), \quad \frac{dn(3)}{dt} = w + n(4), \quad \frac{dn(4)}{dt} = -n(3)$$

and the function defining these equations for $\texttt{ode23}$ then is
function derivs = eqs( t, n, w )
derivs = [n(3), n(4), w+n(4), -n(3)];
end

Here, we provide the parameter \( w \) as an additional argument to this function and will communicate that parameter to the function \texttt{eqs} by exploiting the feature of \texttt{ode23} that provides explicitly for that possibility.

Once this function has been defined in MATLAB, we can find and plot the solution for several values of \( w \) when the particle is started off at the origin at rest with the coding

\[
ic = [ 0.0; 0.0; 0.0; 0.0 ];
\]

\[
w = 0.2;
[t, n02] = ode23( @eqs, [0.0,20.0], ic, [], w );
subplot(2,2,1)
plot( n02(:,1), n02(:,2), 'linewidth',2, 'color','black' )
set(gca, 'fontsize',16)
title( 'w = 0.2', 'fontsize',20 )
xlabel('x', 'fontsize',20); ylabel('y', 'fontsize',20 )
\]

\[
w = 0.5;
[t, n05] = ode23( @eqs, [0.0,20.0], ic, [], w );
subplot(2,2,2)
plot( n05(:,1), n05(:,2), 'linewidth',2, 'color','black' )
set(gca, 'fontsize',16)
title( 'w = 0.5', 'fontsize',20 )
xlabel('x', 'fontsize',20); ylabel('y', 'fontsize',20 )
\]

\[
w = 1.0;
[t, n10] = ode23( @eqs, [0.0,20.0], ic, [], w );
subplot(2,2,3)
plot( n10(:,1), n10(:,2), 'linewidth',2, 'color','black' )
set(gca, 'fontsize',16)
title( 'w = 1.0', 'fontsize',20 )
xlabel('x', 'fontsize',20); ylabel('y', 'fontsize',20 )
\]

\[
w = 1.5;
[t, n15] = ode23( @eqs, [0.0,20.0], ic, [], w );
subplot(2,2,4)
plot( n15(:,1), n15(:,2), 'linewidth',2, 'color','black' )
set(gca, 'fontsize',16)
title( 'w = 1.5', 'fontsize',20 )
xlabel('x', 'fontsize',20); ylabel('y', 'fontsize',20 )
\]

The four graphs generated by this code are shown in Fig. E11.10. The general trajectory has the same form for all values of \( w \). Only the scale along both axes changes. Because of the electric field in the \( \hat{i} \) direction, this particle is initially accelerated in that direction. Once the particle starts moving, however, the magnetic field exerts an additional force in the direction \( \hat{i} \times \hat{k} = -\hat{j} \), so the particle is nudged in the negative \( y \) direction. Overall, the particle oscillates in the \( x \) direction.

Graphs of \( x, y, v_x \), and \( v_y \) as functions of \( t \) for \( w = 0.5 \) can be produced with the statements\(^4\)

\(^4\)Because \texttt{ode23} creates its own set of values of \( t \), we need here to recalculate \( n05 \) so as to have the lengths of \( t \) and \( n05 \) match.
Exercise 11.22 (MATLAB)

Figure E11.10: The trajectory for the indicated values of $w$ when the charge is started at the origin with zero initial velocity.

```
[t, n05] = ode23( @eqs, [0.0,20.0], ic, [], w );
subplot(2,2,1)
plot( t, n05(:,1), 'linewidth',2, 'color','black' )
set(gca, 'fontsize',16)
title( 'x vs. t for w=0.5', 'fontsize',20 )
xlabel('t', 'fontsize',20); ylabel('x', 'fontsize',20)

subplot(2,2,2)
plot( t, n05(:,2), 'linewidth',2, 'color','black' )
set(gca, 'fontsize',16)
title( 'y vs. t for w=0.5', 'fontsize',20 )
xlabel('t', 'fontsize',20); ylabel('y', 'fontsize',20)

subplot(2,2,3)
plot( t, n05(:,3), 'linewidth',2, 'color','black' )
set(gca, 'fontsize',16)
title( 'v_x vs. t for w=0.5', 'fontsize',20 )
xlabel('t', 'fontsize',20); ylabel('dx/dt', 'fontsize',20)

subplot(2,2,4)
plot( t, n05(:,4), 'linewidth',2, 'color','black' )
set(gca, 'fontsize',16)
title( 'v_y vs. t for w=0.5', 'fontsize',20 )
xlabel('t', 'fontsize',20); ylabel('dy/dt', 'fontsize',20)
```

The results are shown in Fig. E11.11. The graph of $y$ versus $t$ reveals a steady motion towards more and more negative values of $y$, i.e., a drift in the direction of $\mathbf{E} \times \mathbf{B}$ though, as judged from the graph of $v_y$ versus $t$, the velocity varies sinusoidally around the average value $\langle v_y \rangle = 0.5$ in our dimensionless units.
Finally, let’s explore what happens when $w = 0.5$, the charge is started at the origin with initial velocities of magnitude 1.5 in the $x$, $y$, $-x$, and $-y$ directions. The coding

```matlab
w=0.5; v = 1.5; vs=num2str(v);

ic = [ 0.0, 0.0,v, 0.0 ];
[t, n1] = ode23( @eqs, [0.0,20.0], ic, [] , w);
subplot(2,2,1)
plot( n1(:,1), n1(:,2), 'linewidth',2, 'color','black' )
axis( [-2 3 -12 0] )
set(gca, 'fontsize',16)
title( ['v_x=',vs], 'fontsize',20 )
xlabel('x', 'fontsize',20); ylabel('y', 'fontsize',20)

ic = [ 0.0, 0.0, 0.0, v ];
[t, n2] = ode23( @eqs, [0.0,20.0], ic, [] , w);
subplot(2,2,2)
plot( n2(:,1), n2(:,2), 'linewidth',2, 'color','black' )
axis( [-1 5 -12 2] )
set(gca, 'fontsize',16)
title( ['v_y=',vs], 'fontsize',20 )
xlim( [-1.0,5.0] )
xlabel('x', 'fontsize',20); ylabel('y', 'fontsize',20)

ic = [ 0.0, 0.0, -v, 0.0 ];
[t, n3] = ode23( @eqs, [0.0,20.0], ic, [] , w);
subplot(2,2,3)
plot( n3(:,1), n3(:,2), 'linewidth',2, 'color','black' )
```

The choice of the value 1.5 was reached after exploring several other choices. This choice revealed the looping behavior most clearly.
Figure E11.12: The trajectory for $w = 0.5$ when all initial values except the one indicated are zero.

\begin{verbatim}
axis([-2 3 -10 2])
set(gca, 'fontsize',16)
title(['v_x=-',vs], 'fontsize',20)
xlabel('x', 'fontsize',20); ylabel('y', 'fontsize',20)
ic = [ 0.0, 0.0, 0.0, -v ];
[t, n4] = ode23( @eqs, [0.0,20.0], ic, [], w );
subplot(2,2,4)
plot( n4(:,1), n4(:,2), 'linewidth',2, 'color','black' )
axis([-3 1 -12 0])
set(gca, 'fontsize',16)
title(['v_y=-',vs], 'fontsize',20)
xlim([-3.0,1.0])
xlabel('x', 'fontsize',20); ylabel('y', 'fontsize',20)
\end{verbatim}

produces Fig. E11.12. Remember that these trajectories all lie in the $xy$-plane. If the charge is given an initial velocity in the $z$ direction (parallel to the $B$ field), that constant velocity on the $z$ direction (out of the plane) is superimposed on the trajectory shown in these figures.
11.24 The Lorenz attractor (MATLAB)

Exercise: An important system in the early study of chaos is described by the Lorenz equations

\[
\begin{align*}
\frac{dx}{dt} &= a(y - x) \\
\frac{dy}{dt} &= -xz + bx - y \\
\frac{dz}{dt} &= xy - cz
\end{align*}
\]

Create an appropriate file defining these equations and then thoroughly explore the behavior of this system. Graphs of \(y\) versus \(x\), \(z\) versus \(x\), and \(z\) versus \(y\) when \(a = 10.0\), \(b = 28.0\) and \(c = 8.0/3.0\) under the initial conditions \(x_0 = 1.0\), \(y_0 = 0.0\), and \(z_0 = 0.0\) are particularly interesting. While graphs of \(y\) versus \(x\), \(z\) versus \(x\), and \(z\) versus \(y\) are interesting, the true beauty of the trajectory is best seen using a three-dimensional space curve. Be sure to examine the path from several different vantage points in the space around the path, an objective most easily accomplished if the display of the path allows rotation of the path on the screen.

Solution: For the Lorenz system with three parameters \(a\), \(b\), and \(c\), we declare the correspondences \(x \mapsto \text{var}[0]\), \(y \mapsto \text{var}[1]\), and \(z \mapsto \text{var}[2]\). Then, we create the pro-file

```
function derivs = lorenz( t, var, flag, a, b, c )
    dxdt = a*(var(2)-var(1));
    dydt = -var(1)*var(3) + b*var(1);
    dzdt = var(1)*var(2) - c*var(3);
    derivs = [ dxdt; dydt; dzdt ];
```

which returns the derivatives of the dependent variables. We elect to use \texttt{ode23}. Then, to find a solution over the time interval from \(0.0 \leq t \leq 50.0\) (which is a guess as to an appropriate interval) to the default precision and the parameters given in the exercise, we use the statements

```
>> ics = [ 5.0; 0.0; 0.0 ];
>> [t1,sln1] = ode23( 'lorenz', [0.0, 50.0 ], ics, '', 10.0, 28.0, 8.0/3.0 );
```

With this solution now in hand, we can generate several graphs. First we look at the behavior of each state variable as a function of time with the statements

```
>> subplot(2,2,1)
>> plot( t1, sln1(:,1), 'Color', 'black' ); title('X1 versus T');
>> subplot(2,2,2)
>> plot( t1, sln1(:,2), 'Color', 'black' ); title('X1 versus T');
>> subplot(2,2,3)
>> plot( t1, sln1(:,3), 'Color', 'black' ); title('X1 versus T');
```

generating the graph in Fig. E11.13. Then, we look at various projections of the 3D trajectory into planes in phase space with the statements

```
>> clf
>> subplot(2,2,1)
>> plot( sln1(:,1), sln1(:,2), 'Color', 'black' ); title('X2 versus X1');
>> subplot(2,2,2)
>> plot( sln1(:,2), sln1(:,3), 'Color', 'black' ); title('X3 versus X2');
>> subplot(2,2,3)
>> plot( sln1(:,3), sln1(:,1), 'Color', 'black' ); title('X1 versus X3');
```
The resulting graph is shown in Fig. E11.14. Finally, reading appropriate scalings from the graphs we already have, we generate a graph of the three-dimensional trajectory with the statements

```matlab
>> plot3( sln(:,1), sln(:,2), sln(:,3), 'Color', 'black' )
```

which generates Fig. E11.15.
Figure E11.14: Trajectories in 2D phase planes.

Figure E11.15: Trajectory in 3D phase space.
11.27 Dynamics of a Chemical Reaction (MATLAB)

**Exercise:** The dynamics of the chemical reaction

\[ \text{A} + \text{B} \rightleftharpoons \text{C} + \text{D} \]

is governed by the equations

\[
\begin{align*}
\frac{dA}{dt} &= -k_f AB + k_r CD \\
\frac{dB}{dt} &= -k_f AB + k_r CD \\
\frac{dC}{dt} &= k_f AB - k_r CD \\
\frac{dD}{dt} &= k_f AB - k_r CD
\end{align*}
\]

where \( A(t), B(t), C(t), \) and \( D(t) \) are the concentrations of each molecule in the reaction vessel, and \( k_f \) and \( k_r \) are the forward and reverse rate constants, respectively. Suppose that the reaction is started with \( A(0) = A_0, B(0) = B_0, \) and \( C(0) = D(0) = 0. \) Cast the equations in dimensionless form, using \( A_0 \) as the unit of concentration and \( k_f A_0 t \) as the dimensionless time. Then explore the behavior of the system as a function of the initial concentration of \( B, \) measured in units of \( A_0 \) and the reverse rate constant, measured in units of \( k_f. \) Look particularly at the dependence of the ultimate equilibrium on these parameters. Make sure your results are generated to adequate accuracy.

**Solution:** To put the above equations into dimensionless form, we first need to realize that this system of equations is not linear. \( A, B, C, \) and \( D \) are all in units of concentration, meaning that we have concentration-squared on the right side and concentration-over-time on the left. Because of this, we must introduce units of inverse concentration for the constants \( k_f \) and \( k_r. \) Since the units are consistent on both sides now, we can substitute \( AB = (AB)/A_0^2, \) \( CD = (CD)/A_0^2 \) and absorb the extra \( A_0 \) term into the time variable, \( \bar{t} = t(k_f A_0). \) Dividing through by \( k_f \) and dropping the bars then yields the dimensionless equations

\[
\begin{align*}
\frac{dA}{dt} &= -AB + \frac{k_r}{k_f} CD \\
\frac{dB}{dt} &= -AB + \frac{k_r}{k_f} CD \\
\frac{dC}{dt} &= AB - \frac{k_r}{k_f} CD \\
\frac{dD}{dt} &= AB - \frac{k_r}{k_f} CD
\end{align*}
\]

To write the function defining these equations for MATLAB’s routine `lsode`, we establish the associations

\[ A \mapsto n(1) \ ; \ B \mapsto n(2) \ ; \ C \mapsto n(3) \ ; \ D \mapsto n(4) \]

and introduce \( k \) to stand for \( k_r/k_f. \) In these terms, the equations become

\[
\begin{align*}
\frac{dn(1)}{dt} &= -n(1) \ast n(2) + k \ast n(3) \ast n(4) \\
\frac{dn(2)}{dt} &= -n(1) \ast n(2) + k \ast n(3) \ast n(4) \\
\frac{dn(3)}{dt} &= n(1) \ast n(2) - k \ast n(3) \ast n(4) \\
\frac{dn(4)}{dt} &= n(1) \ast n(2) - k \ast n(3) \ast n(4)
\end{align*}
\]

These equations can then be solved and plotted in MATLAB in the standard way of creating graphs of ODEs. With the parameter \( k \) being communicated from command level through `ode23` to the function by exploiting the feature of `ode23` that provides explicitly for communicating parameters, the function `chemreac` defined with the statements

```matlab
function derivs = chemreac(t, n, k )
templ = -n(1)*n(2)+k*n(3)*n(4);
temp2 = n(1)*n(2)-k*n(3)*n(4);
derivs = [ templ; templ; temp2; temp2];
end
```
Figure E11.16: An example of a reversible chemical reaction. $k = 0.00001$.

serves to return the values of the derivatives. This coding must be executed by MATLAB before \texttt{lsode} can be invoked. Then, the statements

\begin{verbatim}
ic = [2.0; 1.0; 0.0; 0.0];
k = 0.00001;
[t, soln] = ode23( @chemreac, [0.0,3.0], ic, [], k );
plot( t, soln(:,1), 'color','black', 'linewidth',3 )
title( 'Reversible Chemical Reaction', 'fontsize',20 )
xlabel( 'Time', 'fontsize',20 ); ylabel( 'Concentration', 'fontsize',20 )
set( gca, 'fontsize',16 )
hold on
for i =2:4
    plot( t, soln(:,i), 'color','black', 'linewidth',3 )
end
text( 2.5,1.15, 'A', 'fontsize',16 )
text( 2.5,0.85, 'C,D', 'fontsize',16 )
text( 2.5,0.15, 'B', 'fontsize',16 )
hold off
\end{verbatim}

shows the basics necessary to produce accurate plots of the reactions, as can be viewed in Fig. E11.16. Here, the two decreasing graphs are those of $A$ and $B$, while the increasing graphs, which are on top of each other, are $C$ and $D$. Interesting to note is that at all points in the reaction the total concentration is 3.0, i.e., the number of molecules is conserved.

As can be seen in Fig. E11.17 and Fig. E11.18, changing $k$ (and the positions of the internal labels) changes the graphs significantly. The total concentration of the chemicals never changes, but the rate at which equilibrium is approached, and indeed the equilibrium point itself, both change. Higher values for $k$ mean that the ratio of forward-to-reverse rate of reaction drops, and therefore more reverse reaction is taking place. These extra reactions move the equilibrium points closer to the initial concentrations, and hence they are reached earlier.
Figure E11.17: An example of a reversible chemical reaction. $k = 0.5$.

Figure E11.18: An example of a reversible chemical reaction. $k = 2.0$. 
The Predator-Prey Problem (MATLAB)

**Exercise:**
In classical ecology, the interaction between a predator and a prey, with populations \( x(t) \) and \( y(t) \), respectively, is modeled with the equations
\[
\frac{dx}{dt} = -k_1 x + k_2 xy ; \quad \frac{dy}{dt} = k_3 y - k_4 xy
\]
where \( k_1 \) and \( k_3 \) are parameters describing the way each population would evolve in the absence of the other and \( k_2 \) and \( k_4 \) are parameters describing strength of the interaction between the two species, which we take to be proportional to the likelihood of an encounter between a member of one species and a member of the other species. Depending on the parameters and the initial populations \( x(0) = x_0, y(0) = y_0 \), the system may approach a stable equilibrium or, alternatively, one or the other of the populations may become extinct. Explore this system to determine conditions under which each of these circumstances occurs and write a paragraph or two describing your findings. Make sure your results are generated to an accuracy adequate to support your conclusions.

**Solution:**
We can begin by assuming that this set of equations is already in dimensionless form by choosing the correct units for \( k_1, k_2, k_3 \) and \( k_4 \). Since this is true, all we really need to do then is write a procedure and a set of command-line instructions that will plot a set of graphs depicting the various scenarios. The procedure file `predprey.m`

```matlab
function derivs = predprey( t, n, k1, k2, k3, k4 )
%
% In this function, n[1] = x, n[2] = y, and the return values
% are dx/dt = -k1*x + k2*x*y and dy/dt = k3*y - k4*x*y.
%
% returns a two-element vector which contains the values of the two derivatives and is stored in the
% default directory. The main set of code, entered at command level,
% % k1 = 2.0 ; k2 = 0.3 ; k3 = 0.5 ; k4 = 0.1;
% % ic = [9.0; 18.0];
% % [t, n] = ode23( @predprey, [0.0, 6.0], ic, [], k1, k2, k3, k4 );
% % plot( t, n, 'linewidth',2.0, 'color','black' )
% % title('Predator-Prey Relationships','fontsize',20)
% % xlabel('t', 'fontsize',20 ); ylabel( 'x,y', 'fontsize',20)
% % text( 4.0, 5.0, 'Prey', 'fontsize',16 )
% % text( 1.0, 25.0, 'Predators', 'fontsize',16 )
% % set(gca, 'fontsize',16)
% %
% % determines the properties of the graph. The above code, for example, produced Fig. E11.19, while
% % a slightly-modified set of code
% k1 = 11.0; k2 = 1.0; k3 = 4.0; k4 = 1.0;
% ic = [9.0; 18.0];
% [t, n] = ode23( @predprey, [0.0, 6.0], ic, [], k1, k2, k3, k4 );
% plot( t, n(:,1), 'linewidth',2.0, 'color','black', 'linestyle','-')
% hold on
% plot( t, n(:,2), 'linewidth',2.0, 'color','black', 'linestyle','--')
% title('Predator-Prey Relationships','fontsize',20)
% xlabel('t', 'fontsize',20 ); ylabel( 'x,y', 'fontsize',20)
% set(gca, 'fontsize',16)
% hold off
```

returns a two-element vector which contains the values of the two derivatives and is stored in the default directory. The main set of code, entered at command level,

```matlab
k1 = 2.0 ; k2 = 0.3 ; k3 = 0.5 ; k4 = 0.1;
ic = [9.0; 18.0];
[t, n] = ode23( @predprey, [0.0, 6.0], ic, [], k1, k2, k3, k4 );
plot( t, n, 'linewidth',2.0, 'color','black' )
title('Predator-Prey Relationships','fontsize',20)
xlabel('t', 'fontsize',20 ); ylabel( 'x,y', 'fontsize',20)
text( 4.0, 5.0, 'Prey', 'fontsize',16 )
text( 1.0, 25.0, 'Predators', 'fontsize',16 )
set(gca, 'fontsize',16)
determines the properties of the graph. The above code, for example, produced Fig. E11.19, while
a slightly-modified set of code
```
Figure E11.19: Plot showing the eventual extinction of the predator. This plot is a bit idealized, in that it has been cut off after $t = 6.0$ because the population unrealistically rebounds after it is less than 1. This rebound could be due to the inaccuracy of the model at small populations or internal computer roundoff.

Produced the plot shown in Fig. E11.20. A graph showing a pure equilibrium (highly idealized) is displayed in Fig. E11.21 and was found by letting $k_2 x - k_1 = 0$ and $k_3 - k_4 x = 0$, which causes $dx/dt = dy/dt = 0$. This plot was created with $x_0 = 9.0$, $y_0 = 18.0$, $k_1 = k_3 = 9.0$, $k_2 = 0.5$, $k_4 = 1.0$. 
Figure E11.20: A plot showing an unstable equilibrium between predator (solid line) and prey (dashed line). The equilibrium is periodic, in that neither predator nor prey goes extinct, but the populations fluctuate quite a bit.

Figure E11.21: A pure equilibrium between predator (solid line) and prey (dashed line). This scenario is very contrived, seeing as how reproduction and death never precisely coincide with each other.
Exercise 11.31 (MATLAB)

11.31 The Elastic Swing (MATLAB)

Exercise: (also CPSUP 11.31) One type of child’s swing is hung with elastic ropes. Suppose the ropes are long enough so that the child/swing can be represented by a point mass at the end of a spring, assume the spring obeys Hooke’s law with constant $k$ and has an unstretched length $a$, and let the motion of the mass be confined to a single vertical plane. Show that, in the coordinate system illustrated, the equations of motion are

$$m\frac{d^2x}{dt^2} = -kx + \frac{kx}{\sqrt{x^2 + y^2}}; \quad m\frac{d^2y}{dt^2} = mg - ky + \frac{ky}{\sqrt{x^2 + y^2}}$$

Then, introducing $\omega_0^2 = k/m$, $\bar{t} = \omega_0 t$, $\bar{x} = x/a$, and $\bar{y} = y/a$, cast the equations in dimensionless form. After the suggested rescalings, only one parameter—$g/a\omega_0^2$—remains and this parameter is the square of the ratio of the frequency $\sqrt{g/a}$ of a simple pendulum of length $a$—call it the "swing" frequency—to the frequency $\sqrt{k/m}$ of an object of mass $m$ bobbing on a spring of stiffness $k$—call it the "bounce" frequency. Using MATLAB, explore the motions for several values of this one parameter, including values larger than, equal to, and smaller than 1. Write several paragraphs describing and presenting evidence for your discoveries.

Solution: When the position of the child is $(x, y)$, the distance of the child from the origin is $\sqrt{x^2 + y^2}$ and the stretch of the spring is $\sqrt{x^2 + y^2} - a$, which will be an extension (positive) or a compression (negative) depending on whether the distance of the child from the origin is greater than or less than the unstretched length $a$ of the spring. The magnitude of the force exerted on the child by the spring is

$$|F_{\text{child}}| = k\left|\sqrt{x^2 + y^2} - a\right|$$

Further, if the stretch is positive, the spring exerts a force on the child directed towards the origin. Thus, the $x$ and $y$ components of the force on the child are

$$F_x = -k\left(\sqrt{x^2 + y^2} - a\right)\left(\frac{x}{\sqrt{x^2 + y^2}}\right); \quad F_y = -\left(\sqrt{x^2 + y^2} - a\right)\left(\frac{y}{\sqrt{x^2 + y^2}}\right)$$

Here, $x/\sqrt{x^2 + y^2}$ is the cosine of the angle between the spring and the $x$ axis and $y/\sqrt{x^2 + y^2}$ is the cosine of the angle between the spring and the $y$ axis. Newton’s second law for the motion of this child on the swing becomes

$$m\ddot{x} = F_x \implies m\frac{d^2x}{dt^2} = -kx + \frac{kx}{\sqrt{x^2 + y^2}}$$

$$m\ddot{y} = mg + F_y \implies m\frac{d^2y}{dt^2} = mg - ky + \frac{ky}{\sqrt{x^2 + y^2}}$$

Q.E.D.

These equations are decidedly non-linear and hence have no simple analytic solution. To find solutions, we are forced to adopt a numerical approach. As always, casting the problem in a dimensionless form is prudent. We divide the equations by $ma$, finding that

$$\frac{d^2x/a}{dt^2} = -\frac{kx}{ma} + \frac{kx}{m\sqrt{x^2 + y^2}}; \quad \frac{d^2y/a}{dt^2} = \frac{g}{a} - \frac{ky}{ma} + \frac{k}{m\sqrt{x^2 + y^2}}$$
Next, we set \( k/m = \omega_0^2 \), \( x/a = \bar{x} \), \( y/a = \bar{y} \), and \( \omega_0 t = \bar{t} \) to find that

\[
\frac{d^2 \bar{x}}{dt^2} = -\bar{x} + \frac{x}{\sqrt{x^2 + y^2}} \quad ; \quad \frac{d^2 \bar{y}}{dt^2} = \frac{g/a}{\omega_0^2} - \bar{y} + \frac{y}{\sqrt{x^2 + y^2}}
\]

Finally, we set \( g/(a\omega_0^2) = b \) and drop the overbars to find the simplest basic equations of motion, specifically

\[
\frac{d^2 x}{dt^2} = -x + \frac{x}{\sqrt{x^2 + y^2}} \quad ; \quad \frac{d^2 y}{dt^2} = b - y + \frac{y}{\sqrt{x^2 + y^2}}
\]

We view this pair of second-order equations as the quartet

\[
\frac{dx}{dt} = v \quad ; \quad \frac{dv}{dt} = -x + \frac{x}{\sqrt{x^2 + y^2}} \quad ; \quad \frac{dy}{dt} = w \quad ; \quad \frac{dw}{dt} = b - y + \frac{y}{\sqrt{x^2 + y^2}}
\]

and, for the sake of the numerical approach, we introduce the four-component vector \( X \) composed of the correspondences

\[
x \mapsto X[0] \quad ; \quad v \mapsto X[1] \quad ; \quad y \mapsto X[2] \quad ; \quad w \mapsto X[3]
\]

in terms of which, with primes representing time derivatives, the basic equations become

\[
X(1)' = X(2) \quad ; \quad X(3)' = -X(1) + X(1)/\text{SQRT}(X(1)^2 + X(3)^2)
\]

\[
X(3)' = X(4) \quad ; \quad X(4)' = B - X(3) + X(3)/\text{SQRT}(X(1)^2 + X(3)^2)
\]

The MATLAB function

```matlab
function derivs = elasticswing(t,X)
global B
tmp = sqrt(X(1)^2 + X(3)^2);
derivs = [ X(2); -X(1) + X(1)/tmp; X(4); B - X(3) + X(3)/tmp ];
end
```

takes as input the time and the four-component vector containing the current position and velocities of the child and returns the four-component vector containing the associated velocities and accelerations. We store that function with the name `elasticswing.m` in the default directory and then, taking \( B = 1.0 \) (swing frequency = bounce frequency) and the initial conditions \([ 0.0, 0.0, 1.0, 0.0 ]\), which start the child at rest from the position directly below the point of suspension with the spring neither stretched nor compressed, we submit the statements\(^6\)

\[
global B
B = 1.0;
ic = [ 0.0; 0.0; 1.0; 0.0 ];
[ t, x1 ] = ode23(@elasticswing, [0.0,10.0], ic );
subplot(221)
plot( t,x1(:,1), 'linewidth',3, 'color','black')
grid on
title('x/a vs. t', 'fontsize',16)
axis( [0.0,10.0,-0.6,0.6] )
set(gca, 'fontsize',12)
end
```

\(^6\)We accept the default tolerances for `lsode`. Since those tolerances are on the order of \( 10^{-8} \), they should be plenty adequate in the present case.
Figure E11.22: Position and velocity when \( b = 1 \) and \( [x(0)/a, v(0)/a\omega_0, y(0)/a, w(0)/a\omega_0] = [0.0, 0.0, 1.0, 0.0] \). The motion is confined to bouncing up and down on the \( y \) axis.

Suppose, still with \( B = 1.0 \), that the child’s father draws the swing aside to the point \( x(0) = 1, y(0) = 1 \), i.e., so that \( ic = [1.0, 0.0, 1.0, 0.0] \). If we carry the result for 30 rather than 10 time units, the resulting graphs then are produced with the statements

```matlab
subplot(222)
plot( t,x1(:,2), 'linewidth',3, 'color','black')
grid on
title('v/a\omega_0 vs. t', 'fontsize',16)
axis( [0.0,10.0,-0.6,0.6] )
set(gca, 'fontsize',12)

subplot(223)
plot( t,x1(:,3), 'linewidth',3, 'color','black')
grid on
title('y vs. t', 'fontsize',16)
set(gca, 'fontsize',12)

subplot(224)
plot( t,x1(:,4), 'linewidth',3, 'color','black')
grid on
title('w/a\omega_0 vs. t', 'fontsize',16)
set(gca, 'fontsize',12)
```
\begin{verbatim}
B = 1.0;
ic = [ 1.0; 0.0; 1.0; 0.0 ];
[ t, x1 ] = ode23(@elasticswing, [0.0,30.0], ic);

subplot(221)
plot( t,x1(:,1), 'linewidth',3, 'color','black')
grid on
title('x/a vs. t', 'fontsize',16)
set(gca, 'fontsize',12)

subplot(222)
plot( t,x1(:,2), 'linewidth',3, 'color','black')
grid on
title('v/a\omega_0 vs. t', 'fontsize',16)
set(gca, 'fontsize',12)

subplot(223)
plot( t,x1(:,3), 'linewidth',3, 'color','black')
grid on
title('y vs. t', 'fontsize',16)
set(gca, 'fontsize',12)

subplot(224)
plot( t,x1(:,4), 'linewidth',3, 'color','black')
grid on
title('w/a\omega_0 vs. t', 'fontsize',16)
set(gca, 'fontsize',12)

As shown in Fig. E11.23, these conditions create a more exciting ride for the child.

A more interesting graph of this motion—see Fig. E11.24—shows not the individual components of position and velocity but the actual path followed by the child. That graph is produced from the solution already in hand with the statements

\begin{verbatim}
subplot(1,1,1)
plot( x1(:,1), x1(:,3), 'linewidth',3, 'color','black')
grid on
title('y/a vs. x/a', 'fontsize',16)
set(gca, 'fontsize',12)
axis( 'square' )
text( 0.95,1.02, 'O', 'fontsize',16) % mark starting point
\end{verbatim}

The value $b = (g/a)/\omega_0^2 = 1$ means that $g/a = \omega_0^2$, i.e., that the square of the swing frequency is \emph{equal to} the square of the bounce frequency. Changing $b$ makes one of those frequencies larger than the other. For example, with $b = 0.5$, the swing frequency is smaller than the bounce frequency. We find the solution in that case with the statements

\begin{verbatim}
global B
B=0.5;
ic = [ 1.0; 0.0; 1.0; 0.0 ];
[ t, x1 ] = ode23(@elasticswing, [0.0,30.0], ic);
\end{verbatim}

Fig. E11.25 and Fig. E11.26, which are produced by the same statements that produced Fig. E11.23 and Fig. E11.24,\footnote{Only the coordinates of the mark on the origin in Fig. E11.26 were changed–to (0.95,1.0).} show the results with these initial conditions. Compare the frequencies of the $x$
\end{verbatim}
Figure E11.23: Position and velocity when $b = 1$ and $[x(0)/a, v(0)/a\omega_0, y(0)/a, w(0)/a\omega_0] = [1.0, 0.0, 1.0, 0.0]$. With these initial conditions, the child swings back and forth while simultaneously bounces up and down.

Figure E11.24: The trajectory of the child when $b = 1$ and $[x(0)/a, v(0)/a\omega_0, y(0)/a, w(0)/a\omega_0] = [1.0, 0.0, 1.0, 0.0]$. The motion starts at the point marked with a circle at coordinates $(x/a, y/a) = (1.0, 1.0)$. 
Exercise 11.31 (MATLAB)

Figure E11.25: Position and velocity when $b = 0.5$ and $[x(0)/a, v(0)/a\omega_0, y(0)/a, w(0)/a\omega_0] = [1.0, 0.0, 1.0, 0.0]$. With these initial conditions, the child swings back and forth while simultaneously bounces up and down.

In contrast, if we take $b = 2.0$, the swing frequency is larger than than the bounce frequency. We find the solution in that case with the statements

```matlab
global B
B=2.0;
ic = [ 1.0; 0.0; 1.0; 0.0 ];
[t, x1] = ode23(@elasticswing, [0.0,30.0], ic);
```

Fig. E11.27 and Fig. E11.28, which are produced by the same statements that produced Fig. E11.23 and Fig. E11.24,\(^8\) show the results with these initial conditions. Compare the frequencies of the $x$ and $y$ motions here with that of the motion shown in Fig. E11.23 and Fig. E11.24 and in Fig. E11.25 and Fig. E11.26.

\(^8\)Only the coordinates of the mark on the origin in Fig. E11.28 were changed to $(0.9,1.0)$ and the axis limits were changed away from the default to `axis([-2.0,2.0,1.0,5.0])`. 
Figure E11.26: The trajectory of the child when $b = 0.5$ and \([x(0)/a, v(0)/a\omega_0, y(0)/a, w(0)/a\omega_0] = [1.0, 0.0, 1.0, 0.0]\). The motion starts at the point marked with a circle at coordinates \((x/a, y/a) = (1.0, 1.0)\).

Figure E11.27: Position and velocity when $b = 2.0$ and \([x(0)/a, v(0)/a\omega_0, y(0)/a, w(0)/a\omega_0] = [1.0, 0.0, 1.0, 0.0]\). With these initial conditions, the child swings back and forth while simultaneously bounces up and down.
Figure E11.28: The trajectory of the child when $b = 2.0$ and $[x(0)/a, v(0)/a\omega_0, y(0)/a, w(0)/a\omega_0] = [1.0, 0.0, 1.0, 0.0]$. The motion starts at the point marked with a circle at coordinates $(x/a, y/a) = (1.0, 1.0)$. 

![y/a vs. x/a graph](image-url)
Chapter 13

Evaluating Integrals

13.18 Quantum Harmonic Oscillator Turning Points (MATLAB)

Exercise: The (normalized) wave functions for a quantum harmonic oscillator in its first and second excited states \((n = 1, n = 2)\) are

\[
\psi_1(x) = \sqrt{2} \left( \frac{m\omega}{\pi \hbar} \right)^{1/4} ye^{-y^2/2} ; \quad \psi_2(x) = \frac{1}{\sqrt{2}} \left( \frac{m\omega}{\pi \hbar} \right)^{1/4} (2y^2 - 1) e^{-y^2/2}
\]

where \(y = x/\sqrt{\hbar/\omega}, \) the energies of these states are \(3\hbar\omega/2, \) and \(5\hbar\omega/2, \) respectively, and the symbols have the same meanings as in Section 13.1.8. Find the probability that a harmonic oscillator in each of these states will be found outside the classical turning point.

Solution: The wave functions for a quantum harmonic oscillator in its first and second excited states are

\[
\psi_1(x) = \sqrt{2} \left( \frac{m\omega}{\pi \hbar} \right)^{1/4} ye^{-y^2/2} ; \quad \psi_2(x) = \frac{1}{\sqrt{2}} \left( \frac{m\omega}{\pi \hbar} \right)^{1/4} (2y^2 - 1) e^{-y^2/2}
\]

where \(y = x/a, \ a = \sqrt{\hbar/\omega}, \) and \(\omega = \sqrt{k/m}. \) The energies of these states are \(3\hbar\omega/2\) and \(5\hbar\omega/2, \) respectively. We know that the classical turning point occurs when the potential energy, \(kx^2/2, \) is equal to the energy of the state, so we find that \(x_{1\text{turn}} = \sqrt{3\hbar\omega/k} = \sqrt{3}a\) and \(x_{2\text{turn}} = \sqrt{5\hbar\omega/k} = \sqrt{5}a.\) Now, to find the probability that the oscillator will be outside the classical turning point, we need to subtract the integral from \(-x_{\text{turn}}\) to \(x_{\text{turn}}\) of the square magnitude of the wave function from the total probability of finding the oscillator somewhere, i.e., from 1. Thus, we conclude that

\[
P_1 = 1 - \int_{x = -\sqrt{3}a}^{x = \sqrt{3}a} |\psi_1(x)|^2 \, dx = 1 - 2 \sqrt{\frac{m\omega}{\pi \hbar}} \int_{y = -\sqrt{3}}^{y = \sqrt{3}} y^2 e^{-y^2} \, d(ay)
\]

\[
= 1 - 2 \sqrt{\frac{ma^2 \omega}{\pi \hbar}} \int_{-\sqrt{3}}^{\sqrt{3}} y^2 e^{-y^2} \, dy = 1 - \frac{2}{\sqrt{\pi}} \int_{-\sqrt{3}}^{\sqrt{3}} y^2 e^{-y^2} \, dy
\]

\[
= 1 - \frac{4}{\sqrt{\pi}} \int_{0}^{\sqrt{3}} y^2 e^{-y^2} \, dy
\]

and

\[
P_2 = 1 - \int_{x = -\sqrt{5}a}^{x = \sqrt{5}a} |\psi_2(x)|^2 \, dx = 1 - \frac{1}{2} \sqrt{\frac{m\omega}{\pi \hbar}} \int_{y = -\sqrt{5}}^{y = \sqrt{5}} (2y^2 - 1)^2 e^{-y^2} \, d(ay)
\]
Exercise 13.18 (MATLAB)

\[
= 1 - \frac{1}{2} \sqrt{\frac{ma^2\omega}{\pi\hbar}} \int_{-\sqrt{\frac{5}{2}}}^{\sqrt{\frac{5}{2}}} (2y^2 - 1)^2 e^{-y^2} dy = 1 - \frac{1}{2\sqrt{\pi}} \int_{-\sqrt{\frac{5}{2}}}^{\sqrt{\frac{5}{2}}} (2y^2 - 1)^2 e^{-y^2} dy
\]

where the last form in each case emerges when we recognize that the integrand is an even function of \( y \) and the limits are symmetric. The necessary function M-files then are:

```matlab
function x=quantumn1(y)
Part1 = 4.0/sqrt(pi);
x = Part1.*y.^2.*exp(-y.^2);

and

function x=quantumn2(y)
Part1 = 1.0/sqrt(pi);
x=Part1.*(2.*y.^2-1).^2.*exp(-y.^2);
```

After these file have been saved in the default directories with the names `quantumn1.m` and `quantumn2.m`, we can evaluate the desired integrals by invoking the function `quad` with the statements

```matlab
>> format long
>> n1 = 1.0 - quad( @quantumn1, 0.0, sqrt(3) )
n1 = 0.11161020740695
>> n1 = 1.0 - quad( @quantumn1, 0.0, sqrt(3), 1.0e-9 )
n1 = 0.11161022510938
>> n2 = 1.0 - quad( @quantumn2, 0.0, sqrt(5) )
n2 = 0.09506939842309
>> n2 = 1.0 - quad( @quantumn2, 0.0, sqrt(5), 1.0e-9 )
n2 = 0.09506943487693
```

We learn that the oscillator will be outside the classical turning point 11.16% of the time when it is in its first excited state and 9.51% of the time when it is in the second excited state.
Exercise 13.19 (MATLAB)

13.19 Maxwell-Boltzmann Distribution (MATLAB)

Exercise: The Maxwell-Boltzmann speed distribution yields the integral

\[ f(v) = 4\pi \left( \frac{m}{2\pi kT} \right)^{\frac{3}{2}} \int_0^v e^{-mv'^2/2kT} v'^2 \, dv' \]

for the fraction of the molecules having speed less than \( v \). Using numerical means, explore this integral as a function of \( v \). Hint: Re-express the integral using \( \sqrt{2kT/m} \) as the unit of velocity.

Solution: The Maxwell-Boltzmann speed distribution gives us the integral

\[ f(v) = 4\pi \left( \frac{m}{2\pi kT} \right)^{\frac{3}{2}} \int_0^v e^{-mv'^2/2kT} v'^2 \, dv' \]

for the fraction of molecules having speed less than \( v \). By letting \( X = \sqrt{v^2 m/2kT} \), so that \( dX = \sqrt{m/kT} \, dv \), we can recast this integral as

\[ f(X) = \frac{4}{\sqrt{\pi}} \int_0^X e^{-X'^2} X'^2 \, dX' \]

To evaluate this integral in MATLAB, we begin by creating the function M-file

```matlab
function y=maxbolt(x)
% The Maxwell-Boltzmann speed distribution
% in dimensionless form, x=v/sqrt(2kT/m).
% y=x.^2.*exp(-x.^2);
```

that will return the integrand. Once the M-file is written and saved in the default directory with the name `maxbolt.m`, we can evaluate the integral by invoking the MATLAB command `quad` with the statements

```matlab
>> N=30;
>> x=linspace(0.0,3.0,N+1);
>> for i =1:N+1
       dat(i)=(4/sqrt(pi))*quad( @maxbolt, 0.0, x(i));
>> end
>> plot(x,dat, 'Color', 'black', 'LineWidth', 4)
>> title('Speed Distribution', 'FontSize', 20)
>> xlabel('X', 'Fontsize', 16)
>> ylabel('f(X)', 'Fontsize', 16)
>> grid on
```

The resulting graph is shown in Fig. E13.1. As expected (since more and more molecules are included as \( v \) increases), \( f(X) \) approaches 1 as \( X = v/\sqrt{m/2kT} \) approaches infinity.
Figure E13.1: Maxwell-Boltzmann speed distribution graph.
13.20 Black Body Radiation (MATLAB)

**Exercise:** Planck’s black body radiation law gives the expression

\[ I(\nu_2, \nu_1) = \frac{8\pi h}{c^3} \int_{\nu_1}^{\nu_2} \frac{\nu^3}{e^{h\nu/kT} - 1} d\nu \]

for the power radiated per unit area in the frequency range \( \nu_1 \leq \nu \leq \nu_2 \). Using numerical means, explore the power radiated in the visible spectrum \( 4 \times 10^{14} \text{ Hz} \leq \nu \leq 7 \times 10^{14} \text{ Hz} \) as a function of temperature. *Hint:* One way to approach this exercise would be to choose a reference frequency \( \nu_0 \) arbitrarily (say \( 10^{14} \text{ Hz} \)) and recast the integral on the dimensionless variable \( s = \nu/\nu_0 \). Examination of \( I \) in units of \( 8\pi h\nu_0^4/c^3 \) as a function of \( T \) in units of \( h\nu_0/k \) would then be indicated.

**Solution:** Planck’s black body radiation law gives us the expression

\[ I = \frac{8\pi h}{c^3} \int_{\nu_1/\nu_0}^{\nu_2/\nu_0} \frac{s^3}{e^{hs/\nu_0/kT} - 1} ds \]

If we express \( I \) in units of \( I_0 = 8\pi h\nu_0^4/c^3 \) as a function of \( T \) in units of \( T_0 = h\nu_0/k \), then we have

\[ I' = \int_{\nu_1/\nu_0}^{\nu_2/\nu_0} \frac{s^3}{e^{s/\nu_0/kT_0 - 1} ds} \]

where \( I' = I/(8\pi h\nu_0^4/c^3) \) and \( T' = T/(h\nu_0/k) \). To explore this integral over the frequency range \( 4 \times 10^{14} \text{ Hz} \leq \nu \leq 7 \times 10^{14} \text{ Hz} \), let us choose the reference frequency to be \( 10^{14} \text{ Hz} \) so that the integral of interest ranges from 4 to 7. Note also that this choice of \( \nu_0 \) implies the reference temperature

\[ T_0 = \frac{h\nu_0}{k} = \frac{6.62 \times 10^{-34} \text{ Js} \times 10^{14} \text{ s}}{1.38 \times 10^{-23} \text{ J/K}} = 4797 \text{ K} \]

Since the temperature of a white-hot tungsten filament is about 2800 K and the average temperature of the sun is 5000-6000 K, the region of interest for this exercise probably is in the region of \( T/T_0 \leq 1 \).

The first step in exploring this integral is to write the function M-file

```matlab
function y = planck(s)
% This M-file returns the integrand for the power
% radiated per unit area according to Planck’s black body
% radiation law.
% 
% global t
y = (s.^3)./(exp(s./t)-1);
```

to return the integrand of the integral of concern.

Then, in MATLAB, we establish the global parameter \( t \), generate values for \( t \) within a reasonable range, and invoke \texttt{quad8} to evaluate the integral for values of \( T' \) ranging from 0.0 to 2.0, an end accomplished with the statements
>> global t
>> inc = 2/99;
>> tt = [0.0:inc:2.0];
>> for i=1:100
    t=tt(i);
    p(i)=quad8(@planck, 4.0, 7.0);
end

At this point, however, we discover that some sort of problem exists, because the attempt to
integrate yields some warnings about dividing by zero. We recall that exponentials have a way
of causing floating point under or overflow. Given the integral we are evaluating, the worst point
in this regard will be at the upper limit, when the integrand involves the exponential $e^{7.0/T'}$. At
$T' = 0$, this exponential will definitely cause difficulties, since it will have the value $e^\infty$—a clear
case of floating point overflow. We must, in fact, avoid evaluating the integrand at temperatures for
which $e^{7.0/T'}$ exceeds about $10^{307}$. We find the critical value by searching with MATLAB, using the
statements

>> exp(7/.01)
   ans = 1.014232054735004e+304
>> exp(7/.0099)
   ans = 1.193730791191693e+307
>> exp(7/.0098)
   ans = Inf

Luckily, the plot function knows to ignore these trouble spots. We plot the functions with the
statements

>> plot( tt,p, 'Color', 'k', 'LineWidth', 4 )
>> title( 'Black Body Radiation Intensity', 'FontSize', 20 )
>> xlabel( 'T/T0', 'FontSize', 16 )
>> ylabel( 'Intensity in the Visible', 'FontSize', 16 )
>> grid on

The resulting graph is shown in Fig. E13.2.
Figure E13.2: Planck’s black body radiation intensity $I' = I/I_0$ as a function of temperature $T' = T/T_0$. 
13.21 Confidence Intervals for Gaussian (MATLAB)

Exercise: As used in statistical data analysis, the Gaussian distribution for a variable $t$ is usually expressed in terms of the standard deviation $\sigma$, the distribution function being

$$\frac{1}{\sqrt{2\pi \sigma}} e^{-t^2/2\sigma^2}$$

Thus, the probability of finding a value between $a$ and $b$ is given by

$$P(a, b) = \frac{1}{\sqrt{2\pi \sigma}} \int_a^b e^{-t^2/(2\sigma^2)} \, dt$$

Show analytically that $P(-x, x) = \text{erf}(x/\sqrt{2\sigma})$, and then evaluate $P(-\sigma, \sigma)$, $P(-2\sigma, 2\sigma)$, and $P(-3\sigma, 3\sigma)$ numerically. The values of these three quantities are 0.6827, 0.9545, and 0.9973, respectively—values that give rise to the designations of 68%, 95%, and 99% confidence intervals in statistical data analysis.

Solution: The probability that a single measurement drawn from a normally distributed universe of possible values will lie between $a$ and $b$ is given by the integral

$$P(a, b) = \frac{1}{\sqrt{2\pi \sigma}} \int_a^b e^{-t^2/(2\sigma^2)} \, dt$$

To show analytically that $P(-x, x) = \text{erf}(x/\sqrt{2\sigma})$, we simply change variables. If we let $t^2/(2\sigma^2) = s^2$, then $dt = \sqrt{2\sigma} \, ds$ and, when $t = x$, $s = x/\sqrt{2\sigma}$. Thus, the integral becomes

$$P(-x, x) = \frac{1}{\sqrt{\pi}} \int_{-x/\sqrt{2\sigma}}^{x/\sqrt{2\sigma}} e^{-s^2} \, ds = \frac{2}{\sqrt{\pi}} \int_0^{x/\sqrt{2\sigma}} e^{-s^2} \, ds = \text{erf} \left( \frac{x}{\sqrt{2\sigma}} \right)$$

The first equality is a result of the change of variables. The second stems from the observations that (1) the integrand is an even function of the integration variable and (2) the limits are symmetric.

To evaluate the error function for different values of $x$, we first create and store the function M-file

```matlab
function y = erf(x)
    y = exp(-x.^2);
end
```

we invoke `quad` in MATLAB and print the desired values with the statements

```matlab
>> q = quad( @erf, 0.0, 1/sqrt(2.0) );
>> qerf = (2/(sqrt(pi))*q)
qerf =  0.68268948481321
>> q = quad( @erf, 0.0, 2/sqrt(2.0) );
>> qerf = (2/(sqrt(pi))*q)
qerf =  0.95449977709292
>> q = quad( @erf, 0.0, 3/sqrt(2.0) );
>> qerf = (2/(sqrt(pi))*q)
qerf =  0.99730019457548
```
We have accepted the default tolerance incorporated in the MATLAB routine.

We conclude that a single value randomly selected from a Gaussian distribution stands a 68.3% chance of falling within one standard deviation of the mean, a 95.4% chance of falling within two standard deviations, and a 99.7% chance of falling within three standard deviations.
13.22 Earth Falling into Sun (MATLAB)

**Exercise:** Suppose some cataclysmic event stops the earth dead in its tracks and, responding to the sun’s gravitational attraction, the earth falls into the sun. Using numerical integration, find the time required for the earth to fall over the middle half of its journey to the sun. Expressed in years, what numerically is the value of this time for the earth-sun system? 

**Hint:** Since the gravitational potential is $-GmM/x$, conservation of energy yields

$$\frac{1}{2}m\left(\frac{dx}{dt}\right)^2 - G\frac{mM}{x} = -G\frac{mM}{x_0} \quad \Rightarrow \quad \frac{dx}{dt} = \sqrt{2GM\left(\frac{1}{x} - \frac{1}{x_0}\right)}$$

(The negative square root is taken because $x$, the distance to the sun, is known to be decreasing.)

This expression then leads to the value

$$T_{\text{midhalf}} = \frac{1}{\sqrt{2GM}} \int_{3x_0/4}^{x_0/4} \left(\frac{1}{x} - \frac{1}{x_0}\right)^{-1/2} dx$$

**Hint:** The evaluation will be simpler if you begin by recasting the problem in dimensionless terms, expressing lengths in units of $x_0$ and times in units of $\sqrt{x_0^3/(2GM)}$. To interpret the significance of this unit of time, determine the period of a circular orbit at radius $x_0$, which will turn out to be $2\pi\sqrt{x_0^3/GM}$. For the earth around the sun, this latter time is, of course, 1 year. Optional: See if you can develop a means to determine the time required for the first half of the journey, which unfortunately—for numerical approaches—involves a convergent but improper integral.

**Solution:** Should the earth stop, the time it would take for it to fall over the middle half of its journey into the sun is given by the integral

$$T_{\text{midhalf}} = \frac{1}{\sqrt{2GM}} \int_{x_0/4}^{3x_0/4} \left(\frac{1}{x} - \frac{1}{x_0}\right)^{-1/2} dx$$

To simplify the evaluation, let us introduce the dimensionless variable $\alpha = x/x_0$, so that the integral becomes

$$T_{\text{midhalf}} = \frac{1}{\sqrt{2GM}} \int_{1/4}^{3/4} \left(\frac{1}{\alpha} - 1\right)^{-1/2} d\alpha$$

To find a convenient time unit, let’s determine the period of a circular orbit at radius $x_0$. We first require that the gravitational force match the required centripetal force, i.e., that

$$\frac{mv^2}{x_0} = G\frac{mM}{x_0^2} \quad \Rightarrow \quad v = \sqrt{\frac{GM}{x_0}}$$

and then compute the period from the relationship

$$T_0 = \frac{2\pi x_0}{v} = \frac{2\pi x_0}{\sqrt{GM/x_0}} = \frac{2\pi x_0^{3/2}}{\sqrt{GM}}$$

We conclude that, in this unit (where—for the earth around the sun—$T_0$ is one year),

$$\frac{T_{\text{midhalf}}}{T_0} = \frac{1}{2\pi \sqrt{2}} \int_{1/4}^{3/4} \left(\frac{1}{\alpha} - 1\right)^{-1/2} d\alpha = \frac{1}{2\pi \sqrt{2}} \int_{1/4}^{3/4} \sqrt{\frac{\alpha}{1 - \alpha}} d\alpha$$

Note that the lower limit corresponds to the final position of the earth and the upper limit corresponds to the initial position of the earth.

To use MATLAB to evaluate this integral, we must first create an appropriate M-file to define the integrand, say the file
function y = earthfall(alpha)
    y = (alpha./((1-alpha)).^(0.5))/(2*pi*sqrt(2));

We store this file with the name `earthfall.m` in the default directory. Then we invoke `quad` with the following statement

>> format long
>> quad( @earthfall, 0.25, 0.75 )
    ans = 0.05892560976393

which, given the default (fractional) tolerance of $10^{-6}$, we conclude is accurate to at least five digits after the decimal point. Recognizing that the time unit is one year for the earth, we conclude that the earth would take 0.058926 years (about 21.51 days or one month) to fall over the middle half of its journey to the sun.

While we have the routine in hand, let’s note that the time required to fall over the last half of its journey (from $\alpha = 0.5$ to $\alpha = 0.0$) would be

>> quad( @earthfall, 0.0, 0.5 )
    ans = 0.03211223615604

or about 11.8 days. The time required for the earth to fall over the last 90% of its journey would be

>> quad( @earthfall, 0.0, 0.9 )
    ans = 0.10679987881832

or about 38.98 55 days. Because the integrand diverges as $\alpha \to 1$, we cannot reliably use this approach to find the time required for the earth’s entire journey to the sun. The integral is not a divergent integral, but the divergence of the integrand implies difficulties in obtaining a reliable evaluation over intervals that extend too close to the starting point. We can, however, push the upper limit fairly close to 1.0 before IDL complains about non-convergence. Despite that divergence, MATLAB actually will give a value if asked to integrate over the interval $0 \leq \alpha \leq 1$, specifically

>> quad( @earthfall, 0.0, 1.0 )
Warning: Divide by zero.
(Type "warning off MATLAB:divideByZero" to suppress this warning.)
    ans = 0.17678035619162

which—to the extent we can believe the result—is 64.52 days. Should this unfortunate event occur, we certainly don’t have very much time to do anything about it. (These results can be compared with the analytic results obtained in Exercise 11.6.)
Exercise 13.23 (MATLAB)

13.23 Properties of Lorentz Distribution (MATLAB)

Exercise: The normalized Lorentz distribution function is given by

\[ p(x) = \frac{1}{\pi} \frac{a/2}{x^2 + (a/2)^2} \]

Find the probability that a single, randomly selected value will be in the range \(-a \leq x \leq a\). Make sure to assess the precision of your result by methods that do not exploit a priori knowledge of the exact value. Hint: Before evaluating the integral, introduce the dimensionless variable \( s = x/a \) and note that the result actually doesn’t depend on \( a \), so there is but one number to determine.

Solution: The normalized Lorentz distribution function is

\[ p(x) = \frac{1}{\pi} \frac{a/2}{x^2 + (a/2)^2} \]

To find the probability that a single, randomly chosen value will be in the range \(-a \leq x \leq a\), we must integrate \( p(x)dx \) from \(-a\) to \(a\). First, though, let us express \( x \) in a dimensionless form by introducing the variable \( s = x/a \). Since \( ds = dx/a \), the new form of the integral is

\[ \int p(s)ds = \int_{-1}^{1} \frac{1}{2\pi} \frac{1}{s^2 + 1/4} \, ds \]

To invoke \texttt{quad} in MATLAB to evaluate the integral, we must write the function pro-file

\begin{verbatim}
function y=lorentz(s)
    \%
    \% This program generates the integrand for
    \% the normalized Lorentz distribution.
    \%
    tmp = 1./(s.^2+0.25);
    y = tmp/(2*pi);
\end{verbatim}

to return the integrand. Once this program has been saved in a file named \texttt{lorentz.m}, we can evaluate the integral in MATLAB with the statements

\begin{verbatim}
>> format long
>> q = quad( @lorentz, -1.0, 1.0, 0.0001 )
    q = 0.70487290046758
>> q = quad( @lorentz, -1.0, 1.0, 0.00001 )
    q = 0.70482493736991
>> q = quad( @lorentz, -1.0, 1.0, 0.000001 )
    q = 0.70482483366342
>> q = quad( @lorentz, -1.0, 1.0, 0.0000001 )
    q = 0.70483276302819
>> q = quad( @lorentz, -1.0, 1.0, 0.00000001 )
    q = 0.70483276471738
>> q = quad( @lorentz, -1.0, 1.0, 0.000000001 )
    q = 0.70483276469201
>> q = quad( @lorentz, -1.0, 1.0, 0.0000000001 )
    q = 0.70483276469905
>> q = quad( @lorentz, -1.0, 1.0, 0.00000000001 )
\end{verbatim}
By invoking \texttt{quad} with different fractional accuracies, we learn a bit about the convergence. Since the last four values agree to at thirteen digits, we conclude that, to thirteen digits, the value of the integral of interest is 0.7048327646991, though we might want to be just a bit cautious about the last of these digits. Internal roundoff in double precision arithmetic may be affecting the last couple of displayed digits.
13.24 Electron inside Bohr Radius (MATLAB)

**Exercise:** According to the quantum theory, the probability that the electron in the ground state of the hydrogen atom will be found between the center of the atom and some radius \( r \) is given by

\[
P(r) = \frac{4}{a^3} \int_0^r e^{-2r'/a} r'^2 \, dr' = 4 \int_0^{r/a} e^{-2\rho} \rho^2 \, d\rho
\]

where \( a \) is the Bohr radius and \( \rho = r'/a \). Using numerical integration, evaluate this integral as a function of its upper limit. Then plot and comment on a graph of \( P(r) \) versus \( r/a \).

**Solution:** We seek to evaluate the integral\(^1\)

\[
P(x) = 4 \int_0^x e^{-2y} y^2 \, dy
\]
as a function of \( x \). To use MATLAB, we begin by creating the M-file

```
function f = integ_ex24( y )
f = 4.0.*y.^2.*exp(-2*y);
```
to define the integrand for the integration routine. Then, we store this file in the default directory with the name `integ_ex24.m`. With this M-file in place, the statements

```
>> format long
>> p = quadl( @integ_ex24, 0.0, 1.0 )
p = 0.32332358614222
```

will evaluate the integral over the interval \( 0 \leq y \leq 1.0 \) using the default absolute tolerance \( 10^{-6} \) and display the result.

Now, we need to evaluate the integral as a function of its upper limit in a form that will allow us then to plot the required graph. We begin by deciding to evaluate \( P(x) \) over the interval \( 0 \leq x \leq 4 \) and to evaluate it at values of \( x \) separated by 0.1—a total of 41 points. Thus, we would create a vector of upper limits with the statements

```
x = [0.0 : 0.1 : 4.0 ]
```

Then, in a loop, we evaluate the integral for each of the chosen upper limits and store each new value in the next element of the vector \( p \). The single statement accomplishing these operations is

```
for i = 1:41 p(i) = quadl( @integ_ex24, 0.0, x(i) ); end;
```
The execution prints out the message

**Warning:** Minimum step size reached; singularity possible.

once somewhere during the evaluation, but we need not in this case worry about that message.\(^2\)

Finally, we can plot the requested graph with the statement

```
>> plot( x, p, 'LineWidth', 4, 'Color', 'black' )
```

\(^1\)For a simpler notation, we replace the integration variable with \( y \) and \( r/a \) with \( x \).

\(^2\)Repeating the calculation starting the loop at \( i=2 \) will not produce this message. Evidently, then, the message comes from the very first integral, at which point the upper limit is 0 and the value of the integral is zero.
The resulting graph is shown below. As we would expect, the probability of finding the electron inside the radius $r$ increases towards one as the selected radius gets larger.
Exercise 13.25 (MATLAB)

Elliptic Integrals (MATLAB)

Exercise: The complete elliptic integrals of the first and second kinds are given by

\[
K(k) = \int_0^{\pi/2} \frac{d\phi}{(1 - k^2 \sin^2 \phi)^{1/2}} \quad ; \quad E(k) = \int_0^{\pi/2} (1 - k^2 \sin^2 \phi)^{1/2} d\phi
\]

Explore these integrals as functions of the modulus \( k \). As part of your exploration, obtain a graph of the period \( T \) of a simple pendulum as a function of the amplitude \( \alpha \) of that pendulum. Analytically, the period of that pendulum is given as a function of \( \alpha \) by \( T/T_0 = (2/\pi)K(\sin(\alpha/2)) \), where \( T_0 \) is the period of the pendulum at small amplitude.

Solution: The complete elliptic integrals of the first and second kinds are defined by the expressions

\[
K(k) = \int_0^{\pi/2} \frac{1}{(1 - k^2 \sin^2 \phi)^{1/2}} d\phi \quad ; \quad E(k) = \int_0^{\pi/2} (1 - k^2 \sin^2 \phi)^{1/2} d\phi
\]

To explore these equations as functions of the modulus \( k \), we write function M-files to define the integrands, making sure to include the common parameter \( k \). For the elliptic integral of the first kind, we write the file `ellipone.m` containing the statements

```matlab
function x=ellipone(phi)
% This function returns the elliptic integral of the first kind
% global k
tmp=sqrt(1-k^2*(sin(phi)).^2);
x=1./tmp;
```

We can then evaluate the integral in MATLAB using the statements

```matlab
>> global k
dk=1.0/100.0;
k=[0.0:dk:1.0];
k(101)=0.995; k(102)=0.997; k(103)=0.999;
>> for i=1:103
    k=kk(i);
    valo(i)=quad( @ellipone, 0.0, pi/2 );
end
>> plot(kk, valo, 'Color', 'black', 'LineWidth', 4)
axis( [0.0 1.0 1.0 4.0] )
grid on
title('Elliptic Integral of the First Kind', 'FontSize', 20)
xlabel('k', 'FontSize', 16)
ylabel('K(k)', 'FontSize', 16)
```

Here, we have recognized that values of \( k \) close to 1 are important and have constructed \( kk \) to avoid the value 1 itself but to sneak up on it quite closely.

The resulting graph is shown in Fig. E13.3.

For the elliptic integral of the second kind, we first create the M-file `elliptwo.m`, containing the statements
function x=elliptwo(phi)
% This function returns the elliptic integral of the second kind
% global k
x=sqrt(1-k^2*(sin(phi)).^2);

and then, remembering that we have already in the above created a vector with appropriate values of k, we execute the statements

```
>> for i=1:103
    k=kk(i);
    valt(i)=quad( @elliptwo, 0.0, pi/2);
end
>> plot(kk, valt, 'Color', 'black', 'LineWidth', 4)
>> axis( [0.0 1.0 0.8 1.7] )
>> grid on
>> title('Elliptic Integral of the Second Kind', 'FontSize', 20)
>> xlabel('k', 'FontSize', 16)
>> ylabel('E(k)', 'FontSize', 16)
```

The resulting graph is shown in Fig. E13.4.

For the third part of this exercise, we wish to graph the period of a simple pendulum as a function of the amplitude of the pendulum. The period is given by

$$\frac{T}{T_0} = \frac{2}{\pi} K(\sin(\alpha/2))$$

Figure E13.3: The first elliptic integral, $K(k)$. 

<table>
<thead>
<tr>
<th>Elliptic Integral of the First Kind</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
</tr>
<tr>
<td>3.5</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>2.5</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>1.5</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>0.5</td>
</tr>
<tr>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>k</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
</tr>
<tr>
<td>0.1</td>
</tr>
<tr>
<td>0.2</td>
</tr>
<tr>
<td>0.3</td>
</tr>
<tr>
<td>0.4</td>
</tr>
<tr>
<td>0.5</td>
</tr>
<tr>
<td>0.6</td>
</tr>
<tr>
<td>0.7</td>
</tr>
<tr>
<td>0.8</td>
</tr>
<tr>
<td>0.9</td>
</tr>
<tr>
<td>1</td>
</tr>
</tbody>
</table>
where $T_0$ is the period of the pendulum at small amplitudes and $\alpha$ is the amplitude. We already have two vectors giving $K(k)$ and $k$. We need merely determine a vector $a$ giving the values of the amplitude $a$ associated with each value of $k$, find the period $T/T_0$ associated with each amplitude by multiplying the elliptic integral by $2/\pi$, and plot $T/T_0$ versus $a$. We elect, however, to plot the amplitude in units of $\pi$. We invoke the statements

```matlab
>> a = 2.*asin(kk)/pi;
>> T = (2.0/pi).*valo;
>> plot(a, T, 'Color', 'black', 'LineWidth', 4)
>> axis([0.0, 1.0, 0.0,3.5])
>> grid on
>> title('Period as a Function of Amplitude', 'FontSize', 20)
>> xlabel('Amplitude/\pi', 'FontSize', 16)
>> ylabel('T/T_0', 'FontSize', 16)
```

The resulting graph is shown in Fig. E13.5. Note that, as $\alpha$ approaches $\pi$, the graph of the period approaches infinity. As $\alpha$ approaches 0, the period (in the form $T/T_0$) approaches 1.
Figure E13.5: Period as a function of amplitude.

Period as a Function of Amplitude

T/T₀

Amplitude/π
13.26 Large Amplitude Simple Pendulum (MATLAB)

**Exercise:** The angular position $\theta(t)$ of a simple pendulum swinging with amplitude $\alpha$ is given by the integral

$$\omega t = \int_0^\beta \frac{d\phi}{(1 - k^2 \sin^2 \phi)^{1/2}}$$

where, with $l$ the length of the pendulum and $g$ the acceleration of gravity, $\omega = \sqrt{g/l}$, $k = \sin(\alpha/2)$, and $\beta = \sin^{-1}((\sin(\theta/2))/k)$. Remember that, because of the choice of signs (see Section 13.1.4), this integral is valid only during the portion of the swing from $\theta = 0$ to $\theta = \alpha$. Obtain graphs of $\theta$ versus $\omega t$ over the first quarter of the pendulum’s swing for several different values of $\alpha$.

**Solution:** In this exercise, we are concerned only with the first quarter of the swing of the pendulum, during which $0 \leq \theta \leq \alpha$. Further, the integral gives $t$ when we know $\theta$, which appears—somewhat obscurely—in the upper limit. Normally, the integral gives the position when we know the time, so we must here use the integral to find $t$ as a function of $\theta$ and then plot $\theta$ versus $t$. MATLAB can deal with this variable in the same way that it deals with ordinary numbers since, no matter what the value of $\beta$ actually is, it will always still be a number, $-1 \leq \beta \leq 1$. Therefore, we can make $\beta$ an array and fill that array with numbers that we want to test. The rest of the problem is straightforward and utilizes `quad` to call the procedure.

```matlab
function y = pend(phi, k)
    y = 1.0./((1.0 - k.^2.*sin(phi).^2).^0.5);
end
```

The parameter $k$ must be communicated to the function via a global variable. We must, of course, execute these statements before the procedure is first called.

This function is then used inside the integration routine with the statements

```matlab
alphadeg=[10.0,45.0,90.0,120.0,150.0,178.0]; % Set chosen amplitudes
alpha = alphadeg*pi/180;
kall = sin(alpha/2); % Set k for each amplitude
stpsize = 1.0; % Set interval between plotted points

Then, we evaluate and $\omega t$ as a function of $\theta$ and plot $\theta$ versus $\omega t$ for the first amplitude with the statements

```matlab
thetadeg = [ 0 : stpsize : alphadeg(1) ]; % Set values for theta
theta = thetadeg*pi/180.0;
k = kall(1); % Select value of k
beta = asin(sin(theta/2)/k); % Calculate corresponding beta
q = size(beta);
omega_t=zeros(1,q(2)); % Create array for omega t’s
% Evaluate integral for each beta and plot graph
for i = 1:q(2) omega_t(i) = quad( @pend, 0.0, beta(i), [] , [], k ); end
plot( omega_t,thetadeg, 'color','black', 'linewidth',2 )
q= size(omega_t);
T(1) = omega_t(q(2)); % Save period.
```

Finally, we generate similar graphs for the remaining amplitudes with the statements.
hold on
for j = 2:6
    thetadeg = [ 0 : stpsize : alphadeg(j) ];
    theta = thetadeg*pi/180.0;
    k = kall(j);
    beta = asin(sin(theta/2)/k);
    q = size(beta);
    omega_t=zeros(1,q(2));
    for i = 1:q(2) omega_t(i) = quad( @pend, 0.0, beta(i), [], [], k ); end
    plot( omega_t,thetadeg, 'color','black', 'linewidth',2 )
    q = size(omega_t);
    T(j) = omega_t(q(2));
end

Finally, we label the graph, including labels on the axes, and we add a horizontal line at \( \theta = 180 \) with the statements

```
set(gca, 'fontsize',16)
xlabel('\omega t', 'fontsize', 20)
ylabel('\theta(t)', 'fontsize',20 )
title('First quarter of cycle for values of \alpha', 'fontsize',20)
```

The graph shown in Fig. E13.6 shows the final result of the above code.

Note in particular that, the larger the amplitude the longer the period. More specifically, we can reveal the periods for the amplitudes we chose with the statements

```
T = 4.0*T % Evaluate actual period
for i = 1:6
    astr = num2str(alphadeg(i)); tstr = num2str(T(i) );
    disp(['For amplitude ', astr, ' degrees the period in the above units is ', tstr] )
end
```

For amplitude 10 degrees the period in the above units is 6.2952
For amplitude 45 degrees the period in the above units is 6.5343
For amplitude 90 degrees the period in the above units is 7.4163
For amplitude 120 degrees the period in the above units is 8.6261
For amplitude 150 degrees the period in the above units is 11.0723
For amplitude 178 degrees the period in the above units is 21.7397

For small amplitude, both \( \alpha \) and \( \theta \) approach zero, so the ratio approaches 1 and \( \beta = \sin^{-1}(1) = \pi/2 \) and

\[
\omega t = \int_0^{\pi/2} d\phi = \frac{\pi}{2}
\]

so \( 4\omega t = 2\pi \) and the small amplitude period in the units used above is \( 2\pi = 6.2832 \). Thus, relative to the small amplitude period,
Figure E13.6: Graphs of angular position $\theta(t)$ versus $\omega t$ for several values of $\alpha$.

Amplitude 10 degrees the period relative to small amplitude is 1.0019
Amplitude 45 degrees the period relative to small amplitude is 1.0400
Amplitude 90 degrees the period relative to small amplitude is 1.1803
Amplitude 120 degrees the period relative to small amplitude is 1.3729
Amplitude 150 degrees the period relative to small amplitude is 1.7622
Amplitude 178 degrees the period relative to small amplitude is 3.4600

Even at 45° the small angle approximation to the period of a real pendulum is off by only 4%! 
13.27 N-th Order Bessel Functions (MATLAB)

Exercise: The n-th order Bessel function can be defined by the integral

\[ J_n(x) = \frac{1}{\pi} \int_0^\pi \cos(n\theta - x \sin \theta) \, d\theta \]

By evaluating this integral numerically as a function of \( x \) for different values of \( n \), obtain graphs of \( J_0(x) \), \( J_1(x) \), and \( J_2(x) \) over the range \( 0 \leq x \leq 10 \).

Solution: The n-th order Bessel function is defined by the integral

\[ J_n(x) = \frac{1}{\pi} \int_0^\pi \cos(n\theta - x \sin \theta) \, d\theta \]

To obtain graphs of \( J_0(x) \), \( J_1(x) \), and \( J_2(x) \) over the range \( 0 \leq x \leq 10 \), we must define an m-file with two parameters, \( x \) and \( n \). The primary independent variable is, of course, the integration variable, but we can add the extra parameters as arguments to the function. Thus, a possible m-file to return the integrand might be

```matlab
function derivs = nbessel( theta, n, x )
derivs = cos(n*theta-x*sin(theta))/pi
```

which we store in the default directory with the name `nbessel.m`. Then, to evaluate the function \( J_0(x) \) over the specified interval, we would use the coding

```matlab
>> x = [0.0 : 0.05 : 10.0 ];  % Set independent variables.
>> n = 0;                    % Set value of n.
>> for i=1:201               % Evaluate J0 for each x.
    J0(i) = quad( @nbessel, 0.0, pi, ...  % J0(i) = quad( ... end
                  [], [], n, x(i) );
```

Similar operations defining \( J_1 \) and \( J_2 \) and setting \( n = 1 \) and \( n = 2 \) at the beginning will yield values for the remaining two Bessel functions in \( J_1 \) and \( J_2 \). Then we produce the graph of Fig. E13.7 with the statements

```matlab
>> plot( x,J0, 'Color','black', 'LineWidth',3 )
>> hold on
>> plot( x,J1, 'Color','black', 'LineWidth',3, 'LineStyle','--' )
>> plot( x,J2, 'Color','black', 'LineWidth',3, 'LineStyle','-.' )
```
Figure E13.7: The Bessel function for $J_0(x)$ (solid line), $J_1(x)$ (dashed line), and $J_2(x)$ (dash-dot line).
13.28 The Bessel Function \( J_1(x) \) (MATLAB)

**Exercise:** The Bessel function \( J_1(x) \) can be defined by the integral

\[
\frac{1}{x} J_1(x) = \frac{2}{\pi} \int_0^1 (1 - u^2)^{1/2} \cos(xu) \, du
\]

Using this definition, obtain a graph of \( J_1(x) \) versus \( x \) over the range \( 0 \leq x \leq 10 \).

**Solution:** The solution to this exercise is obtained in relatively the same manner as other numerical integrations in MATLAB, except that the independent variable \( x \) is a parameter in the integrand, not embedded in one of the limits of integration. To sneak that variable into the integrand, we must utilize MATLAB’s global variables. The function

```matlab
function y = bessel(u, tempx)
    y = sqrt(1-u.^2).*cos(tempx*u);
end
```

provides the integrand. This coding must, of course, be executed before the function is first invoked. We then invoke the coding

```matlab
x = [0:0.1:10];
J = zeros(1,101);
for i = 1:101
    tempx = x(i);
    J(i) = (2.0*x(i)/pi)*quad( @bessel, 0.0, 1.0, [], [], tempx );
end
plot(x,J, 'color','black', 'linewidth',3 )
grid on
set(gca, 'fontsize', 14)
title( 'Bessel Function J_1(x)', 'fontsize',20 )
xlabel( 'x', 'fontsize',16 ); ylabel( 'J_1(x)', 'fontsize',16 )
```

to evaluate the integrals and plot the graph, which is shown in Fig. E13.8.
Figure E13.8: A plot showing the Bessel function $J_1(x)$ for $0 \leq x \leq 10$. 

Bessel Function $J_1(x)$

The Bessel function $J_1(x)$ is plotted for $x$ values between 0 and 10. The graph shows the oscillatory behavior typical of Bessel functions, with peaks and valleys at various intervals.
13.29 Off-Axis Potential of Circular Ring (MATLAB)

**Exercise:** A circular ring of radius $a$ resides in the $xy$ plane with its center at the origin and carries a charge $Q$ uniformly distributed about its perimeter. The electrostatic potential established by this ring at an observation point whose cylindrical coordinates are $(r, \phi, z)$ is

$$
\frac{V(r, \phi, z)}{Q/4\pi \varepsilon_0 a} = \frac{1}{\pi} \int_0^{\pi} \left( 1 - 2 \frac{r}{a} \cos \phi' + \frac{r^2}{a^2} + \frac{z^2}{a^2} \right)^{-1/2} d\phi'
$$

Explore this integral as a function of $r/a$ for several values of $z/a$.

**Solution:** A circular ring, radius $a$, lies in the $xy$ plane with its center at the origin. It carries a total charge $Q$ uniformly distributed around its circumference, thereby producing an electric field and an electrostatic potential in its vicinity. The electrostatic potential at the point $(r, \phi, z)$ (cylindrical coordinates) is given by the equation

$$
\frac{V(r, \phi, z)}{Q/4\pi \varepsilon_0 a} = \frac{1}{\pi} \int_0^{\pi} \left( 1 - 2 \frac{r}{a} \cos \phi' + \frac{r^2}{a^2} + \frac{z^2}{a^2} \right)^{-1/2} d\phi'
$$

Introducing the dimensionless quantities $R = r/a$ and $Z = z/a$, we can simplify this expression to

$$
\frac{V(r, \phi, z)}{Q/4\pi \varepsilon_0 a} = \frac{1}{\pi} \int_0^{\pi} \left( 1 - 2R \cos \phi' + R^2 + Z^2 \right)^{-1/2} d\phi'
$$

We want to use MATLAB to explore this electrostatic potential as a function of $R$ for various values of $Z$. Choosing to measure $V$ in units of $V_0 = Q/4\pi \varepsilon_0 a$ and using the MATLAB global variables $R$ and $Z$ for the dimensionless variables $R$ and $Z$, we first create the function M-file

```matlab
function z = chgdring(phi)
    global R Z
    den= sqrt( 1 - 2*R*cos(phi) + R^2 + Z^2 );
    z = 1.0./(den*pi);
end
```

The parameters $R$ and $Z$, which appear embedded within the integrand, are provided through a declaration of global parameters. The parameters $R$ and $Z$ must be declared global at the command line and assigned values prior to invoking `chgdring` which uses those global values.

After storing the above file with the name `chgdring.m` in the default directory, we are ready to invoke `quad` to evaluate the integral. We would, for example, determine and plot the $x$ and $z$ components of the field at $z = 0.1$ with the statements

```matlab
>> global R Z
>> dr = 1/40.0;
>> r = [0.0: dr : 3.0];
>> N=121;
>> V01 = [1.0: 1.0: N];
>> Z= 0.1;
>> for i =1:N
    R =r(i); V01(i) = quad(@chgdring, 0.0, pi);
  end
>> plot(r, V01)
```
Exercise 13.29 (MATLAB)

(It took a bit of experimenting to find a suitable range of values over which to extend the variable \(r\).) Note that \(N\) is the number of elements in \(r\). The results are incorporated in the more complete graph in Fig. E13.9.

Similar statements generate the electrostatic potential for other values of \(z\). In particular, we execute the statements

```matlab
>> V03 = [1.0: 1.0: N];
>> Z= 0.3;
>> for i =1:N
    R =r(i); V03(i) = quad(@chgdring, 0.0, pi);
end
>> V06 = [1.0: 1.0: N];
>> Z=0.6;
>> for i =1:N
    R =r(i); V06(i) = quad(@chgdring, 0.0, pi);
end
>> V10 = [1.0: 1.0: N];
>> Z=1.0;
>> for i =1:N
    R =r(i); V10(i) = quad(@chgdring, 0.0, pi);
end
```

Finally, we generate a composite plot showing the \(x\) component of the magnetic field produced by this source by invoking the statements

```matlab
>> plot(r, V01, 'k')
>> hold on
>> plot(r, V03, 'k:');
>> plot(r, V06, 'k--');
>> plot(r, V10, 'k-.');
>> axis([0.0,3.0,0.0,1.5])
>> title('Electrostatic Potential of a Charged Ring', 'FontSize', 18)
>> xlabel('r/a', 'FontSize', 14)
>> ylabel('V/V_0', 'FontSize', 14)
>> plot([1.5,2.1], [1.4,1.4], 'LineStyle', '-', 'Color', 'Black')
>> text(2.2, 1.4, 'z/a = 0.1', 'FontSize', 14)
>> plot([1.5,2.1], [1.3,1.3], 'LineStyle', ':', 'Color', 'Black')
>> text(2.2, 1.3, 'z/a = 0.3', 'FontSize', 14)
>> plot([1.5,2.1], [1.2,1.2], 'LineStyle', '--', 'Color', 'Black')
>> text(2.2, 1.2, 'z/a = 0.6', 'FontSize', 14)
>> plot([1.5,2.1], [1.1,1.1], 'LineStyle', '-.', 'Color', 'Black')
>> text(2.2, 1.1, 'z/a = 1.0', 'FontSize', 14)
>> hold off
```

The resulting graph is shown in Fig. E13.9. Note that, since \(x/a = 1\) corresponds to the point directly above the loop, the peaks in these graphs occur right over the loop.
Figure E13.9: The electrostatic potential of a charged ring in various planes parallel to the plane of the ring.
13.30 Off-Axis Field of Current Loop (MATLAB)

**Exercise:** A circular current loop of radius $a$ lies in the $xy$-plane with its center at the origin and carries a current $I'$ counterclockwise as viewed from a point on the positive $z$ axis. The magnetic field at a point in the $xz$ plane is given by

$$ \mathbf{B}(x, z) = \frac{\mu_0 I'}{2\pi a} \int_0^\pi \frac{z \cos \phi' \hat{i} + (a - x \cos \phi') \hat{k}}{[x^2 + z^2 + a^2 - 2ax \cos \phi']^{3/2}} \, d\phi' $$

Explore both components of this magnetic field numerically as functions of $x/a$ for various values of $z/a$, including $z/a = 0$ (which will require some creativity for dealing with the point $x/a = 1.0$, at which the integrand diverges at one point in the range of the integration variable).

**Solution:** A circular current loop, radius $a$, lies in the $xy$ plane with its center at the origin. It carries a current $I'$ counterclockwise as viewed from the positive $z$ axis, thereby producing a magnetic field $\mathbf{B}$. This field, at a point in the $xz$ plane, is given by the equation

$$ \mathbf{B}(x, z) = \frac{\mu_0 I'}{2\pi a} \int_0^\pi \frac{z \cos \phi' \hat{i} + (a - x \cos \phi') \hat{k}}{[x^2 + z^2 + a^2 - 2ax \cos \phi']^{3/2}} \, d\phi' $$

Introducing the dimensionless quantities $X = x/a$ and $Z = z/a$, we can simplify this expression to

$$ \mathbf{B}(X, Z) = \frac{\mu_0 I'}{2\pi a} \int_0^\pi \frac{Z \cos \phi' \hat{i} + (1 - X \cos \phi') \hat{k}}{[X^2 + Z^2 + 1 - 2X \cos \phi']^{3/2}} \, d\phi' $$

We want to use MATLAB to explore both components of this field as functions of $X$ for various values of $Z$. Choosing to measure $B$ in units of $B_0 = \mu_0 I'/2\pi a$ and using the MATLAB variables $x$ and $z$ for the dimensionless variables $X$ and $Z$, we first create the function M-files

```
function y = magnetixx(phi)
    ...
    den = (xx.^2+zz.^2+1-2*xx*cos(phi)).^1.5;
    y = zz.*cos(phi)./den;
```

and

```
function y = magneticz(phi)
    ...
    den = (xx.^2+zz.^2+1-2*xx*cos(phi)).^1.5;
    y = (1-xx.*cos(phi))./den;
```

In each case, the parameters $x$ and $z$, which appear embedded within the integrands, are provided through the global variables `xx` and `zz`. These functions, of course, return the $x$ and $z$ components of the integrand, respectively.

Now, we are ready to invoke `quad` to evaluate the integrals. We would, for example, determine and plot the $x$ and $z$ components of the field at $z = 0.1$ with the statements
>> global xx zz
>> inc = 10/100;
>> x = [ -5:inc:5 ];
>> zz = 0.1;
>> for i = 1:101
    xx = x(i);
    Bx01(i) = quad( @magneticx, 0.0, pi );
    Bz01(i) = quad( @magneticz, 0.0, pi );
end
>> subplot(1,2,1)
>> plot( x, Bx01, 'Color', 'k', 'LineWidth', 2 )
>> title( 'B_x', 'FontSize', 20 )
>> subplot(1,2,2)
>> plot( x, Bz01, 'Color', 'k', 'LineWidth', 2 )
>> title( 'B_z', 'FontSize', 20 )

(It took a bit of experimenting to find a suitable range of values over which to extend the variable \( x \).) The results are incorporated in the more complete graphs of Fig. E13.10 and Fig. E13.11.

Similar statements generate the magnetic fields for other values of \( z \). In particular, we execute the statements

>> zz = 0.3;
>> for i = 1:101
    xx = x(i);
    Bx03(i) = quad( @magneticx, 0.0, pi );
    Bz03(i) = quad( @magneticz, 0.0, pi );
end
>> zz = 0.6;
>> for i = 1:101
    xx = x(i);
    Bx06(i) = quad( @magneticx, 0.0, pi );
    Bz06(i) = quad( @magneticz, 0.0, pi );
end
>> zz = 1.0;
>> for i = 1:101
    xx = x(i);
    Bx10(i) = quad( @magneticx, 0.0, pi );
    Bz10(i) = quad( @magneticz, 0.0, pi );
end

Finally, we generate a composite plot showing the \( x \) component of the magnetic field produced by this source by invoking the statements

>> subplot(2,2,1)
>> plot( x, Bx01 )
>> subplot(2,2,2)
>> plot( x, Bx03 )
>> subplot(2,2,3)
>> plot( x, Bx06 )
>> subplot(2,2,4)
>> plot( x, Bx10 )
>> subplot(1,1,1)
>> plot( x, Bx01, 'Color', 'k', 'LineWidth', 2 )
Exercise 13.30 (MATLAB)

Figure E13.10: The $x$ component of the magnetic field for various values of $z$.

```
>> title('x Component of Magnetic Field', 'FontSize', 20)
>> xlabel('x/a', 'FontSize', 16)
>> ylabel('B_x/B_0', 'FontSize', 16)
>> axis([0.0 3.0 0.0 10.0])
>> hold on
>> plot(x, Bx03, 'Color', 'k', 'LineWidth', 2, 'LineStyle', '-. ')
>> plot(x, Bx06, 'Color', 'k', 'LineWidth', 2, 'LineStyle', '-- ')
>> plot(x, Bx10, 'Color', 'k', 'LineWidth', 2, 'LineStyle', ':' )
>> plot([1.5, 1.7], [9.0, 9.0], 'Color', 'k')
>> text(1.8, 9.0, 'z/a=0.1', 'FontSize', 16)
>> plot([1.5, 1.7], [8.0, 8.0], 'Color', 'k', 'LineWidth', 2, 'LineStyle', '-. ')
>> text(1.8, 8.0, 'z/a=0.3', 'FontSize', 16)
>> plot([1.5, 1.7], [7.0, 7.0], 'Color', 'k', 'LineWidth', 2, 'LineStyle', '-- ')
>> text(1.8, 7.0, 'z/a=0.6', 'FontSize', 16)
>> plot([1.5, 1.7], [6.0, 6.0], 'Color', 'k', 'LineWidth', 2, 'LineStyle', ': ')
>> text(1.8, 6.0, 'z/a=1.0', 'FontSize', 16)
```

The resulting graph is shown in Fig. E13.10. Note that, since $x/a = 1$ corresponds to the point directly above the loop, the peaks in these graphs occur right over the loop.

A similar set of statements will plot a graph of the $z$ component of the field produced by this source. We execute the statements

```
>> hold off
>> plot(x, Bz01, 'Color', 'k', 'LineWidth', 2)
>> title('z Component of Magnetic Field', 'FontSize', 20)
>> xlabel('x/a', 'FontSize', 16)
>> ylabel('B_z/B_0', 'FontSize', 16)
>> axis([0.0 3.0 -4.0 8.0])
>> hold on
>> plot(x, Bz03, 'Color', 'k', 'LineWidth', 2, 'LineStyle', '-. ')
```
Figure E13.11: The $z$ component of the magnetic field for various values of $z$.

The resulting graph is shown in Fig. E13.11.
Chapter 14

Finding Roots

14.12 Square Root by Newton’s Method (MATLAB)

Exercise: One way to find the square root of a (positive) number $a$ is to find the root of the function $f(x) = x^2 - a$. (a) Apply Newton’s method symbolically to show that $x_{n+1} = (x_n + a/x_n)/2$. (b) Using a pocket calculator and starting with the guess $x_0 = 2$, work out the first few iterates by hand and note how quickly this algorithm converges to $\sqrt{2} = 1.41421$. (This algorithm is the algorithm that most pocket calculators invoke when the square root key is pressed.) (c) Using whatever computational tool appeals to you, write a program that asks for the value of $a$, an initial guess for $\sqrt{a}$, and a tolerance and then implements Newton’s method to find $\sqrt{a}$, printing out each iterate along the way and stopping automatically when successive iterates differ by less than the specified tolerance.

Solution: (a) The square root of a number $a$ can be found by locating the roots of the equation $f(x) = x^2 - a$. With the shorthand notation $f_n = f(x_n)$ and $f'_n = df(x)/dx|_{x_n}$, we can write Newton’s method for the function of interest in the form

$$x_{n+1} = x_n - \frac{f_n}{f'_n} = x_n - \frac{x_n^2 - a}{2x_n} = x_n - \frac{x_n}{2} + \frac{a}{2x_n} = \frac{1}{2} \left(x_n + \frac{a}{x_n}\right)$$

(b) With a pocket calculator, if we start with $a = 2$ and the initial guess $x_0 = 2$, we then find that

$$x_1 = \frac{1}{2} \left(x_0 + \frac{2}{x_0}\right) = \frac{1}{2} \left(2 + \frac{2}{2}\right) = 0.5 \times 3 = 1.5$$

Then, the next iteration gives

$$x_2 = \frac{1}{2} \left(x_1 + \frac{2}{x_1}\right) = \frac{1}{2} \left(1.5 + \frac{2}{1.5}\right) = 1.4166667$$

and the next iteration gives

$$x_3 = \frac{1}{2} \left(x_2 + \frac{2}{x_2}\right) = \frac{1}{2} \left(1.4166667 + \frac{2}{1.4166667}\right) = 1.4142157$$

and the next gives

$$x_4 = \frac{1}{2} \left(x_3 + \frac{2}{x_3}\right) = \frac{1}{2} \left(1.4142157 + \frac{2}{1.4142157}\right) = 1.4142136$$

The square root of 2 obtained with the square root key on this pocket calculator gives 1.4142136, so four iterations have apparently given us the square root of 2 to seven digits after the decimal place.
(c) Using basic commands, it is not difficult to construct a program that utilizes Newton’s method to find the square root of a number. A sample MATLAB M-file might contain the statements

```matlab
% newton.m
%M-file newton.m finds the square root of a number specified
%by the user. The procedure uses an initial guess of
%the square root supplied by the user to start the process and then
%stops when it has reached the desired tolerance which is also specified
%by the user.
%
a = input('Number whose root is to be found: ');
val = input('Initial Guess: ');
toler = input('Absolute Tolerance: ');
diff = toler + 1.0;
disp(' ') 
disp('Iterative Results')
while diff > toler
    temp = (val + (a/val))/2.0;
diff = abs(temp - val);
val = temp;
end;
disp('Square Root is'); val
```

(The statement `diff = toler + 1.0` is included before the loop to make sure that `diff` is indeed greater than `toler` when the loop is started.) Note that the tolerance is an absolute tolerance and that each iterate is printed so the user can see how quickly the algorithm converges.

Once this program has been stored in the default directory with the name `newton.m`, we can find the square root of any positive number to any tolerance. For example, to find the square root of 2, we might run this command file with the statement and input

```
>> format long
>> newton
Number whose root is to be found: 2.0
Initial Guess: 1.0
Absolute Tolerance: 0.0001

Iterative Results

val = 1.50000000000000
val = 1.41666666666667
val = 1.41421568627451
val = 1.41421356237469

Square Root is
val = 1.41421356237469
```

Especially given the specified tolerance, we do not, of course, believe all of the digits printed out in the long format.

A few more sample sessions are shown below. First, we seek the square root of 2 again, but with a different initial guess, with the “conversation”
>> newton
Number whose root is to be found: 2.0
Initial Guess: 5.0
Absolute Tolerance: 0.0001

Iterative Results

<table>
<thead>
<tr>
<th>val</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.70000000000000</td>
</tr>
<tr>
<td>1.72037037037037</td>
</tr>
<tr>
<td>1.44145536817765</td>
</tr>
<tr>
<td>1.41447098136777</td>
</tr>
<tr>
<td>1.414213556237310</td>
</tr>
<tr>
<td>1.41421356237310</td>
</tr>
</tbody>
</table>

Square Root is
val = 1.41421356237310

Next, we seek the square root of 55 with the “conversation”

>> newton
Number whose root is to be found: 55.0
Initial Guess: 7.0
Absolute Tolerance: 0.0001

Iterative Results

<table>
<thead>
<tr>
<th>val</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.42857142857143</td>
</tr>
<tr>
<td>7.41620879120879</td>
</tr>
<tr>
<td>7.41619848710282</td>
</tr>
</tbody>
</table>

Square Root is
val = 7.41619848710282

In all these examples, convergence to five digits after the decimal point occurs in no more than half a dozen iterations.
14.14 Natural Frequencies of Bar (MATLAB)

**Exercise:** The natural frequencies for the transverse vibrations of a bar of uniform cross section that has length \( L \) and is free at both ends are given by

\[
\omega_n = \frac{4K}{L^2} \sqrt{\frac{E}{\rho}} \alpha_n^2
\]

where \( K \) is the radius of gyration of the cross section of the bar, \( E \) is Young’s modulus for the material of the bar, \( \rho \) is the density (mass/unit volume) of the material of the bar, and \( \alpha_n \) is a solution to the equation

\[
\tan \alpha = \pm \tanh \alpha
\]


**Solution:** The natural frequencies for the transverse vibrations of a bar of uniform cross section are given by the equation

\[
\omega_n = \frac{4K}{L^2} \sqrt{\frac{E}{\rho}} \alpha_n^2
\]

where \( L \) is the length of the bar, \( K \) is the radius of gyration of the cross-section of the bar, \( E \) is Young’s modulus for the material of the bar, \( \rho \) is the density of the material of the bar, and \( \alpha_n \) is a solution to the equation

\[
\tan(\alpha) = \pm \tanh(\alpha)
\]

Since \( K, L, E, \) and \( \rho \) are defined for a given bar, the natural frequencies are determined by \( \alpha_n \). The previous equation can be rewritten as

\[
\tan(\alpha) \mp \tanh(\alpha) = 0
\]

Therefore, we can find the first six values of \( \alpha_n^2 \), and thus the lowest half dozen natural frequencies of the bar, by finding the roots of this equation and then squaring them.

To find the roots of this equation using MATLAB, we first create graphs of the upper and lower equations. From these graphs we can tell about where the roots are and then use this information to specify a point near the root so that we can find the root with the function `fzero`. The statements

```matlab
>> dx = 20.0/250.0;
>> x = [0.0:dx:20.0];
>> tanx = tan(x);, tanhx = tanh(x);
>> tup = tanx - tanhx;, tlow = tanx + tanhx;
>> plot(x, tup)
>> axis([ 0.0, 20.0, -2.0, 2.0 ])
>> hold on
>> plot(x, tlow)
>> hold off
```

specify the independent variables for plotting the two functions over the interval \( 0.0 \leq x \leq 20.0 \) (dividing that interval into 250 segments); evaluate \( \tan(x) \), \( \tanh(x) \) and the two functions \( \tan(x) - \tanh(x) \) and \( \tan(x) + \tanh(x) \); and plot each (specifying a limit on the vertical range so that the divergences of the tangent function do not force the interesting parts of the graphs to be compressed near the axis \( y = 0 \)). Even so, these graphs are flawed by sharp (nearly—and incorrect) vertical lines

---

\(^1\)Throughout this solution, we use the symbol \( x \) in the MATLAB code for what we call \( \alpha \) in the statement of the exercise.
Exercise 14.14 (MATLAB)

Figure E14.1: Graph of the equations with the upper sign, i.e., of \(y = \tan(\alpha) - \tanh(\alpha)\) (solid line) and the lower sign, i.e., \(y = \tan(\alpha) + \tanh(\alpha)\) (dashed line).

at the points at which the tangent function diverges \((x = \pi/2, 3\pi/2, 5\pi/2, \ldots)\). To remove those vertical lines, we embellish the plotting procedure with the statements\(^2\)

\[
\begin{align*}
&\text{axis( [0.0, 20.0, -2.0, 2.0] )} \\
&\text{grid on} \\
&\text{index} = \begin{bmatrix} \pi/2, 3\pi/2, 5\pi/2, 7\pi/2, 9\pi/2, 11\pi/2, 13\pi/2 \end{bmatrix}/dx; \\
&\text{ub} = \text{int16(index - 2)}; \text{, ub(7)=250;} \\
&\text{ub} = \text{double(ub);} \\
&\text{lb} = [1, \text{int16(index+2)}]; \\
&\text{lb} = \text{double(lb);} \\
&\text{hold on} \\
&\text{for } i = 1:7 \text{ lbnd} = \text{lb(i); ubnd} = \text{ub(i);} \\
&\text{plot(x(lbnd:ubnd), tup(lbnd:ubnd), 'LineWidth', 3);} \text{ end} \\
&\text{for } i = 1:7 \text{ lbnd} = \text{lb(i); ubnd} = \text{ub(i);} \\
&\text{plot(x(lbnd:ubnd), tlow(lbnd:ubnd), 'LineWidth', 3, 'LineStyle', '- -'); end} \\
&\text{hold off}
\end{align*}
\]

thereby creating the display shown in Fig. E14.1. We then locate the roots by noting the approximate points at which these graphs cross the axis \(y = 0\).

Alternatively, we might graph the tangent and the hyperbolic tangent separately and look to points of intersection of these graphs. Statements accomplishing this objective might be

\[
\begin{align*}
&\text{axis( [0.0, 20.0, -2.0, 2.0] )} \\
&\text{grid on}
\end{align*}
\]

\(^2\)We must convert \(lb\) and \(ub\) back to doubles because MATLAB does not allow integers to be manipulated.

\(^3\)Here, \(\text{index}\) stores the values of index in \(x\) at which the tangent function diverges (indices corresponding to the points at which \(x\) is an odd multiple of \(\pi/2\)). Then \(\text{ub}\) stores a value of the index slightly lower than the index at which the tangent diverges (with the last value adjusted to identify the end of the array), and \(\text{lb}\) stores a value slightly higher than the index at which the tangent diverges. The values in \(\text{lbnd}\) then become the lower bounds for the intervals in which plotting will take place and the values in \(\text{ubnd}\) become the upper bounds for those intervals in the \texttt{for} loop, which plots the various branches of the curve while skipping over those points that would otherwise generate a steep vertical portion of the graph from the point at which it goes out of bounds at a positive value of \(y\) to the point at which it comes back in bounds at a negative value of \(y\).
Exercise 14.14 (MATLAB)

Figure E14.2: Graph of \( y = \tan(x) \) (solid line) and \( y = \pm \tanh(x) \) (dashed line).

\[
\begin{align*}
>> & \text{hold on} \\
>> & \text{plot}(x, \tan x, 'LineStyle', '\--', 'LineWidth', 3) \\
>> & \text{plot}(x, -\tanh x, 'LineStyle', '\--', 'LineWidth', 3) \\
>> & \text{for } i = 1:7 \text{ lbnd } = \text{lb}(i); \text{ ubnd } = \text{ub}(i); \\
>> & \text{plot}(x(\text{lbnd:ubnd}), \tan x(\text{lbnd:ubnd}), 'LineWidth', 3); \text{ end} \\
>> & \text{hold off}
\end{align*}
\]

create the display shown in Fig. E14.2, in which graphs of the two functions \( y = \pm \tanh(x) \) are overlayed on a graph of \( y = \tan(x) \).

From these graphs, we determine that the lowest half dozen roots we seek lie in the intervals

\[
\begin{align*}
\tan(\alpha) = -\tanh(\alpha) & \quad 2.0 < x < 3.0 \quad 5.2 < x < 6.0 \quad 8.0 < x < 9.0 \\
\tan(\alpha) = \tanh(\alpha) & \quad 3.5 < x < 4.5 \quad 6.5 < x < 7.5 \quad 10.0 < x < 10.7
\end{align*}
\]

(We ignore the root at \( x = 0 \) but we do make sure that the points selected are not on the opposite side of a point at which the tangent function diverges from the root being sought, i.e., on the wrong side of the values \( x = 1.57, 4.71, 7.85, 10.99, 14.14 \), since crossing that divergence could well mean trouble for root-finding algorithms.)

After creating the graphs, we write the upper and lower equations as function M-files as follows

```
function y = upmin(x)
%upmin.m
y = tan(x) - tanh(x);

function y = lowpl(x)
%lowpl.m
y = tan(x) + tanh(x);
```

which evaluate, respectively, the functions \( \tan(\alpha) - \tanh(\alpha) \) and \( \tan(\alpha) + \tanh(\alpha) \). After creating the function M-files, we create a command M-file that will find the roots of the function near a specified point using the function \texttt{fzero}. We write this program so that the user specifies where to start the search and which equation to look at, upper (upmin) or lower (lowpl). A sample file might be
Exercise 14.14 (MATLAB)

% alpharoots.m
% M-file that is used to find the roots of the alpha function, test those roots in the alpha function, and then square those roots. The M-file reads in the search point and which equation to search from the user. Then, the function fzero is used to find the root using the information supplied by the user.

a = input('Provide an initial guess of the root: '); name = input('upper(upmin) or lower(lowpl) solution: ', 's'); root = fzero(name, a); disp(''); disp('Root is '); root; trial = feval(name, root); disp('Alpha at Root is '); trial; sqr = root^2; disp('Root^2 is '); sqr

After creating this command M-file we can easily find the six lowest roots of the alpha equation. To find the first root in the equation with the upper sign (and save the square of the root), we execute the statements

>> alpharoots
Provide an initial guess of the root: 2.5
upper(upmin) or lower(lowpl) solution: lowpl

Root is
root = 2.3650
Alpha at Root is
trial = 6.6613e-16
Root^2 is
sqr = 5.5933
>> rtsq(1) = sqr;

Further sessions to find the other roots are shown below.

>> alpharoots
Provide an initial guess of the root: 5.5
upper(upmin) or lower(lowpl) solution: lowpl

Root is
root = 5.4978
Alpha at Root is
trial = 3.3307e-16
Root^2 is
sqr = 30.2258
>> rtsq(3) = sqr;

>> alpharoots
Provide an initial guess of the root: 8.5
upper(upmin) or lower(lowpl) solution: lowpl

Root is
root = 8.6394
Next, let us display the values of the squared roots, both in absolute terms and as multiples of the lowest frequency, with the statements

```matlab
>> rtsq
rtsq =  5.5933  15.4182  30.2258  49.9649  74.6389  104.2477
>> rtsq/rtsq(1)
an =  1.0000  2.7565  5.4039  8.9330  13.3443  18.6379
```

Note in particular that the frequencies at which this bar oscillates are not integer multiples of the lowest (fundamental) frequency.

From these sessions we can see that the values of $\alpha_n^2$ that correspond to the lowest six natural frequencies are 5.5933, 15.4182, 30.2258, 49.9649, 74.6389, and 104.2477. Thus, the lowest
half dozen natural frequencies of this bar are

\[
\begin{align*}
\omega_1 &= 5.59332 \frac{4K}{L^2} \sqrt{\frac{E}{\rho}} \\
\omega_2 &= 15.4182 \frac{4K}{L^2} \sqrt{\frac{E}{\rho}} = 2.75654\omega_1 \\
\omega_3 &= 30.2259 \frac{4K}{L^2} \sqrt{\frac{E}{\rho}} = 5.40392\omega_1 \\
\omega_4 &= 49.9649 \frac{4K}{L^2} \sqrt{\frac{E}{\rho}} = 8.93295\omega_1 \\
\omega_5 &= 74.6389 \frac{4K}{L^2} \sqrt{\frac{E}{\rho}} = 13.3443\omega_1 \\
\omega_6 &= 104.248 \frac{4K}{L^2} \sqrt{\frac{E}{\rho}} = 18.6379\omega_1
\end{align*}
\]
14.17 Finite Depth Square Well (MATLAB)

**Exercise:** Explore the way the energy levels of the well described in Section 14.1.5 change as the parameter $c$, which is determined by the depth and the width of the well, increases. At base, changing $c$ changes the slopes of the straight lines in Fig. E14.3. As $c$ increases and the well becomes deeper, the lines become more and more nearly horizontal and the number of energy levels increases. Seek ultimately to generate a graph that shows the energy of each allowed level on the vertical axis as a function of the parameter $c$ along the horizontal axis.

**Solution:** We examine the way that the energy levels of a quantum well change as the depth and width of the well (specified by $c$) change. As described in CPSUP Section 14.1.5, the value of $s$ for an allowed energy level must satisfy the equation

$$ s \cot(s) = -\sqrt{c^2 - s^2} $$

We can simplify this equation, as is done in Section 12.1.5, by squaring it, finding that the values of $s$ for allowed energy levels are the roots of the expression

$$ \sin(s) \mp \frac{s}{c} = 0 $$

We can find the roots of this equation in MATLAB by first finding the general area of the roots through graphing and then using `lubisect` to find the exact roots in those general areas. We first show the procedure for finding the energy levels for $c = 2$ and then show the procedure for finding

Figure E14.3: Graphs to reveal solutions to $\sin s \mp s/c = 0$. In each frame, the graph labeled ‘up’ corresponds to the upper sign in this equation and the graph labeled ‘low’ corresponds to the lower sign in this equation. In (a), roots lie where the graphs intersect the horizontal axis; in (b), roots lie where the sloped lines intersect the sine curve.
the roots for $c = 25$. These routines are almost identical, and the procedure for finding the energy levels for any other value of $c$ can be easily deduced from either procedure.

For $c = 2$, we first create a graph containing the plots of the upper and lower equations as well as the line $y = 0$. To create the graph, shown in Fig. E14.4, we execute the statements

```matlab
>> ds = 1.0/100;
>> s = [0.0:ds:10.0];
>> y1 = sin(s) - s/2;
>> y2 = sin(s) + s/2;
>> y3 = 0*s;
>> plot(s, y1, 'Color', 'black')
>> title('Finding Roots for c = 2', 'FontSize', 16)
>> hold on
>> plot(s, y2,'Color', 'black')
>> plot(s, y3,'Color', 'black')
>> text(7.5, -2.0, 'upper', 'FontSize', 14)
>> text(7.5, 3.0, 'lower', 'FontSize', 14)
>> hold off
```

From the graph we see that the upper equation has two roots, one between $-0.5 \leq s \leq 0.5$ and one between $1.5 \leq s \leq 2.5$, and the lower equation has one root between $-0.5 \leq s \leq 0.5$. Now that we have the general area of the roots we use `lubisect` to find the roots exactly. To make the procedure of finding roots easier, we create the M-files `qmupper.m` and `qmlower.m`. The file `qmupper.m` contains the lines

```matlab
function fct = qmupper(s)
global C
fct = sin(s) - s/C;
```

Figure E14.4: Plot of $\sin(s) \pm s/c$ for $c = 2$. The plot shows the general area of the roots to this equation.
and the file `qmlower.m` contains the lines

```matlab
function fct = qmlower(s)
global C
fct = sin(s) + s/C;
```

Note that we have declared `C` a global variable in these files so that we can change the value of `C` without editing the programs. Before we use the M-files we must declare `C` global in the MATLAB session and then assign it a value. Thus to find the roots, we execute the statements

```matlab
>> global C
>> C = 2.0;
>> upper2(1) = lubisect('qmupper', -0.5, 0.5, 1.0e-8, 40);
>> upper2(2) = lubisect('qmupper', 1.5, 2.5, 1.0e-8, 40);
>> lower2(1) = lubisect('qmlower', -0.5, 0.5, 1.0e-8, 40);
>> upper2
upper2 =
    0  1.8955
>> lower2
lower2 =
    0
```

Note that the size of the array needed to store the data varies for the upper and lower solutions as well as for each value of `c`. The size of the array is determined by how many roots appear on the graph. After creating the arrays we want to combine the upper and lower solutions as well as weed out the spurious solutions. We begin by eliminating the spurious solutions. To accomplish this task we plug the solutions back into the equation

\[ s \cot(s) = \sqrt{c^2 - s^2} \]

with the statements

```matlab
>> upper2.*cos(upper2)./sin(upper2) + sqrt(2.0^2-upper2.^2)
ans =
    1.0e-07 *
        NaN    0.3868
>> lower2.*cos(lower2)./sin(lower2) + sqrt(2.0^2-lower2.^2)
ans =
        NaN
```

Only those roots which produce a result of zero (within the precision of our determination of the root) are acceptable solutions. Therefore, we only accept `upper2(1)` in this case. We reject `upper2(0)` and `lower2(0)` because \( \lim_{s \to 0} s \cot(s) = 1 \) while \( -\sqrt{25^2 - s^2} = -25 \) and thus these roots do not satisfy the original equation. We then combine the acceptable upper and lower solutions into a single array and plug that array into the equation for \( E/V_0 \), specifically

\[ E/V_0 = -(1 - s^2/c^2) \]

to find the energies associated with the roots. (These energies, after all are the quantities in which we are actually interested.) We accomplish these tasks with the statements

\[ ^4 \text{Note that the statements used to accomplish the merging of the two separate solutions is different for small values of } c, \text{ but follow the form used for } c = 25 \text{ for larger values of } c \]
Exercise 14.17 (MATLAB)

```matlab
>> s2 = upper2(2);
>> s2
s2 =
    1.8955

>> energy2 = -(1 -s2.^2./2.0^2)
energy2 =
    -0.1018
```

Now that we have the allowed energy level for \( c = 2 \), we calculate the allowed energy levels for a different value of \( c \). We show the procedure for \( c = 25 \), since it is representative of the procedure for a large value of \( c \). Like the procedure for \( c = 2 \), we first create a graph with the statements

```matlab
>> ds = 3.0/100;
>> s = [0.0:ds:30.0];
>> y1 = sin(s) -s/25;
>> y2 = sin(s) +s/25;
>> y3 = 0*s;
>> plot(s, y1, 'Color', 'black')
>> title('Finding Roots for c = 25', 'FontSize', 16)
>> hold on
>> plot(s, y2, 'Color', 'black')
>> plot(s, y3, 'Color', 'black')
>> set(gca, 'XTick', 0:1:30)
>> text(23.5, -2.1, 'upper', 'FontSize', 14)
>> text(20.5, 2.1, 'lower', 'FontSize', 14)
>> grid on
>> hold off
```

The graph is shown in Fig. E14.5. Reading values for the intervals in which roots occur from this graph, we are then ready to use `lubisect` to find the exact roots. Before we use `lubisect`, however, we must change the value of \( C \). We then weed out the spurious roots and combine the two solutions. Finally, we use the solutions to find the allowed energy levels. These remaining tasks are accomplished with the statements

```matlab
>> upper25(1) = lubisect('qmupper', -0.5, 0.5, 1.0e-8, 40);
>> upper25(2) = lubisect('qmupper', 2.5, 3.5, 1.0e-8, 40);
>> upper25(3) = lubisect('qmupper', 6.0, 7.0, 1.0e-8, 40);
>> upper25(4) = lubisect('qmupper', 8.5, 9.5, 1.0e-8, 40);
>> upper25(5) = lubisect('qmupper', 12.5, 13.5, 1.0e-8, 40);
>> upper25(6) = lubisect('qmupper', 14.5, 15.5, 1.0e-8, 40);
>> upper25(7) = lubisect('qmupper', 19.0, 20.0, 1.0e-8, 40);
>> upper25(8) = lubisect('qmupper', 20.5, 21.5, 1.0e-8, 40);
>> lower25(1) = lubisect('qmlower', -0.5, 0.5, 1.0e-8, 40);
>> lower25(2) = lubisect('qmlower', 3.0, 4.0, 1.0e-8, 40);
>> lower25(3) = lubisect('qmlower', 5.5, 6.5, 1.0e-8, 40);
>> lower25(4) = lubisect('qmlower', 9.5, 10.5, 1.0e-8, 40);
>> lower25(5) = lubisect('qmlower', 11.5, 12.5, 1.0e-8, 40);
>> lower25(6) = lubisect('qmlower', 16.0, 17.0, 1.0e-8, 40);
>> lower25(7) = lubisect('qmlower', 17.5, 18.5, 1.0e-8, 40);
>> lower25(8) = lubisect('qmlower', 22.5, 23.5, 1.0e-8, 40);
>> lower25(9) = lubisect('qmlower', 23.5, 24.5, 1.0e-8, 40);
>> upper25
upper25 =
```

Figure E14.5: Plot of \( \sin(s) \mp \frac{c}{s} \) for \( c = 25 \). The plot shows the general area of the roots to this equation.
Exercise 14.17 (MATLAB)

Columns 7 through 8
20.9943 23.8645

>> energy25 = -(1-s25.^2./25.0^2)
energy25 =
Columns 1 through 6
-0.9854 -0.9416 -0.8688 -0.7672 -0.6370 -0.4791
Columns 7 through 8
-0.2948 -0.0888

After finding the allowed energy levels for \(c = 25\), we use similar procedures to find the allowed energy levels for other values of \(c\). Once we have found the allowed energy levels for a variety of \(c\) values, we plot a diagram that shows the allowed energy levels for different values of \(c\). In our case we choose 2, 5, 20, 25 and 50 as our values of \(c\). To create the plot we use the statements

>> plot([0.2, 0.8], [energy2(1), energy2(1)], 'Color', 'black', ... 'linewidth', 2.0)
>> hold on
>> for i=1:2 plot([1.2, 1.8], [energy5(i), energy5(i)], ... 'Color', 'black', 'LineWidth', 2.0); end
>> for i=1:3 plot([2.2, 2.8], [energy10(i), energy10(i)], ... 'Color', 'black', 'LineWidth', 2.0); end
>> for i=1:8 plot([3.2, 3.8], [energy25(i), energy25(i)], ... 'Color', 'black', 'LineWidth', 2.0); end
>> for i=1:16 plot([4.2, 4.8], [energy50(i), energy50(i)], ... 'Color', 'black', 'LineWidth', 2.0); end
>> set(gca, 'Visible', 'off', 'Ylim', [-1.5, 0.5], 'XLim', [0.0, 5.5]);
>> plot([0.0, 0.0], [-1.5,0.5], 'Color', 'black');
>> plot([0.0, 0.1], [0.0,0.0], 'Color', 'black');
>> text( 0.05, 0.4, 'E/V0', 'FontSize', 14);
>> text( -0.5, -1.1, '-1.0', 'FontSize', 12);
>> text( 0.4, -1.1, '2', 'FontSize', 12);
>> text( 1.4, -1.1, '5', 'FontSize', 12);
>> text( 2.4, -1.1, '10', 'FontSize', 12);
>> text( 3.4, -1.1, '25', 'FontSize', 12);
>> text( 4.4, -1.1, '50', 'FontSize', 12);
>> hold off

The resulting graph is shown in Fig. E14.6. From this diagram, we see that the number of allowed energy levels increases as the value of \(c\) increases. Since the energy levels are confined to a finite well, this means that the energy levels are closer together for larger values of \(c\). In addition, we see that for a specific value of \(c\), the energy levels get closer together as the energy levels get closer to the bottom of the well.
Figure E14.6: Energy level diagrams for the one-dimensional well for five different values of $c$. The heavy lines show the energies, measured in units of $V_0$, for the five different values of $c$ (here 2, 5, 10, 25, and 50); the light lines show the energies of the bottom ($-1$) and the top ($0$) of the well.
14.18 Single-Slit Diffraction (MATLAB)

**Exercise:** The intensity $I(x)$ in the diffraction pattern produced by a single slit is given by

$$
\frac{I(x)}{I_0} = \frac{\sin^2 x}{x^2}
$$

where $I_0$ is the intensity in the center and $x$ is related to the position of the observation point away from the central maximum. The zeroes in this pattern are easy to locate (they occur at $x = n\pi$, $n = 0, \pm 1, \pm 2, \ldots$). Careful location of the maxima, however, is more complicated. They don’t occur where $\sin^2 x = 1$ because of the influence of the denominator that steadily increases as $x$ increases. Locate the positions of the first half dozen maxima in this pattern, which—basically—is a request to find the roots of the derivative of the function (though note that not all roots correspond to maxima). Use at least three different methods and at least two different computational tools, and compare the results. Do your results confirm that the roots approach odd multiples of $\frac{1}{2}\pi$ as they become large? Optional: You might also find it interesting to approximate the function with a power series expansion for $\sin x$, keeping quite a few terms but converting the root finding problem into that of finding the roots of a polynomial. Then, use methods for finding roots of polynomials and see if you can come to understand how accuracy depends on how many terms you keep and which root you seek.

**Solution:** The intensity $I(x)$ in the diffraction pattern produced by a single slit is given by

$$
\frac{I(x)}{I_0} = \frac{\sin^2 x}{x^2}
$$

A graph of this function can be made easily in MATLAB by generating values for $x$ and evaluating the function at each point (correcting the value NaN produced at $x \neq 0$). An example plot is made by entering the statements

```matlab
>> dx = 10.0/100.0;
>> x = [0.0: dx: 10.0];
>> inten = sin(x).^2./x.^2;
>> inten(1) = 1.0;
>> plot(x, inten, 'Color', 'black', 'LineWidth', 3)
>> grid on
>> title('Intensity in diffraction pattern', 'FontSize', 16)
>> xlabel('x', 'FontSize', 12)
>> ylabel('I/I_0', 'FontSize', 12)
```

We want to find the values of $x$ at which this function has maxima. To do so, we elect to seek the roots of $(I/I_0)'$, i.e., the roots of

$$
g(x) = \frac{d}{dx} \left( \frac{\sin^2 x}{x^2} \right) = \frac{2\sin x \cos x}{x^2} - \frac{2\sin^2 x}{x^3}
$$

We can use `fzero` to find these roots. First, we write the function M-file

```matlab
function y = intensity(x)

p = 2*sin(x).*cos(x)./x.^2;
q = 2*sin(x).^2./x.^3;

y = p - q;
```

A graph of the roots produced by `fzero` is made by entering the statements

```matlab
>> roots = fzero(@intensity, [0, 10])
roots =
   0.0000
   0.1571
   0.5493
   0.9403
   1.3305
   1.7206
```

Then, we write another M-file with the `roots` as an input

```matlab
function intensity(x)

y = intensity(x, roots);
end
```

A graph of the positions of the maxima is made by entering the statements

```matlab
>> x = [0:0.1:10];
>> y = intensity(x, roots);
>> plot(x, y, 'Color', 'black', 'LineWidth', 3)
>> grid on
>> title('Positions of maxima', 'FontSize', 16)
>> xlabel('x', 'FontSize', 12)
>> ylabel('y', 'FontSize', 12)
```
and save it to the default directory with the name intensity.m. Once in MATLAB, we need an initial guess for the root we seek. Being careful to avoid the value $x = 0$, we see that 0.001 may find the maximum at zero with the statement

```matlab
>> fzero('intensity', 0.001)
ans = -8.0947e-09
```

This value is indeed close to zero (probably is zero within roundoff), which corresponds to the location of the central maximum in the diffraction pattern. By looking at the graph, we can see about where each subsequent maximum occurs. Therefore, we can determine what each initial guess should be so that `fzero` will converge an the desired maximum. The first maximum above the central maximum appears to occur in the vicinity of $x = 4.5$, so we invoke the statement

```matlab
>> fzero('intensity', 4.5)
ans = 4.4934
```

and, to check the closeness to $3\pi/2$, we execute as well the statement

```matlab
>> 3*pi/2 - ans
ans = 0.2190
```

We find additional maxima and check their approach to $(2n + 1)\pi/2$ with the statements

```matlab
>> fzero('intensity', 7.5)
ans = 7.7253
>> 5*pi/2 - ans
ans = 0.1287
```

```matlab
>> fzero('intensity', 10.5)
ans = 10.9041
>> 7*pi/2 - ans
ans = 0.0915
```
Exercise 14.18 (MATLAB)

>> fzero('intensity', 13.5)
an = 14.0662
>> 9*pi/2 - ans
ans = 0.0710

>> fzero('intensity', 17.0)
an = 17.2208
>> 11*pi/2 - ans
ans = 0.0580

By looking at the roots and the comparisons that follow, we see that the maxima are found at \(x \approx 0, 3\pi/2, 5\pi/2, 7\pi/2, \text{ etc.}\). Furthermore, we see that the difference between the root and the multiple of \(\pi/2\) lessens as the size of the root increases. So, the maxima do indeed approach odd multiples of \(\pi/2\) as the roots become large.
14.19 Roots of $J_0(x)$ (MATLAB)

**Exercise:** Using at least three different methods and at least two different computational tools, find the first half dozen roots of the zeroth-order Bessel function $J_0(x)$. Note that these roots are related to the radii of circular nodes in some of the vibrations of a circular membrane. The values of these roots tabulated in Abramowitz and Stegun are $2.404825577$, $5.5200781103$, $8.6537279129$, $11.7915344391$, $14.9309177086$, and $18.0710639679$. *Hint:* Most computational tools have built-in capabilities for evaluating the Bessel functions. Consult the appropriate vendor manuals.

**Solution:** We wish to know the first half dozen roots of the zeroth-order Bessel function. The values of the roots as tabulated by Abramowitz and Stegun are $2.404855577$, $5.5200781103$, $8.6537279129$, $11.7915344391$, $14.9309177086$, and $18.0710639679$. The bessel function is available in MATLAB under the name `besselj(0,x)`. If we define and save the function, we can use `fzero` and a few good initial guesses to find the roots.

```matlab
function y = bess(x)
y=besselj(0,x);
```

To use this, we execute the statements

```matlab
>> fzero( @bess, 2.0 )
ans = 2.4048

>> fzero( @bess, 5.0 )
ans = 5.5201

>> fzero( @bess, 8.0 )
ans = 8.6537

>> fzero( @bess, 11.0 )
ans = 11.7915

>> fzero( @bess, 14.5 )
ans = 14.9309

>> fzero( @bess, 17.5 )
ans = 18.0711
```

So, we find that the first half dozen roots of the zeroth-order Bessel function are $2.4048$, $5.5201$, $8.6537$, $11.7915$, $14.9309$, and $18.0711$.

Had we not known ahead of time where the roots lie, we could produce a graph—shown in Fig. E14.8—with the statements

```matlab
>> x=[0:0.1:20.0];
>> plot( x, bess(x), 'Color', 'black', 'Linewidth', 4 );
>> grid on
>> title('J_0(x) versus x', 'FontSize', 20 )
```

Then, we could read the approximate location of each root from the scales on the graph.

---

Figure E14.8: Graph of $J_0(x)$ versus $x$. 

$J_0(x)$ versus $x$
14.20 Double-Welled Potential (MATLAB)

**Exercise:** Suppose a particle moves in one dimension under the influence of the potential energy

\[ V(x) = \frac{-V_0 a^2 (a^2 + x^2)}{8a^4 + x^4} \]

where \( \pi = x/a \). Using at least three different methods and at least two different computational tools, find the coordinates \( \pi \) of all turning points when the total energy \( E \) of the particle is \( E = -0.2V_0 \) and also when the total energy is \( E = -0.1V_0 \). *Optional:* Obtain graphs of the position of each turning point as a function of particle energy over the allowed range of energies for bound states.

**Solution:** We can examine the one-dimensional motion of a particle under the influence of a potential energy

\[ \frac{V(x)}{V_0} = \frac{1 + X^2}{8 + X^4} \]

We wish to know the coordinates of all the turning points when the total energy \( E \) is \( E = -0.1V_0 \) as well as when \( E = -0.2V_0 \). To do this we must find the roots of the equations

\[ V_1 = 0.1 - \frac{1 + X^2}{8 + X^4} \quad \text{and} \quad V_2 = 0.2 - \frac{1 + X^2}{8 + X^4}, \]

respectively. (At the roots, the kinetic energy \( K \) will be \( K = 0 \).)

To emphasize the effect of changing the total energy on the locations of the turning points, we graph \( V(x)/V_0 \) with the two different total energies clearly marked. The graph appears in Fig. E14.9 and was created with the statements

```matlab
>> dx=10.0/101; 
>> x=[-5.0:dx:5.0];
>> y=-((1.0+x.^2)./(8.0+x.^4));
>> plot(x,y, 'Color', 'black', 'LineWidth', 4)
>> title('Potential Energy Curve', 'FontSize', 20)
>> xlabel('X','FontSize',16)
>> ylabel('V1, V2','FontSize',16)
>> grid on
>> hold on

To find the roots of \( V_1 \), we first create the function M-file `vone.m` containing the statements

```matlab
function y=vone(x)
% Equation for the turning points when E=-0.1V_0 
% y = 0.1-((1.0+x.^2)./(8.0+x.^4));
% y = 0.1-((1.0+x.*2)./(8.0+x.*4));
```

and save it to the default directory. Then, referring to our graph for appropriate starting points, we can call `fzero` in MATLAB with the statements

```matlab
>> dx=10.0/101; 
>> x=[-5.0:dx:5.0];
>> y=-((1.0+x.^2)./(8.0+x.^4));
>> plot(x,y, 'Color', 'black', 'LineWidth', 4)
>> title('Potential Energy Curve', 'FontSize', 20)
>> xlabel('X','FontSize',16)
>> ylabel('V1, V2','FontSize',16)
>> grid on
>> hold on

To find the roots of \( V_1 \), we first create the function M-file `vone.m` containing the statements

```matlab
function y=vone(x)
% Equation for the turning points when E=-0.1V_0 
% y = 0.1-((1.0+x.^2)./(8.0+x.^4));
% y = 0.1-((1.0+x.*2)./(8.0+x.*4));
```

and save it to the default directory. Then, referring to our graph for appropriate starting points, we can call `fzero` in MATLAB with the statements
To find the roots of $V_1$, we proceed in a similar fashion. First we create the function M-file `vtwo.m` containing the statements

```matlab
function y=vtwo(x)
% Equation for the turning points when E=-0.2V_o
y = 0.2-((1.0+x.^2)./(8.0+x.^4));
```

and save it to the default directory. Then, referring again to our graph for appropriate starting points, we call `fzero` in MATLAB with the statements

```matlab
>> t1=fzero('vtwo','-2.0')
t1 = -2.07431329305194
>> t2=fzero('vtwo','-1.0')
t2 = -0.83499961812447
>> t3=fzero('vtwo','1.0')
t3 = 0.83499961812447
>> t4=fzero('vtwo','2.0')
t4 = 2.07431329305194
```
14.24 A Fluid Mechanics Problem (MATLAB)

**Exercise:** A particular problem—see Problem 3-19 in the fourth edition of *Fluid Flow* by Rolf H. Sabersky, Allan J. Acosta, Edward G. Hauptmann, and E. M. Gates (Prentice-Hall, Upper Saddle River, NJ, 1999)—in fluid flow leads to the need to find the roots of the fourth-order polynomial 
\[ 12x^4 - 12x^3 + 4x - 1. \]
Use graphical methods to find bounds on the roots and at least three different computational approaches to find all real roots of this polynomial.

**Solution:** Graphs of the fourth-order polynomial

\[ 12x^4 - 12x^3 + 4x - 1 \]

are given in Fig. E14.10 and Fig. E14.11. To create the first graph, which shows the fourth-order polynomial over a fairly wide interval in \( x \), we execute the statements

```matlab
>> dx = 2.0/100.0;
>> x= [-10:dx:10.0];
>> y =12.*x.^4 - 12.*x.^3+ 4.*x-1;
>> plot(x, y, 'color','black', 'LineWidth', 3 )
>> hold on
>> plot([-10.0,10.0], [0.0,0.0], 'color', 'black',...
      'LineStyle', '-.', 'LineWidth', 2 );
>> hold off
```

This graph, however, does not reveal the location of the roots very well, so we draw it again over a much smaller range of values of \( x \). To create the second graph we invoke the statements

```matlab
>> x= [-1.0:dx:1.0];
>> y =12.*x.^4 - 12.*x.^3+ 4.*x-1;
>> plot(x, y, 'color','black', 'LineWidth', 3 )
>> hold on
>> plot([-1.0,1.0], [0.0,0.0], 'color', 'black',...
      'LineStyle', '-.', 'LineWidth', 2 );
>> hold off
```

From the graphs we see that the roots lie within the range of \(-1 \leq x \leq 1\).

Figure E14.10: Graph of the polynomial with a wide domain.

Figure E14.11: Graph of the polynomial around the origin.
Exercise 14.24 (MATLAB)

Now that we have this information, we can find the roots of the polynomial in two ways.

**Method 1:** Using `roots`. In this method, we define a vector containing the coefficients of the polynomial (starting with the coefficient of the highest order term and then invoke `roots`.

```matlab
>> A = [12.0, -12.0, 0.0, 4.0, -1.0];
>> theroots = roots(A)
theroots = -0.5570
  0.6218 + 0.3013i
  0.6218 - 0.3013i
  0.3134
```

We find four roots (as we must for a fourth-order polynomial. As we could infer from the graph, however, only two of these roots are real, namely $x = -0.5570$ and $x = 0.3134$.

**Method 2:** Using `fzero`. In this method, we first need an M-file to define the function, so we create the file `flowpoly.m` containing the lines

```plaintext
function y = flowpoly(x)
    y = 12*x^4 - 12*x^3 + 4*x - 1;
```

With that file stored in the default directory, we then execute the statements

```matlab
rootone = fzero( @flowpoly, -1.0 )
rootone = -0.5570
roottwo = fzero( @flowpoly, 0.0 )
roottwo = 0.3134
```

We use `fzero` by first creating an initial guess for one of the roots. We then pass this guess to `fzero` along with the function M-file that contains the given polynomial.

From both methods we see that the two real roots of the given fourth-order polynomial are $-0.5570$ and $0.3134$. 
Chapter 15

Introduction to Partial Differential Equations

15.2 A Vertically Hanging String

Exercise: Suppose the one-dimensional string of length $l$ discussed in Section 15.1.1 hangs vertically and is acted on by gravity. Suppose that $u(x, t)$ and $v(x, t)$ give the transverse (horizontal) and longitudinal (vertical) displacements of the particle of the string nominally located at $x$, which is measured downward from the top of the string. (a) Examine the forces acting on this string, deduce the general equations for both longitudinal and transverse motion of the string, and show ultimately that, if the motion is entirely transverse and the amplitude of the motion is small, the equations reduce to

$$
\rho(x) \frac{\partial^2 u}{\partial t^2} = \frac{\partial}{\partial x} \left( \tau(x) \frac{\partial u}{\partial x} \right) ; \quad \tau(x) = -g \int_x^l \rho(x') \, dx'
$$

where $\rho(x)$ is the mass per unit length of the string. (b) Taking $\rho$ to be constant, show that the tension $\tau(x)$, which is simply equal to the weight of the string below $x$, is given by $\tau(x) = \rho g (l - x)$. With this restriction, the equation of motion then becomes

$$
\frac{\partial^2 u}{\partial t^2} = \frac{\partial}{\partial x} \left( (l - x) g \frac{\partial u}{\partial x} \right)
$$

(c) Recast this equation in dimensionless form by introducing the variables $\bar{x} = x/l$ and $\bar{t} = t \sqrt{g/l}$. (d) Suppose you seek a sinusoidal solution for which $u(\bar{x}, \bar{t}) = f(\bar{x}) \cos \omega \bar{t}$. Find the ODE satisfied by $f(\bar{x})$ and then introduce the new independent variable $y$ defined by $y^2 = 1 - \bar{x}$ to find that, expressed as a function of $y$, $f$ satisfies the Bessel equation

$$
y^2 \frac{d^2 f}{dy^2} + y \frac{df}{dy} - 4\omega^2 y^2 f = 0
$$

Note: The variable transformation $y^2 = 1 - x$ in effect recognizes that the vertical string is more appropriately treated by locating points on the string relative to the bottom rather than the top of the string!

Solution: (a) In order to deduce the equation of motion for a string that hangs vertically and is acted on by gravity, we first determine the forces that act on the string. A force diagram is shown in Fig. E15.1. Then, using Newton’s second law, $\mathbf{F} = m\mathbf{a}$, we write the equations of motion. We take the transverse displacement to be $u(x, t)$, and the longitudinal displacement to be $v(x, t)$. Also
Figure E15.1: Force diagram for a vertically hanging string acted upon by gravity.

Note that we have chosen the positive x direction to be down, and we have chosen to measure u positive to the right and v positive down. Then, with the mass of the element shown in the figure given by \( \rho(x) \Delta x \), we write Newton’s second law for that element to be

\[
\rho(x) \Delta x \frac{\partial^2 u}{\partial t^2} = \tau(x + \Delta x, t) \sin \theta(x + \Delta x, t) - \tau(x, t) \sin \theta(x, t)
\]

for the transverse motion of the string and

\[
\rho(x) \Delta x \frac{\partial^2 v}{\partial t^2} = \rho(x) \Delta x g + \tau(x + \Delta x, t) \cos \theta(x + \Delta x, t) - \tau(x, t) \cos \theta(x, t)
\]

for the longitudinal motion of the string. Now, we divide by \( \Delta x \), and let \( \Delta x \) go to zero, reducing the equations of motion to

\[
\rho(x) \frac{\partial^2 u}{\partial t^2} = \frac{\partial}{\partial x} \left( \frac{\tau(x, t) \sin \theta(x, t)}{\left( \frac{\partial u}{\partial x} \right)^2 + \left( 1 + \frac{\partial v}{\partial x} \right)^2} \right)
\]

and

\[
\rho(x) \frac{\partial^2 v}{\partial t^2} = \rho(x) g + \frac{\partial}{\partial x} \left( \tau(x, t) \cos \theta(x, t) \right)
\]

We have too many unknowns in the above equations, so we must eliminate \( \theta(x, t) \). Using Fig. E15.2, we can see that

\[
\sin \theta = \frac{\Delta u}{\Delta s} = \frac{\Delta u}{\sqrt{\left( \Delta u^2 + (\Delta x + \Delta v)^2 \right)^{1/2}}} = \frac{\partial u}{\partial x} \left[ \left( \frac{\partial u}{\partial x} \right)^2 + \left( 1 + \frac{\partial v}{\partial x} \right)^2 \right]^{1/2}
\]

and that

\[
\cos \theta = \frac{\Delta u + \Delta v}{\Delta s} = \frac{1 + \frac{\partial v}{\partial x}}{\left[ \left( \frac{\partial u}{\partial x} \right)^2 + \left( 1 + \frac{\partial v}{\partial x} \right)^2 \right]^{1/2}}
\]

We can now substitute \( \sin \theta \) and \( \cos \theta \) into the two equations from which we need to eliminate \( \theta \) to find that

\[
\rho(x) \frac{\partial^2 u}{\partial t^2} = \frac{\partial}{\partial x} \left[ \frac{\tau(x, t) \frac{\partial u}{\partial x}}{\left[ \left( \frac{\partial u}{\partial x} \right)^2 + \left( 1 + \frac{\partial v}{\partial x} \right)^2 \right]^{1/2}} \right]
\]
\[ \rho(x) \frac{\partial^2 v}{\partial t^2} = \rho(x) g + \frac{\partial}{\partial x} \left[ \frac{\tau(x, t) \left(1 + \frac{\partial u}{\partial x}\right)}{\left(\left(\frac{\partial u}{\partial x}\right)^2 + \left(1 + \frac{\partial v}{\partial x}\right)^2\right)^{1/2}} \right] \]

At this point, we have only the unknowns that we are looking for, \(u(x, t)\) and \(v(x, t)\). In order to be able to solve these equations, we must make a few approximations. If (1) the amplitude of the motion is small so that
\[ \Delta u \ll \Delta x \implies \frac{\partial u}{\partial x} \ll 1 \]

and (2) the motion is transverse so that \(v = 0\) everywhere and always, then the equation for \(v\) reduces to
\[ 0 = \rho(x) g + \frac{\partial \tau}{\partial x} \implies \tau(x) = -\int_{l}^{x} \rho(x') g \, dx' \]

where we find that, in this limit, \(\tau(x)\) cannot depend on \(t\). (The limits are chosen so that \(\tau\) will be zero at the bottom of the string where \(x = l\) and grow to the total weight of the string at the top of the string, where \(x = 0\).) In this same limit, the equation for \(u\) becomes
\[ \rho(x) \frac{\partial^2 u}{\partial t^2} = \frac{\partial}{\partial x} \left[ \tau(x) \frac{\partial u}{\partial x} \right] \]

In the special case that \(\rho(x)\) is, in fact, constant, then the integral giving \(\tau(x)\) can be quickly evaluated to give that
\[ \tau(x) = \rho g (l - x) \]

and, in consequence, the equation for \(u\) becomes
\[ \frac{\partial^2 u}{\partial t^2} = \frac{\partial}{\partial x} \left[ g (l - x) \frac{\partial u}{\partial x} \right] \]

We next cast this result in a dimensionless form by introducing the variables \(\bar{x} = x/l\) and \(\bar{t} = t \sqrt{g/l}\) to find that
\[ \frac{\partial^2 u}{\partial \bar{t}^2} = \frac{\partial}{\partial \bar{x}} \left[ g \bar{l} (1 - \bar{x}) \frac{\partial u}{l \partial \bar{x}} \right] = \frac{g}{\bar{l}} \frac{\partial}{\partial \bar{x}} \left[ (1 - \bar{x}) \frac{\partial u}{\partial \bar{x}} \right] \]

and then that
\[ \frac{\partial^2 u}{\partial \bar{t}^2} = \frac{\partial}{\partial \bar{x}} \left[ (1 - \bar{x}) \frac{\partial u}{\partial \bar{x}} \right] \]
Suppose, now, that \( u(\vec{x}, t) = f(\vec{x}) \cos \omega t \). Substitution of this form into the equation we have deduced leads to the result

\[
\omega^2 f = \frac{d}{d\vec{x}} \left[ (1 - \vec{x}) \frac{df}{d\vec{x}} \right]
\]

Finally, let us substitute the independent variable \( y \) defined by \( y^2 = 1 - \vec{x} \), in terms of which

\[
2y \frac{dy}{dx} = -1 \quad \Rightarrow \quad \frac{dy}{dx} = -\frac{1}{2y}
\]

With that change, the equation becomes

\[
\omega^2 f = -\frac{1}{2y} \frac{d}{dy} \left[ y^2 \left( -\frac{1}{2y} \right) \frac{df}{dy} \right] = \frac{1}{4y} \frac{d}{dy} \left[ y \frac{df}{dy} \right] = \frac{1}{4y} \left[ y \frac{d^2 f}{dy^2} + \frac{df}{dy} \right]
\]

Multiplying by \( 4y^2 \) bringing all terms to the same side of the equation, we finally find that

\[
y^2 \frac{d^2 f}{dy^2} + y \frac{df}{dy} - 4\omega^2 y^2 f = 0
\]

which is, in fact, the zeroth-order Bessel equation.
15.3 Second-Order ODEs Made Self-Adjoint

**Exercise:** Show that every second-order linear ODE can be cast in self-adjoint form, i.e., show that functions $\alpha(x), \gamma(x),$ and $f(x)$ can be found such that the general second-order linear ODE

$$a(x) \frac{d^2 \varphi}{dx^2} + b(x) \frac{d \varphi}{dx} + c(x) \varphi = g(x)$$

can be recast in self-adjoint form as defined in Section 15.1.7. **Hint:** Multiply the equation by an undetermined integrating factor $h(x)$, find a (first-order) differential equation for $h(x)$, and solve that equation—at least formally.

**Solution:** We want to show that the general second-order ODE

$$a(x) \frac{d^2 \varphi}{dx^2} + b(x) \frac{d \varphi}{dx} + c(x) \varphi = g(x)$$

can be recast in self-adjoint form, specifically

$$-\alpha(x) \frac{d^2 \varphi(x)}{dx^2} - \alpha'(x) \frac{d \varphi(x)}{dx} + \beta(x) \varphi(x) = f(x)$$

We'll begin by multiplying the general equation by an undetermined function $h(x)$, which we will later find as an integrating factor, obtaining that

$$h(x) a(x) \frac{d^2 \varphi}{dx^2} + h(x) b(x) \frac{d \varphi}{dx} + h(x) c(x) \varphi(x) = h(x) g(x)$$

We can easily see that, for this equation to be recast in self-adjoint form, we must choose $h(x)$ so that

$$h(x) a(x) = -\alpha(x) \quad \text{and} \quad h(x) b(x) = -\alpha'(x)$$

or

$$\frac{d}{dx} \left( h(x) a(x) \right) = h(x) b(x) \quad \Rightarrow \quad h'(x) a(x) + h(x) a'(x) = h(x) b(x)$$

Equivalently, we have that

$$h'(x) = \frac{dh}{dx} = h(x) \left( \frac{b(x) - a'(x)}{a(x)} \right) \quad \Rightarrow \quad \frac{dh}{h} = \left( \frac{b(x) - a'(x)}{a(x)} \right) dx$$

This last expression can be integrated (indefinite integral) to yield that

$$\int \frac{dh}{h} = \int \left( \frac{b(x) - a'(x)}{a(x)} \right) dx \quad \Rightarrow \quad \ln h = \int \frac{b(x)}{a(x)} dx - \ln a + C'$$

or, on exponentiation, that

$$h(x) = C \frac{e^{\int (b(x)/a(x)) \, dx}}{a(x)}$$

where $C = e^{C'}$ is an arbitrary constant.

Having found the integrating factor, we now multiply the original equation by this factor—though we can ignore the constant $C$ to find that

$$e^{\int (b(x)/a(x)) \, dx} a(x) \frac{d^2 \varphi(x)}{dx^2} + \frac{b}{a} e^{\int (b(x)/a(x)) \, dx} \frac{d \varphi(x)}{dx} + \frac{c}{a} e^{\int (b(x)/a(x)) \, dx} \varphi(x) = \frac{g}{a} e^{\int (b(x)/a(x)) \, dx}$$

If this equation is to match the desired form, we must make the identification

$$\alpha(x) = -e^{\int (b(x)/a(x)) \, dx}$$
With that choice, we then find that
\[ \alpha'(x) = \frac{d}{dx} \left( -e^{\int \frac{b}{a} \, dx} \right) = -e^{\int \frac{b}{a} \, dx} \frac{d}{dx} \int \frac{b}{a} \, dx = -\frac{b}{a} e^{\int \frac{b}{a} \, dx} \]

which is exactly the coefficient of the derivative \( d\varphi/dx \) in the first equation in this paragraph. To finish, we recognize by comparing the ODE just derived with the second equation in this solution that
\[ \beta(x) = \frac{c}{a} e^{\int \frac{b}{a} \, dx} \]

and
\[ f(x) = \frac{g}{a} e^{\int \frac{b}{a} \, dx} \]
15.7 FDM 1D Homogeneous Helmholtz Equation

Exercise: Consider the differential equation and boundary conditions
\[ \frac{d^2 u}{dx^2} + k^2 u = 0 \quad ; \quad u(0) = u(L) = 0 \]

(a) Find the analytic solution to this problem and identify the special values of \( k \) that permit non-trivial solutions. (b) Show that the finite difference approach to the problem leads to the matrix eigenvalue problem \( Au = -k^2 L^2 u/N^2 \) where \( N \) is the number of equal-length segments into which the length \( L \) of the domain is divided and
\[
A = \begin{pmatrix}
  -2 & 1 & 0 & \cdots & 0 & 0 & 0 \\
  1 & -2 & 1 & \cdots & 0 & 0 & 0 \\
  0 & 1 & -2 & \cdots & 0 & 0 & 0 \\
  \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\
  0 & 0 & 0 & \cdots & 1 & -2 & 1 \\
  0 & 0 & 0 & \cdots & 0 & 1 & -2 \\
\end{pmatrix} \quad ; \quad u = \begin{pmatrix}
  u_1 \\
  u_2 \\
  u_3 \\
  \vdots \\
  u_{N-3} \\
  u_{N-2} \\
  u_{N-1} \\
\end{pmatrix}
\]

Note that, when the domain \( 0 \leq x \leq L \) is divided into \( N \) segments, there will be \( N + 1 \) nodes ranging from \( x_0 = 0 \) to \( x_{N+1} = L \). Because, along the way to a solution, the boundary conditions result in the rows and columns associated with these two nodes being deleted, these matrices will have only \( N - 1 \) rows and columns.

(c) Taking \( N = 100 \), use an available numeric processing program like IDL, MATLAB, OCTAVE, or PYTHON to find the first several eigenvalues \( k_n \) and compare your results with the values found in part (a).

Solution: (a) The general solution to the differential equation is well known to be
\[ u(x) = A \sin kx + B \cos kx \]
The boundary conditions require that
\[ u(0) = B = 0 \quad ; \quad u(L) = A \sin kL = 0 \implies kL = n\pi, n = 1, 2, 3, \ldots \]
so there is a family of solutions
\[ u(x) = A \sin \left( \frac{n\pi x}{L} \right) = A \sin k_n x \quad ; \quad k_n = nk_1 = n \frac{\pi}{L} \]
The non-trivial solutions are sinusoidal with arbitrary amplitude, and the wave number \( k_n \) associated with the \( n \)-th solution is an integer multiple of the wave number \( k_1 = \pi/L \) associated with the first solution.

(b) To cast the equation
\[ \frac{d^2 u}{dx^2} + k^2 u = 0 \]
in a discretized form, we divide the interval \( 0 \leq x \leq L \) into \( N \) segments by placing nodes at the points \( x_i = i \Delta x \) where \( \Delta x = L/N, 0 \leq i \leq N \), introduce the quantities \( u_i = u(x_i) \), and remember that, with that grid,
\[ \frac{d^2 u}{dx^2} \bigg|_{x=x_i} = \frac{u_{i+1} - 2u_i + u_{i-1}}{\Delta x^2} \]
Thus, we find that evaluating the differential equation at \( x = x_i \) leads to the algebraic equation
\[ \frac{u_{i+1} - 2u_i + u_{i-1}}{\Delta x^2} + k^2 u_i = 0 \quad \implies \quad u_{i+1} - 2u_i + u_{i-1} = -k^2 \Delta x^2 u_i = -\frac{k^2 L^2}{N^2} u_i \]
At the two ends of the domain \((i = 0\) and \(i = N\)) however, this equation involves quantities—\(u_{-1}\) and \(u_{N+1}\)—that are outside the domain, so we don’t know them. But the boundary conditions allow us to ignore the original equations,

\[
\begin{align*}
\frac{k^2L^2}{N^2}u_1 - 2u_0 + u_{-1} &= - \frac{k^2L^2}{N^2}u_0; \\
\frac{k^2L^2}{N^2}u_{N+1} - 2u_N + u_{N-1} &= - \frac{k^2L^2}{N^2}u_N
\end{align*}
\]

replacing them with the equations

\[
\begin{align*}
u_0 &= 0; \\
u_N &= 0
\end{align*}
\]

respectively. In total, then, the equations we seek to solve are

\[
\begin{align*}
u_0 &= 0 \\
u_i - 1 &= - \frac{k^2L^2}{N^2}u_{i-1} & \text{for} \; 2 \leq i \leq N - 2 \\
u_N &= 0
\end{align*}
\]

To recast these equations in a matrix form, we would leave out the first and last (for which we already know a solution) and introduce the vector

\[
\left( u_1, u_2, u_3, \ldots, u_{N-2}, u_{N-1} \right)
\]

(though we should probably write it as a column rather than a row). Further, we then note that we need to single out the equations for \(i = 1\) and \(i = N - 1\) for special treatment since, without that treatment the first of these refers to \(u_0\) and the second refers to \(u_N\), which we have omitted from the equations to be solved. For \(i = 1\), for example, the equation becomes

\[
u_2 - 2u_1 + u_0 = - \frac{k^2L^2}{N^2}u_1 \implies u_2 - 2u_1 = - \frac{k^2L^2}{N^2}u_1
\]

(remember that \(u_0 = 0\)). Similarly, the equation for \(i = N - 1\) becomes

\[
u_{N-2} - 2u_{N-1} + u_N = - \frac{k^2L^2}{N^2}u_{N-1} \implies u_{N-2} - 2u_{N-1} = - \frac{k^2L^2}{N^2}u_{N-1}
\]

(remember that \(u_N = 0\)). Finally, we conclude that the equations to be solved are

\[
\begin{align*}
-2u_1 + u_2 &= - \frac{k^2L^2}{N^2}u_1 \\
u_{i-1} - 2u_i + u_{i+1} &= - \frac{k^2L^2}{N^2}u_i & 2 \leq i \leq N - 2 \\
u_{N-2} - 2u_{N-1} &= - \frac{k^2L^2}{N^2}u_{N-1}
\end{align*}
\]

Here, we have reversed the order of the terms on the left-hand side so that the indices increase across the equation rather than decrease, thus rendering more evident the equivalent—and more compact—matrix presentation

\[
\begin{pmatrix}
-2 & 1 & 0 & 0 & \ldots & 0 & 0 & 0 \\
1 & -2 & 1 & 0 & \ldots & 0 & 0 & 0 \\
0 & 1 & -2 & 1 & \ldots & 0 & 0 & 0 \\
0 & 0 & 1 & -2 & \ldots & 0 & 0 & 0 \\
\vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\
0 & 0 & 0 & 0 & \ldots & -2 & 1 & 0 \\
0 & 0 & 0 & 0 & \ldots & 1 & -2 & 1 \\
0 & 0 & 0 & 0 & \ldots & 0 & 1 & -2
\end{pmatrix}
\begin{pmatrix}
u_1 \\
u_2 \\
u_3 \\
u_4 \\
\vdots \\
u_{N-3} \\
u_{N-2} \\
u_{N-1}
\end{pmatrix} = - \frac{k^2L^2}{N^2}
\begin{pmatrix}
u_1 \\
u_2 \\
u_3 \\
u_4 \\
\vdots \\
u_{N-3} \\
u_{N-2} \\
u_{N-1}
\end{pmatrix}
\]
Exercise 15.7

In terms of these quantities, the problem we must solve is the simple eigenvalue problem

\[ Au = \lambda u = -\frac{k^2 L^2}{N^2} u \quad \Rightarrow \quad k_n = \sqrt{-\lambda_n} \frac{N}{L} \quad \Rightarrow \quad \frac{k_n}{k_1} = \frac{\sqrt{-\lambda_n}}{\sqrt{-\lambda_1}} \]

where \( A \) is the coefficient matrix, \( u \) is a vector of the unknown values at the nodes 1 to \( N-1 \), and \( \lambda = -\frac{k^2 L^2}{N^2} \) is the eigenvalue to be found. At this point, we hope that the values of \( \lambda_n \) turn out to be negative. Note that these matrices have dimension \((N-1) \times (N-1)\), even though there are \( N + 1 \) nodes in the discretization on this domain.

(c-MATLAB) We can pursue the actual eigenvalues using the \texttt{eig} function in MATLAB. We choose to divide the domain \( 0.0 \leq x \leq L \) into 100 segments. Specifically, the statements

\[
\begin{align*}
 n &= 99; \\
 A &= \text{zeros}(n,n); \\
 \text{for } i=1:n \ A(i,i) &= -2; \ \text{end} \\
 \text{for } i=1:n-1 \ A(i,i+1) &= 1; \ \text{end} \\
 \text{for } i=1:n-1 \ A(i+1,i) &= 1; \ \text{end} \\
 \lambda &= \text{eig}(A); \\
 \text{tmp} &= \text{sort}(\sqrt{-\lambda}); \\
 \text{format long} \\
 \text{tmp}(1) &\quad \text{ans} = \quad 0.031414634623641 \\
 \text{format short} \\
 k_n &\quad \text{tmp}(1:20)/\text{tmp}(1); \\
 k_n &\quad \text{tmp}' \\
 \text{ans} &= \begin{array}{cccccccc}
 16.7994 & 17.7619 & 18.7200 & 19.6734 \\
\end{array}
\]

find and display the ratios \( k_n/k_1 \) for the first twenty eigenvalues. As expected, these values are essentially integer multiples of the first one, though this solution is approximate, and the values determined fall increasingly short of the expected results as \( n \) increases. Note that, as here determined,

\[ k_1 = \sqrt{-\lambda_1} \frac{N}{L} = 0.031414634623641 \frac{100}{L} = \frac{3.1414634623641}{L} \]

differs from the the exact result \( k_1 = \pi/L = 3.1415926535/L \) determined in part (a) by less than 0.00014/L (0.0044%).

A graph of these ratios versus \( n \) underscores even more broadly how closely the approximate values obtained via finite differences agree with the exact values. That graph is produced with the statements

\[
\begin{align*}
 x &= [1:20]; \\
 \text{plot( } x, \text{ knk1, 'color','black', 'linewidth',3 }) \\
 \text{grid on} \\
 \text{set(gca, 'fontsize', 12)} \\
 \text{xlabel('n','fontsize',16); ylabel('k_n / k_1', 'fontsize',16)} \\
 \text{hold on} \\
 \text{plot( } x, x, 'color', 'black', 'linewidth',3 ) \\
 \text{hold off}
\]

and is shown in Fig. E15.3. The upper graph shows \( k_n/k_1 = n \), the expected exact result. The graph of the approximate results in this exercise falls a bit below the exact graph, but—to the resolution of the graph—that difference is barely noticeable until about \( n = 17 \).
Figure E15.3: Figure for Exercise 15.7. The upper graph shows the exact eigenvalues determined in part \( \text{(a)} \); the lower graph shows the approximate eigenvalues determined in part \( \text{(c)} \). This graph was produced with MATLAB.
15.8 FEM 1D Homogeneous Helmholtz Equation

Exercise: Consider the differential equation and boundary conditions

\[ \frac{d^2u}{dx^2} + k^2u = 0 \quad ; \quad u(0) = u(L) = 0 \]

(a) Find the analytic solution to this problem and identify the special values of \( k \) that permit non-trivial solutions.

(b) Show that the finite element approach to the problem leads to the generalized eigenvalue problem \( Au = -k^2 L^2 Bu / 6N^2 \) where \( N \) is the number of equal-length elements into which the length \( L \) of the domain is divided and

\[
A = \begin{pmatrix}
-2 & 1 & 0 & \cdots & 0 & 0 & 0 \\
1 & -2 & 1 & \cdots & 0 & 0 & 0 \\
0 & 1 & -2 & \cdots & 0 & 0 & 0 \\
\vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\
0 & 0 & 0 & \cdots & -2 & 1 & 0 \\
0 & 0 & 0 & \cdots & 1 & -2 & 1 \\
0 & 0 & 0 & \cdots & 0 & 1 & -2 \\
\end{pmatrix} \quad ; \quad B = \begin{pmatrix}
4 & 1 & 0 & \cdots & 0 & 0 & 0 \\
1 & 4 & 1 & \cdots & 0 & 0 & 0 \\
0 & 1 & 4 & \cdots & 0 & 0 & 0 \\
\vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\
0 & 0 & 0 & \cdots & 4 & 1 & 0 \\
0 & 0 & 0 & \cdots & 1 & 4 & 1 \\
0 & 0 & 0 & \cdots & 0 & 1 & 4 \\
\end{pmatrix}
\]

and

\[
u = \begin{pmatrix}
u_1 \\
u_2 \\
u_3 \\
\vdots \\
u_{N-3} \\
u_{N-2} \\
u_{N-1}
\end{pmatrix}
\]

Note that, when the domain \( 0 \leq x \leq L \) is divided into \( N \) segments, there will be \( N+1 \) nodes ranging from \( x_0 = 0 \) to \( x_{N+1} = L \). Because, along the way to a solution, the boundary conditions result in the rows and columns associated with these two nodes being deleted, these matrices will have only \( N-1 \) rows and columns.

(c) Taking \( N = 100 \), use an available numeric processing program like IDL, MATLAB, OCTAVE, or PYTHON to find the first several eigenvalues \( k_n \) and compare your results with the values found in part (a).

Solution: (a) The general solution to the differential equation is well known to be

\[ u(x) = A \sin kx + B \cos kx \]

The boundary conditions require that

\[ u(0) = B = 0 \quad ; \quad u(L) = A \sin kL = 0 \implies kL = n\pi, n = 1, 2, 3, \ldots \]

so there is a family of solutions

\[ u(x) = A \sin \left( \frac{n\pi x}{L} \right) = A \sin k_n x \quad ; \quad k_n = nk_1 = n \frac{\pi}{L} \]

The non-trivial solutions are sinusoidal with arbitrary amplitude, and the wave number \( k_n \) associated with the \( n \)-th solution is an integer multiple of the wave number \( k_1 = \pi / L \) associated with the first solution.

(b) We begin by identifying the values of the parameters in Eq. (15.67) that will reduce that general equation to the equation of current interest. Specifically, we must identify \( \alpha = -1, \beta = k^2, \) and \( f = 0 \). Further, to yield the boundary conditions in this exercise, we must set \( p = 0, \gamma = 0, \) and \( q = 0. \)
Following the pattern laid out in Section 15.9, we begin by discretizing the domain \(0 \leq x \leq L\) into equally-length elements whose end points lie at

\[
x_i = i \left( \frac{L}{N} \right), \quad i = 0, 1, 2, \ldots, N
\]

so that, for the \(e\)-th element,

\[
x_1^{(e)} = x_e; \quad x_2^{(e)} = x_e + L/N, \quad e = 0, 1, 2, \ldots, N - 1
\]

With these choices, the interpolation functions in Section 15.9.2 reduce to

\[
N_1^{(e)}(x) = \frac{x_2^{(e)} - x}{L/N}; \quad N_2^{(e)}(x) = \frac{x - x_1^{(e)}}{L/N}
\]

and the approximate solution over the \(e\)-th element is given by

\[
\tilde{u}^{(e)}(x) = \sum_{j=1}^{2} N_1^{(e)}(x) \tilde{u}_j^{(e)}
\]

where the \(\tilde{u}_j^{(e)}\) s for \(j = 1, 2\) are constants to be determined. The residual \(r(x)\) as defined in Section 15.9.3 is for this exercise given by

\[
r(x) = \frac{d^2 u}{dx^2} + k^2 u
\]

The weighted averages of this residual over the \(e\)-th element when the weights are the interpolation functions then are defined by

\[
R_i^{(e)} = \int_{x_1^{(e)}}^{x_2^{(e)}} N_i^{(e)} \left( \frac{d^2 \tilde{u}^{(e)}}{dx^2} + k^2 \tilde{u}^{(e)} \right) dx
\]

\[
= - \int_{x_1^{(e)}}^{x_2^{(e)}} \left( \frac{dN_i^{(e)}}{dx} \frac{d\tilde{u}^{(e)}}{dx} - k^2 N_i^{(e)} \tilde{u}^{(e)} \right) dx + \left. \left( N_i^{(e)} \frac{d\tilde{u}^{(e)}}{dx} \right) \right|_{x_1^{(e)}}^{x_2^{(e)}} \quad i = 1, 2
\]

Here, the second form is calculated by evaluating the integral of the second derivative in the first form by parts. Substituting from the above equation for \(u^{(e)}(x)\), we next learn that

\[
R_i^{(e)} = - \left[ \sum_{j=1}^{2} R_j^{(e)} \int_{x_1^{(e)}}^{x_2^{(e)}} \left( \frac{dN_j^{(e)}}{dx} \frac{dN_i^{(e)}}{dx} - k^2 N_i^{(e)} N_j^{(e)} \right) dx \right] + \left. \left( N_i^{(e)} \frac{d\tilde{u}^{(e)}}{dx} \right) \right|_{x_1^{(e)}}^{x_2^{(e)}}
\]

In matrix form, these several equations for \(1 \leq i, j \leq 2\) have the form

\[
\{ R^{(e)} \} = [K^{(e)}] \{ \tilde{u}^{(e)} \} + \{ g^{(e)} \}
\]

or more compactly as

\[
\{ R^{(e)} \} = [K^{(e)}] \{ \tilde{u}^{(e)} \} + \{ g^{(e)} \}
\]

where

\[
K_{ij}^{(e)} = - \int_{x_1^{(e)}}^{x_2^{(e)}} \left( \frac{dN_i^{(e)}}{dx} \frac{dN_j^{(e)}}{dx} - k^2 N_i^{(e)} N_j^{(e)} \right) dx
\]

and

\[
g_i^{(e)} = \left. \left( N_i^{(e)} \frac{d\tilde{u}^{(e)}}{dx} \right) \right|_{x_1^{(e)}}^{x_2^{(e)}}
\]

\[
= \left. \left( N_i^{(e)} \frac{d\tilde{u}^{(e)}}{dx} \right) \right|_{x=x_1^{(e)}} - \left. \left( N_i^{(e)} \frac{d\tilde{u}^{(e)}}{dx} \right) \right|_{x=x_2^{(e)}}
\]
We have several integrals to evaluate. Because \( dN_1^{(e)} / dx = -L/N \) and \( dN_2^{(e)} / dx = L/N \) or, more compactly \( dN_k^{(e)} = (-1)^k L/N \), we conclude quickly that

\[
\int_{x_1^{(e)}}^{x_2^{(e)}} \frac{dN_1^{(e)}}{dx} \frac{dN_j^{(e)}}{dx} \, dx = \int_{x_1^{(e)}}^{x_2^{(e)}} \frac{L^3}{N^3} \, (-1)^j \frac{L}{N} \, dx = (-1)^{i+j} \frac{L^2}{N^2} (x_2^{(e)} - x_1^{(e)}) = (-1)^{i+j} \frac{L^3}{N^3}
\]

Next, note that\(^1\)

\[
\int_{x_1^{(e)}}^{x_2^{(e)}} N_1^{(e)} N_1^{(e)} \, dx = \frac{L^2}{N^2} \int_{x_1^{(e)}}^{x_2^{(e)}} (x_2^{(e)} - x)^2 \, dx = \frac{L^5}{3N^5}
\]

\[
\int_{x_1^{(e)}}^{x_2^{(e)}} N_2^{(e)} N_2^{(e)} \, dx = \frac{L^2}{N^2} \int_{x_1^{(e)}}^{x_2^{(e)}} (x - x_1^{(e)})^2 \, dx = \frac{L^5}{3N^5}
\]

\[
\int_{x_1^{(e)}}^{x_2^{(e)}} N_1^{(e)} N_2^{(e)} \, dx = \int_{x_1^{(e)}}^{x_2^{(e)}} N_2^{(e)} N_1^{(e)} \, dx = \frac{L^2}{N^2} \int_{x_1^{(e)}}^{x_2^{(e)}} (x_2^{(e)} - x)(x - x_1^{(e)}) \, dx = \frac{L^5}{6N^5}
\]

(Remember that \( x_2^{(e)} = x_1^{(e)} + L/N \).) We conclude that

\[
\begin{bmatrix}
\mathcal{K}^{(e)}
\end{bmatrix} = - \begin{bmatrix}
\frac{L^3}{N^3} & -\frac{L^3}{N^3} \\
-\frac{L^3}{N^3} & \frac{L^3}{N^3}
\end{bmatrix} + k^2 \begin{bmatrix}
\frac{L^5}{3N^5} & \frac{L^5}{6N^5} \\
\frac{L^5}{6N^5} & \frac{L^5}{3N^5}
\end{bmatrix}
\]

\[
= \frac{L^3}{N^3} \begin{bmatrix}
-1 & 1 \\
1 & -1
\end{bmatrix} + \frac{k^2 L^5}{6N^5} \begin{bmatrix}
2 & 1 \\
1 & 2
\end{bmatrix}
\]

To complete the evaluation, we note that\(^2\)

\[
g_1^{(e)} = \left. \left( N_1^{(e)} \frac{d\tilde{u}}{dx} \right) \right|_{x=x_1^{(e)}} - \left. \left( N_1^{(e)} \frac{d\tilde{u}}{dx} \right) \right|_{x=x_2^{(e)}} = -\tilde{u}'(x_1^{(e)}) = -\tilde{u}'(x_e)
\]

and

\[
g_2^{(e)} = \left. \left( N_2^{(e)} \frac{d\tilde{u}}{dx} \right) \right|_{x=x_2^{(e)}} - \left. \left( N_2^{(e)} \frac{d\tilde{u}}{dx} \right) \right|_{x=x_1^{(e)}} = \tilde{u}'(x_2^{(e)}) = \tilde{u}'(x_{e+1})
\]

so

\[
g^{(e)} = \begin{bmatrix}
-\tilde{u}'(x_e) \\
\tilde{u}'(x_{e+1})
\end{bmatrix}
\]

Setting the average residual on the \( e \)-th element to zero, we conclude that the elemental equation for the \( e \)-th element is

\[
\{ R^{(e)} \} = \left( \frac{L^3}{N^3} \begin{bmatrix}
-1 & 1 \\
1 & -1
\end{bmatrix} + \frac{k^2 L^5}{6N^5} \begin{bmatrix}
2 & 1 \\
1 & 2
\end{bmatrix} \right) \begin{bmatrix}
\tilde{u}_e \\
\tilde{u}_{e+1}
\end{bmatrix} + \begin{bmatrix}
-\tilde{u}'(x_e) \\
\tilde{u}'(x_{e+1})
\end{bmatrix} = \begin{bmatrix}
0 \\
0
\end{bmatrix}
\]

or

\[
\left( \frac{L^3}{N^3} \begin{bmatrix}
-1 & 1 \\
1 & -1
\end{bmatrix} + \frac{k^2 L^2}{6N^2} \begin{bmatrix}
2 & 1 \\
1 & 2
\end{bmatrix} \right) \begin{bmatrix}
\tilde{u}_e \\
\tilde{u}_{e+1}
\end{bmatrix} = \frac{N^3}{L^3} \begin{bmatrix}
-\tilde{u}'(x_e) \\
\tilde{u}'(x_{e+1})
\end{bmatrix}
\]

or, with \( \beta = k^2 L^2 / 6N^2 \),

\[
\begin{bmatrix}
-1 + 2\beta & 1 + \beta \\
1 + \beta & -1 + 2\beta
\end{bmatrix} \begin{bmatrix}
\tilde{u}_e \\
\tilde{u}_{e+1}
\end{bmatrix} = \frac{N^3}{L^3} \begin{bmatrix}
-\tilde{u}'(x_e) \\
\tilde{u}'(x_{e+1})
\end{bmatrix}
\]

\(^1\)These integrals are simple enough to be evaluated by hand. Alternatively, they can be evaluated with an available symbolic manipulating program like MAXIMA, MAPLE, or Mathematica. The details of those evaluations are omitted.

\(^2\)Remember that \( N_1^{(e)}(x_1^{(e)}) = 1, N_1^{(e)}(x_2^{(e)}) = 0, N_2^{(e)}(x_1^{(e)}) = 0, \) and \( N_2^{(e)}(x_2^{(e)}) = 1.\)
As guidance for inferring the general case, we write out these equation for a system in which there are six nodes \((N = 6, i = 0, 1, 2, 3, 4, 5)\) and five elements \(e = 0, 1, 2, 3, 4\). For \(e = 0\), the equation, expanded to involve \(6 \times 6\) matrices and six-element vectors, is

\[
\begin{bmatrix}
-1 + 2\beta & 1 + \beta & 0 & 0 & 0 & 0 \\
1 + \beta & -1 + 2\beta & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 \\
\end{bmatrix}
\begin{bmatrix}
\tilde{u}_0 \\
\tilde{u}_1 \\
\tilde{u}_2 \\
\tilde{u}_3 \\
\tilde{u}_4 \\
\tilde{u}_5 \\
\end{bmatrix}
= \frac{N^3}{L^3}
\begin{bmatrix}
-\tilde{u}'(x_0) \\
\tilde{u}'(x_1) \\
0 \\
0 \\
0 \\
0 \\
0 \\
\end{bmatrix}
\]

For \(e = 1\),

\[
\begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 0 \\
0 & -1 + 2\beta & 1 + \beta & 0 & 0 & 0 \\
0 & 1 + \beta & -1 + 2\beta & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 \\
\end{bmatrix}
\begin{bmatrix}
\tilde{u}_0 \\
\tilde{u}_1 \\
\tilde{u}_2 \\
\tilde{u}_3 \\
\tilde{u}_4 \\
\tilde{u}_5 \\
\end{bmatrix}
= \frac{N^3}{L^3}
\begin{bmatrix}
0 \\
-\tilde{u}'(x_1) \\
\tilde{u}'(x_2) \\
0 \\
0 \\
0 \\
0 \\
\end{bmatrix}
\]

For \(e = 2\),

\[
\begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & -1 + 2\beta & 1 + \beta & 0 & 0 \\
0 & 0 & 1 + \beta & -1 + 2\beta & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 \\
\end{bmatrix}
\begin{bmatrix}
\tilde{u}_0 \\
\tilde{u}_1 \\
\tilde{u}_2 \\
\tilde{u}_3 \\
\tilde{u}_4 \\
\tilde{u}_5 \\
\end{bmatrix}
= \frac{N^3}{L^3}
\begin{bmatrix}
0 \\
0 \\
-\tilde{u}'(x_2) \\
\tilde{u}'(x_3) \\
0 \\
0 \\
0 \\
\end{bmatrix}
\]

For \(e = 3\)

\[
\begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & -1 + 2\beta & 1 + \beta & 0 \\
0 & 0 & 0 & 1 + \beta & -1 + 2\beta & 0 \\
0 & 0 & 0 & 0 & 0 & 0 \\
\end{bmatrix}
\begin{bmatrix}
\tilde{u}_0 \\
\tilde{u}_1 \\
\tilde{u}_2 \\
\tilde{u}_3 \\
\tilde{u}_4 \\
\tilde{u}_5 \\
\end{bmatrix}
= \frac{N^3}{L^3}
\begin{bmatrix}
0 \\
0 \\
0 \\
-\tilde{u}'(x_3) \\
\tilde{u}'(x_4) \\
0 \\
0 \\
\end{bmatrix}
\]

We have also rearranged the terms a bit to reduce the "width" of the equation.
For \( e = 4 \)

\[
\begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & -1 + 2\beta & 1 + \beta & 0 \\
0 & 0 & 0 & 1 + \beta & -1 + 2\beta & 0 \\
\end{bmatrix}
\begin{bmatrix}
\tilde{u}_0 \\
\tilde{u}_1 \\
\tilde{u}_2 \\
\tilde{u}_3 \\
\tilde{u}_4 \\
\tilde{u}_5 \\
\end{bmatrix}
= \frac{N^3}{L^3}
\begin{bmatrix}
0 \\
0 \\
0 \\
-\tilde{u}'(x_4) \\
\tilde{u}'(x_5) \\
\end{bmatrix}
\]

The next step is to add these equations to find the single matrix equation

\[
\begin{bmatrix}
-1 + 2\beta & 1 + \beta & 0 & 0 & 0 & 0 \\
1 + \beta & -2 + 4\beta & 1 + \beta & 0 & 0 & 0 \\
0 & 1 + \beta & -2 + 4\beta & 1 + \beta & 0 & 0 \\
0 & 0 & 1 + \beta & -2 + 4\beta & 1 + \beta & 0 \\
0 & 0 & 0 & 1 + \beta & -1 + 2\beta & 0 \\
0 & 0 & 0 & 0 & 1 & 0 \\
\end{bmatrix}
\begin{bmatrix}
\tilde{u}_0 \\
\tilde{u}_1 \\
\tilde{u}_2 \\
\tilde{u}_3 \\
\tilde{u}_4 \\
\tilde{u}_5 \\
\end{bmatrix}
= \frac{N^3}{L^3}
\begin{bmatrix}
-\tilde{u}'(x_0) \\
0 \\
0 \\
0 \\
\tilde{u}'(x_5) \\
0 \\
\end{bmatrix}
\]

Finally, we have to incorporate the boundary conditions, which essentially require that \( \tilde{u}_0 = \tilde{u}_5 = 0 \). We change the first row and last rows in the coefficient matrix to reflect these conditions:

\[
\begin{bmatrix}
1 & 0 & 0 & 0 & 0 & 0 \\
1 + \beta & -2 + 4\beta & 1 + \beta & 0 & 0 & 0 \\
0 & 1 + \beta & -2 + 4\beta & 1 + \beta & 0 & 0 \\
0 & 0 & 1 + \beta & -2 + 4\beta & 1 + \beta & 0 \\
0 & 0 & 0 & 1 + \beta & -2 + 4\beta & 1 \\
0 & 0 & 0 & 0 & 1 & 0 \\
\end{bmatrix}
\begin{bmatrix}
\tilde{u}_0 \\
\tilde{u}_1 \\
\tilde{u}_2 \\
\tilde{u}_3 \\
\tilde{u}_4 \\
\tilde{u}_5 \\
\end{bmatrix}
= \frac{N^3}{L^3}
\begin{bmatrix}
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
\end{bmatrix}
\]

Further, since the second element in the first column of the coefficient matrix is multiplied by \( \tilde{u}_0 = 0 \) and the next to last element in the last column of the coefficient matrix is multiplied by \( \tilde{u}_5 = 0 \), we can replace each of those elements by 0. Further, since the right hand side of this equation is entirely zero, we can remove the factor \( N^3/L^3 \). We find that

\[
\begin{bmatrix}
1 & 0 & 0 & 0 & 0 & 0 \\
0 & -2 + 4\beta & 1 + \beta & 0 & 0 & 0 \\
0 & 1 + \beta & -2 + 4\beta & 1 + \beta & 0 & 0 \\
0 & 0 & 1 + \beta & -2 + 4\beta & 1 + \beta & 0 \\
0 & 0 & 0 & 1 + \beta & -2 + 4\beta & 0 \\
0 & 0 & 0 & 0 & 1 & 0 \\
\end{bmatrix}
\begin{bmatrix}
\tilde{u}_0 \\
\tilde{u}_1 \\
\tilde{u}_2 \\
\tilde{u}_3 \\
\tilde{u}_4 \\
\tilde{u}_5 \\
\end{bmatrix}
= \begin{bmatrix}
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
\end{bmatrix}
\]
Recast in the form
\[
\begin{bmatrix}
1 & 0 & 0 & 0 & 0 \\
0 & -2 & 1 & 0 & 0 \\
0 & 1 & -2 & 1 & 0 \\
0 & 0 & 1 & -2 & 0 \\
0 & 0 & 0 & 0 & 1 \\
\end{bmatrix}
\begin{bmatrix}
\tilde{u}_0 \\
\tilde{u}_1 \\
\tilde{u}_2 \\
\tilde{u}_3 \\
\tilde{u}_4 \\
\tilde{u}_5 \\
\end{bmatrix} = -\beta
\begin{bmatrix}
0 & 0 & 0 & 0 & 0 \\
0 & 4 & 1 & 0 & 0 \\
0 & 1 & 4 & 1 & 0 \\
0 & 0 & 0 & 1 & 4 \\
0 & 0 & 0 & 0 & 1 \\
\end{bmatrix}
\begin{bmatrix}
\hat{u}_0 \\
\hat{u}_1 \\
\hat{u}_2 \\
\hat{u}_3 \\
\hat{u}_4 \\
\hat{u}_5 \\
\end{bmatrix}
\]
we conclude that the problem is actually a bit simpler, since we already know \(\tilde{u}_0\) and \(\tilde{u}_5\). We can therefore delete the first and last rows and columns to find the essential problem we must solve to be
\[
\begin{bmatrix}
2 & -1 & 0 & 0 \\
-1 & 2 & -1 & 0 \\
0 & -1 & 2 & -1 \\
0 & 0 & -1 & 2 \\
\end{bmatrix}
\begin{bmatrix}
\tilde{u}_1 \\
\tilde{u}_2 \\
\tilde{u}_3 \\
\tilde{u}_4 \\
\end{bmatrix} = -\beta
\begin{bmatrix}
4 & 1 & 0 & 0 \\
1 & 4 & 1 & 0 \\
0 & 1 & 4 & 1 \\
0 & 0 & 1 & 4 \\
\end{bmatrix}
\begin{bmatrix}
\hat{u}_1 \\
\hat{u}_2 \\
\hat{u}_3 \\
\hat{u}_4 \\
\end{bmatrix}
\]
We can now readily extend this result to apply to an arbitrary number of elements by introducing the quantities\(^4\)
\[
A = \begin{pmatrix}
-2 & 1 & 0 & \cdots & 0 & 0 & 0 \\
1 & -2 & 1 & \cdots & 0 & 0 & 0 \\
0 & 1 & -2 & \cdots & 0 & 0 & 0 \\
\vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\
0 & 0 & 0 & \cdots & -2 & -1 & 0 \\
0 & 0 & 0 & \cdots & -1 & 2 & -1 \\
0 & 0 & 0 & \cdots & 0 & -1 & 2 \\
\end{pmatrix}
\quad ; \quad B = \begin{pmatrix}
4 & 1 & 0 & \cdots & 0 & 0 & 0 \\
1 & 4 & 1 & \cdots & 0 & 0 & 0 \\
0 & 1 & 4 & \cdots & 0 & 0 & 0 \\
\vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\
0 & 0 & 0 & \cdots & 4 & 1 & 0 \\
0 & 0 & 0 & \cdots & 1 & 4 & 1 \\
0 & 0 & 0 & \cdots & 0 & 1 & 4 \\
\end{pmatrix}
\]
and
\[
u = \begin{pmatrix}
u_1 \\
u_2 \\
u_3 \\
\vdots \\
u_{N-3} \\
u_{N-2} \\
u_{N-1} \\
\end{pmatrix}
\]
In terms of these quantities, the problem we must solve is the generalized eigenvalue problem
\[
Au = -\beta Bu = \frac{k^2 L^2}{6 N^2} Bu \implies k_n = \sqrt{6\beta_n} \frac{N}{L} \implies \frac{k_n}{k_1} = \sqrt{\beta_n \beta_1}
\]
where \(A\) and \(B\) are matrices, \(u\) is a vector of the unknown values at the nodes 1 to \(N - 1\), and \(\beta = k^2 L^2 / 6 N^2\) is the eigenvalue to be found. At this point, we hope that the values of \(\beta_n\) are positive. Note that these matrices have dimension \((N - 1) \times (N - 1)\), even though there are \(N + 1\) nodes in the discretization on this domain.

\(^4\)We now drop each tilde, understanding that \(u\) now stands for the approximate solution.
15.9 1-D Inhomogeneous Helmholtz Equation

**Exercise:** When $\alpha$, $\beta$, and $f$ are constants, Eqs. (15.67), (15.68), and (15.69) can be solved analytically. Show, for example, that the solution to this boundary value problem is given by

$$\varphi(x) = A \sin \lambda x + \left( p - \frac{f}{\beta} \right) \cos \lambda x + \frac{f}{\beta}$$

where

$$A = \frac{(q - \gamma f/\beta) + (p - f/\beta)(\alpha \lambda \sin \lambda L - \gamma \cos \lambda L)}{\alpha \lambda \cos \lambda L + \gamma \sin \lambda L}$$

and $\lambda = \sqrt{-\beta/\alpha}$ when $\beta/\alpha < 0$. Using graphical displays in particular, compare the analytic solution in this exercise with the solution obtained numerically by finite difference and finite element approaches. **Optional:** Find corresponding solutions when $\beta/\alpha = 0$ and $\beta/\alpha > 0$.

**Solution:** Assuming that $\alpha$, $\beta$, and $f$ are constants, we can rewrite Eqs. (15.67), (15.68), and (15.69) as

$$-\alpha \frac{d^2 \varphi(x)}{dx^2} + \beta \varphi(x) = f \quad (E15.9.1)$$

where $\alpha'$ is zero,

$$\varphi(0) = p \quad (E15.9.2)$$

and

$$\left. \frac{d \varphi(x)}{dx} + \gamma \varphi(x) \right|_{x=L} = q \quad (E15.9.3)$$

If we divide the differential equation by $-\alpha$, and let $\lambda = \sqrt{-\beta/\alpha}$ or, equivalently, $\lambda^2 = -\beta/\alpha$ (assuming that $\beta/\alpha < 0$), then the differential equation becomes

$$\frac{d^2 \varphi(x)}{dx^2} + \lambda^2 \varphi(x) = -\frac{f}{\alpha} \quad (E15.9.4)$$

which is an inhomogeneous Helmholtz equation in one dimension.

We know that the general solution to this inhomogeneous equation consists of the sum of a general solution to the homogeneous equation ($f = 0$) and a particular solution which, when substituted into the equation, yields the inhomogeneity. While the independent variable is a space variable rather than time, the homogeneous equation is basically the equation for simple harmonic oscillation and has the general solution

$$\varphi(x) = A \sin(\lambda x) + B \cos(\lambda x) \quad (E15.9.5)$$

We find the particular solution by noting that, if we substitute $\varphi = C$, where $C$ is an as yet undetermined constant, the differential equation Eq. (E15.9.4) becomes

$$\lambda^2 C = -\frac{f}{\alpha} \quad \implies \quad C = -\frac{f}{\lambda^2 \alpha} = \frac{f}{\beta} \quad (E15.9.6)$$

Thus, having identified the particular solution $\varphi_p = f/\beta$, we conclude that the general solution to the inhomogeneous equation is

$$\varphi(x) = A \sin(\lambda x) + B \cos(\lambda x) + \frac{f}{\beta} \quad (E15.9.7)$$

Now we are left with the task of using the boundary conditions to solve for the constants $A$ and $B$. We first use Eq. (E15.9.2) to find that

$$\varphi(0) = B + \frac{f}{\beta} = p \quad \text{or} \quad B = p - \frac{f}{\beta} \quad (E15.9.8)$$
Then we use Eq. (E15.9.3) to find $A$. We substitute the solution for $\varphi(x)$ from Eq. (E15.9.7) to find that
\[
\left[ \alpha \left( A\lambda \cos(\lambda x) - B\lambda \sin(\lambda x) \right) + \gamma \left( A\sin(\lambda x) + B\cos(\lambda x) + \frac{f}{\beta} \right) \right] \bigg|_{x=L} = q \quad (E15.9.9)
\]
If we substitute $x = L$, rearrange the equation to solve for $A$, and substitute the result for $B$ from Eq. (E15.9.8), we find that
\[
A = \frac{(q - \gamma f/\beta) + (p - f/\beta)(\alpha\lambda\sin \lambda L - \gamma \cos \lambda L)}{\alpha\lambda \cos \lambda L + \gamma \sin \lambda L} \quad (E15.9.10)
\]
The final solution to Eq. (E15.9.1), is
\[
\varphi(x) = \frac{(q - \gamma f/\beta) + (p - f/\beta)(\alpha\lambda\sin \lambda L - \gamma \cos \lambda L)}{\alpha\lambda \cos \lambda L + \gamma \sin \lambda L} \sin \lambda x + \left( p - \frac{f}{\beta} \right) \cos \lambda x + \frac{f}{\beta} \quad (E15.9.11)
\]
As a test of this solution (and to compare the analytic solution with the approximate solutions
developed in the text by finite element and finite difference methods), we execute the following
MATLAB statements. First, we regenerate the solution we deduced in Section 15.4.2 of the text for
\( N = 100 \) with the statements

\[
\text{fdm1d}
\]

Enter number of segments (N): 100
Enter alpha: -4.0
Enter beta: 3.0
Enter f: 0.0
Enter L: 10.0
Enter p: 0.0
Enter gamma: 0.0
Enter q: -4.0

\( x_{\text{fdm}} = x \); \( \phi_{\text{fdm}} = \phi \);

\( \text{plot( } x_{\text{fdm}}, \phi_{\text{fdm}}, 'color', 'black', 'linewidth', 2 ) } \)

\( \text{grid on} \)

\( \text{set( gca, 'fontsize', 12) } \)

\( \text{xlabel('x', 'fontsize', 16); ylabel('
\phi', 'fontsize', 16) } \)

Second, we generate and overplot the solution for the same parameters via the finite difference
method with the statements

\[
\text{fem1d}
\]

Enter number of segments (N): 100
Enter alpha: -4.0
Enter beta: 3.0
Enter f: 0.0
Enter l: 0.1
Enter p: 0.0
Enter gamma: 0.0
Enter q: -4.0

\( x_{\text{fem}} = x \); \( \phi_{\text{fem}} = \phi \);

\( \text{hold on} \)

\( \text{plot( } x_{\text{fem}}, \phi_{\text{fem}}, 'color', 'black', 'linewidth', 3) } \)

(Remember that this command file asked for the length of each segment rather than the length of
the string.)

Finally, we generate values of \( x \) for evaluating the exact solution, evaluate that solution, and
overplot it for the same parameters with the statements

\[
\begin{align*}
\text{x_exact} &= [0 : 0.1 : 10]; \\
\lambda &= \sqrt{-\beta/\alpha}; \\
\theta &= \lambda L; \\
A\number &= q - \gamma f/\beta + B \left( \alpha \lambda \sin(\theta) - \gamma \cos(\theta) \right); \\
A &= A\number/(\alpha \lambda \cos(\theta) + \gamma \sin(\theta)); \\
\phi_{\text{exact}} &= A \sin(\lambda x) + B \cos(\lambda x) + f/\beta; \\
\text{plot( } x_{\text{exact}}, \phi_{\text{exact}}, 'color', 'black', 'linewidth', 4) 
\end{align*}
\]

(Remember that the execution of \texttt{fdm1d.m} left the proper values already assigned to the several
parameters.) The resulting graph is shown in Fig. E15.4. Though it is difficult to discern, the exact
solution falls right on top of the approximate solutions worked out in the text.\(^5\)

\(^5\)To be convinced of this assertion, you will have to watch the generation of the graphs on the screen. Using the
Figure E15.4: Comparison of exact and approximate solutions to the differential equation addressed in Exercise 15.9. This graph was produced with MATLAB.

To assess the agreement more quantitatively, we might simply evaluate and plot the differences with the statements:

\[ \text{fdmexact} = \text{phifdm} - \text{phi_exact}'; \]
\[ \text{femexact} = \text{phifem} - \text{phi_exact}'; \]
\[ \text{plot}(\text{xfdm, fdmexact, 'color', 'black', 'linewidth', 2}) \]
\[ \text{hold on} \]
\[ \text{plot}(\text{xfem, femexact, 'color', 'black', 'linestyle', '--', 'linewidth', 2}) \]
\[ \text{set(gca, 'fontsize', 12)} \]
\[ \text{grid on} \]
\[ \text{xlabel('x', 'fontsize', 16)}; \text{ylabel('difference', 'fontsize', 16)} \]

The graph is shown in Fig. E15.5. It appears as if, over the range \(0 \leq x \leq 10\), the approximate solutions differ from the exact solution by no more than about \(\pm 0.006\), a difference well within the resolution of Fig. E15.4. Since the solution itself has a maximum (absolute) value of about 1.6, this difference amounts to an error of about 0.006/1.6 = 0.00375, i.e., about 0.4%. Intriguingly, it appears that, for this situation, the finite difference solution tends to overestimate the solution when the finite element solution underestimates it, and vice versa. The statements

\[ \text{hold off} \]

different line thicknesses helps. Draw the FDM solution first with \(\text{linewidth} = 2\), then overplot the FEM solution with \(\text{linewidth} = 3\), and finally, overplot the exact solution with \(\text{linewidth} = 4\). Watch as each thicker line is added to the graph.

\[ ^6 \text{Note that } \text{xfdm, xfem, and x_exact are, in fact the same, so the values of the independent variables at which each solution is obtained are the same, thus justifying the direct subtraction of one solution from another.} \]

\[ ^7 \text{We also need to transpose } \text{phi_exact because we have created it as a row vector but } \text{phifdm and phifem have been created as column vectors.} \]
Figure E15.5: Difference between exact and approximate solutions to the differential equation addressed in Exercise 15.9. The solid line conveys the difference $\varphi_{fdm} - \varphi_{exact}$. The dashed line conveys the difference $\varphi_{fem} - \varphi_{exact}$. This graph was produced with MATLAB.

$$\text{erroravg} = \frac{\text{phifdm} + \text{phifem}}{2.0} - \text{phi_exact}';$$

```matlab
plot(xfdm, erroravg, 'color', 'black', 'linewidth', 2)
grid on
max(abs(erroravg))
ans = 0.0010
```

produce a graph of the difference between the average of the two approximate solutions and the exact solution, though the graph is not here shown. They also evaluate the maximum of the absolute value of the difference between that average and the exact solution, which is about one sixth of the maximum difference between either approximate solution and the exact solution.
Optional Part

To address the optional part of this exercise, we look next at what happens when $\beta/\alpha = 0$. The differential equation then becomes

$$\frac{d^2 \varphi(x)}{dx^2} = -\frac{f}{\alpha}$$

and the general solution, which we can obtain simply by integrating the equation twice and inserting appropriate integration constants, is

$$\varphi(x) = -\frac{fx^2}{2\alpha} + Ax + B$$

(E15.9.13)

Once again we use the boundary conditions to solve for $A$ and $B$.

$$\varphi(0) = B = p$$

(E15.9.14)

$$\left[ \alpha \frac{d\varphi(x)}{dx} + \gamma \varphi(x) \right]_{x=L} = \alpha \left( -\frac{fL}{\alpha} + A \right) + \gamma \left( -\frac{fL^2}{2\alpha} + AL + B \right) = q$$

(E15.9.15)

$$A = \frac{q - p\gamma + fL + \frac{fL^2\gamma}{2\alpha}}{\alpha + \gamma L}$$

(E15.9.16)

So the final solution for the case $\beta/\alpha = 0$ is

$$\varphi(x) = -\frac{fx^2}{2\alpha} + \frac{q - p\gamma + fL + \frac{fL^2\gamma}{2\alpha}}{\alpha + \gamma L}x + p$$

(E15.9.17)

Finally, we deduce the solution when $\beta/\alpha > 0$. In this case, we would set $\xi^2 = \beta/\alpha$ and write the differential equation in the form

$$\frac{d^2 \varphi(x)}{dx^2} - \xi^2 \varphi(x) = -\frac{f}{\alpha}$$

(E15.9.18)

The solution to the homogeneous equation this time is a linear combination of the exponentials $e^{\xi x}$ and $e^{-\xi x}$ and a suitable particular solution is still $\phi_p = f/\beta$. Thus, the general solution to this equation is

$$\varphi(x) = Ae^{\xi x} + Be^{-\xi x} + \frac{f}{\beta}$$

(E15.9.19)

Once again we use the boundary conditions to find that the constants $A$ and $B$ must satisfy both

$$\varphi(0) = A + B + \frac{f}{\beta} = p \quad \implies \quad A + B = p - \frac{f}{\beta}$$

(E15.9.20)

and

$$\left[ \alpha(A\xi e^{\xi x} - B\xi e^{-\xi x}) + \gamma(A\xi e^{\xi x} + Be^{-\xi x} + \frac{f}{\beta}) \right]_{x=L} = q$$

$$\implies \quad (\gamma + \alpha\xi)e^{\xi L}A + (\gamma - \alpha\xi)e^{-\xi L}B = q - \frac{\gamma f}{\beta}$$

(E15.9.21)

Solved for $A$ and $B$, these two equations, yield the results

$$A = \frac{q - \gamma f/\beta + e^{-\xi L}\left((\alpha\xi - \gamma)(p - f/\beta)\right)}{(\alpha\xi + \gamma)e^{\xi L} + (\alpha\xi - \gamma)e^{-\xi L}}$$

(E15.9.22)

and

$$B = \frac{-\left(q - \gamma f/\beta\right) + e^{\xi L}\left((\alpha\xi + \gamma)(p - f/\beta)\right)}{(\alpha\xi + \gamma)e^{\xi L} + (\alpha\xi - \gamma)e^{-\xi L}}$$

(E15.9.23)
With these values, the final solution for the case $\beta/\alpha > 0$ is

$$
\varphi(x) = \frac{q - \gamma f/\beta + e^{-\xi L} \left( (\alpha \xi - \gamma) (p - f/\beta) \right)}{(\alpha \xi + \gamma) e^{\xi L} + (\alpha \xi - \gamma) e^{-\xi L}} e^{\xi x} \\
+ \frac{-(q - \gamma f/\beta) + e^{\xi L} \left( (\alpha \xi + \gamma) (p - f/\beta) \right)}{(\alpha \xi + \gamma) e^{\xi L} + (\alpha \xi - \gamma) e^{-\xi L}} e^{-\xi x} + \frac{f}{\beta}
$$

(E15.9.24)
15.12 Assembly for Three-Element Linear System

Exercise: Continuing with the circumstances of Exercises 15.10 and 15.11, suppose the region of interest is divided into three three-noded elements, \( e = 1, 2, 3 \) with global nodes 1, 2, 3, 4, 5, 6, 7, nodes 1, 2, and 3 in element 1, nodes 3, 4, and 5 in element 2, and nodes 5, 6, and 7 in element 3. Suppose node 2 is midway between nodes 1 and 3, node 4 is midway between nodes 3 and 5, and node 6 is midway between nodes 5 and 7, but do not suppose the lengths of the three elements are the same. Following the pattern in Sections 15.9.4 and 15.9.5, assemble the elemental equations for these three elements into an equation for the whole system analogous to the last equation in Section (15.9.5) if the solution is required to satisfy the boundary conditions of Eqs. (15.68) and (15.69).

Solution: The specific geometry of the system in this exercise is clarified in Fig. E15.6. The system of interest has three elements and seven distinct nodes.

In Exercise 15.11, we found that the elemental equation applicable to a three-noded linear element \( (e) \) is

\[
\begin{bmatrix}
K_{11}^{(e)} & K_{12}^{(e)} & K_{13}^{(e)} \\
K_{21}^{(e)} & K_{22}^{(e)} & K_{23}^{(e)} \\
K_{31}^{(e)} & K_{32}^{(e)} & K_{33}^{(e)}
\end{bmatrix}
\begin{bmatrix}
\varphi_1^{(e)} \\
\varphi_2^{(e)} \\
\varphi_3^{(e)}
\end{bmatrix}
= 
\begin{bmatrix}
b_1^{(e)} \\
b_2^{(e)} \\
b_3^{(e)}
\end{bmatrix}
+ 
\begin{bmatrix}
g_1^{(e)} \\
0 \\
g_3^{(e)}
\end{bmatrix}
\]

For the system shown in Fig. E15.6, we want to assemble the elemental equations for each element into an equation for the whole system if the solution is required to satisfy the boundary conditions

\[
\varphi(x)|_{x=0} = p ; \quad \left[ \alpha(x) \frac{d\varphi(x)}{dx} + \gamma \varphi(x) \right]_{x=L} = q
\]

We write the elemental equation for the first element as

\[
\begin{bmatrix}
K_{11}^{(1)} & K_{12}^{(1)} & K_{13}^{(1)} \\
K_{21}^{(1)} & K_{22}^{(1)} & K_{23}^{(1)} \\
K_{31}^{(1)} & K_{32}^{(1)} & K_{33}^{(1)}
\end{bmatrix}
\begin{bmatrix}
\varphi_1^{(1)} \\
\varphi_2^{(1)} \\
\varphi_3^{(1)}
\end{bmatrix}
= 
\begin{bmatrix}
b_1^{(1)} \\
b_2^{(1)} \\
b_3^{(1)}
\end{bmatrix}
+ 
\begin{bmatrix}
g_1^{(1)} \\
0 \\
g_3^{(1)}
\end{bmatrix}
\]

or equivalently,

\[
K_{11}^{(1)} \varphi_1^{(1)} + K_{12}^{(1)} \varphi_2^{(1)} + K_{13}^{(1)} \varphi_3^{(1)} = b_1^{(1)} + g_1^{(1)}
\]

\[
K_{21}^{(1)} \varphi_1^{(1)} + K_{22}^{(1)} \varphi_2^{(1)} + K_{23}^{(1)} \varphi_3^{(1)} = b_2^{(1)}
\]

\[
K_{31}^{(1)} \varphi_1^{(1)} + K_{32}^{(1)} \varphi_2^{(1)} + K_{33}^{(1)} \varphi_3^{(1)} = b_3^{(1)} + g_3^{(1)}
\]
Similarly, we can write out the equations for the remaining two elements, finding that
\[
\begin{align*}
K_{11}^{(2)} \phi_1^{(2)} + K_{12}^{(2)} \phi_2^{(2)} + K_{13}^{(2)} \phi_3^{(2)} &= b_1^{(2)} + g_1^{(2)} \\
K_{21}^{(2)} \phi_1^{(2)} + K_{22}^{(2)} \phi_2^{(2)} + K_{23}^{(2)} \phi_3^{(2)} &= b_2^{(2)} \\
K_{31}^{(2)} \phi_1^{(2)} + K_{32}^{(2)} \phi_2^{(2)} + K_{33}^{(2)} \phi_3^{(2)} &= b_3^{(2)} + g_3^{(2)}
\end{align*}
\]
and
\[
\begin{align*}
K_{11}^{(3)} \phi_1^{(3)} + K_{12}^{(3)} \phi_2^{(3)} + K_{13}^{(3)} \phi_3^{(3)} &= b_1^{(3)} + g_1^{(3)} \\
K_{21}^{(3)} \phi_1^{(3)} + K_{22}^{(3)} \phi_2^{(3)} + K_{23}^{(3)} \phi_3^{(3)} &= b_2^{(3)} \\
K_{31}^{(3)} \phi_1^{(3)} + K_{32}^{(3)} \phi_2^{(3)} + K_{33}^{(3)} \phi_3^{(3)} &= b_3^{(3)} + g_3^{(3)}
\end{align*}
\]
For this region, the relationship between the local and global numbering systems is defined as
\[
\begin{align*}
x_1^{(e)} &= x_{2e-1} \\
x_2^{(e)} &= x_{2e} \\
x_3^{(e)} &= x_{2e+1}
\end{align*}
\]
for \(e = 1, 2, 3\).

The condition of continuity at nodes 2 and 3 then requires that \(\tilde{\phi}_3^{(1)} = \phi_1^{(2)}\) and \(\tilde{\phi}_3^{(2)} = \phi_1^{(3)}\). Further, as evidenced in Fig. E15.6,
\[
\begin{align*}
\tilde{\phi}_3^{(1)} &= \tilde{\phi}_1 \\
\tilde{\phi}_3^{(2)} &= \tilde{\phi}_5 \\
\tilde{\phi}_3^{(3)} &= \tilde{\phi}_6 \\
\phi_3^{(2)} &= \phi_3^{(3)} = \phi_7
\end{align*}
\]
Thus, we rewrite the individual elemental equations in the form
\[
\begin{align*}
K_{11}^{(1)} \phi_1^{(1)} + K_{12}^{(1)} \phi_2^{(1)} + K_{13}^{(1)} \phi_3^{(1)} &= b_1^{(1)} + g_1^{(1)} \\
K_{21}^{(1)} \phi_1^{(1)} + K_{22}^{(1)} \phi_2^{(1)} + K_{23}^{(1)} \phi_3^{(1)} &= b_2^{(1)} \\
K_{31}^{(1)} \phi_1^{(1)} + K_{32}^{(1)} \phi_2^{(1)} + K_{33}^{(1)} \phi_3^{(1)} &= b_3^{(1)} + g_3^{(1)} \\
K_{11}^{(2)} \phi_3^{(2)} + K_{12}^{(2)} \phi_4^{(2)} + K_{13}^{(2)} \phi_5^{(2)} &= b_1^{(2)} + g_1^{(2)} \\
K_{21}^{(2)} \phi_3^{(2)} + K_{22}^{(2)} \phi_4^{(2)} + K_{23}^{(2)} \phi_5^{(2)} &= b_2^{(2)} \\
K_{31}^{(2)} \phi_3^{(2)} + K_{32}^{(2)} \phi_4^{(2)} + K_{33}^{(2)} \phi_5^{(2)} &= b_3^{(2)} + g_3^{(2)} \\
K_{11}^{(3)} \phi_5^{(3)} + K_{12}^{(3)} \phi_6^{(3)} + K_{13}^{(3)} \phi_7^{(3)} &= b_1^{(3)} + g_1^{(3)} \\
K_{21}^{(3)} \phi_5^{(3)} + K_{22}^{(3)} \phi_6^{(3)} + K_{23}^{(3)} \phi_7^{(3)} &= b_2^{(3)} \\
K_{31}^{(3)} \phi_5^{(3)} + K_{32}^{(3)} \phi_6^{(3)} + K_{33}^{(3)} \phi_7^{(3)} &= b_3^{(3)} + g_3^{(3)}
\end{align*}
\]

Unfortunately, we now have nine equations but only seven unknowns. We know that some of these equations must be redundant, and we therefore need to reduce the number of equations. In order to do this, we can replace the third and fourth equations by their sum, as well as the sixth and seventh equations by their sum. We then find that
\[
\begin{bmatrix}
K_{11}^{(1)} & K_{12}^{(1)} & K_{13}^{(1)} & 0 & 0 & 0 & 0 \\
K_{21}^{(1)} & K_{22}^{(1)} & K_{23}^{(1)} & 0 & 0 & 0 & 0 \\
K_{31}^{(1)} & K_{32}^{(1)} & K_{33}^{(1)} & K_{11}^{(2)} & K_{12}^{(2)} & K_{13}^{(2)} & 0 \\
0 & 0 & K_{21}^{(2)} & K_{22}^{(2)} & K_{23}^{(2)} & 0 & 0 \\
0 & 0 & K_{31}^{(2)} & K_{32}^{(2)} & K_{33}^{(2)} & K_{11}^{(3)} & K_{12}^{(3)} & K_{13}^{(3)} \\
0 & 0 & 0 & 0 & K_{21}^{(3)} & K_{22}^{(3)} & K_{23}^{(3)} & 0 \\
0 & 0 & 0 & 0 & K_{31}^{(3)} & K_{32}^{(3)} & K_{33}^{(3)} & 0
\end{bmatrix}
\begin{bmatrix}
\phi_1 \\
\phi_2 \\
\phi_3 \\
\phi_4 \\
\phi_5 \\
\phi_6 \\
\phi_7
\end{bmatrix}
= \begin{bmatrix}
b_1^{(1)} \\
b_2^{(1)} \\
b_3^{(1)} + b_1^{(2)} \\
b_2^{(2)} \\
b_3^{(2)} + b_1^{(3)} \\
b_3^{(3)} + g_3^{(3)}
\end{bmatrix}
\begin{bmatrix}
g_1^{(1)} \\
g_2^{(1)} \\
g_3^{(1)} + g_2^{(2)} \\
g_2^{(2)} \\
g_3^{(2)} + g_3^{(3)}
\end{bmatrix}
In Exercise 15.11, we found that

\[ g_1^{(e)} = -\alpha(x_1^{(e)}) \left. \frac{d\tilde{\varphi}^{(e)}}{dx} \right|_{x_1^{(e)}} \]

\[ g_2^{(e)} = 0 \]

\[ g_3^{(e)} = \alpha(x_3^{(e)}) \left. \frac{d\tilde{\varphi}^{(e)}}{dx} \right|_{x_3^{(e)}} \]

With these values, we conclude also that

\[ g_3^{(1)} + g_1^{(2)} = \alpha(x_3^{(1)}) \left. \frac{d\tilde{\varphi}^{(1)}}{dx} \right|_{x_3^{(1)}} - \alpha(x_1^{(2)}) \left. \frac{d\tilde{\varphi}^{(2)}}{dx} \right|_{x_1^{(2)}} = 0 \]

and

\[ g_3^{(2)} + g_1^{(3)} = \alpha(x_3^{(2)}) \left. \frac{d\tilde{\varphi}^{(2)}}{dx} \right|_{x_3^{(2)}} - \alpha(x_1^{(3)}) \left. \frac{d\tilde{\varphi}^{(3)}}{dx} \right|_{x_1^{(3)}} = 0 \]

because \( x_3^{(1)} = x_1^{(2)} = x_3^{(2)} = x_3^{(3)} = x_5 \), and the derivative of \( \tilde{\varphi} \) must be continuous across the boundary between elements. Hence, all elements except the first and last in \( \{g\} \) are zero, and our system of equations now becomes

\[
\begin{bmatrix}
K_{11}^{(1)} & K_{12}^{(1)} & K_{13}^{(1)} & 0 & 0 & 0 & 0 \\
K_{21}^{(1)} & K_{22}^{(1)} & K_{23}^{(1)} & 0 & 0 & 0 & 0 \\
K_{31}^{(1)} & K_{32}^{(1)} & K_{33}^{(1)} + K_{11}^{(1)} & K_{12}^{(1)} & K_{13}^{(1)} & 0 & 0 \\
0 & 0 & K_{21}^{(2)} & K_{22}^{(2)} & K_{23}^{(2)} & 0 & 0 \\
0 & 0 & K_{31}^{(2)} & K_{32}^{(2)} & K_{33}^{(2)} + K_{11}^{(2)} & K_{12}^{(2)} & K_{13}^{(2)} \\
0 & 0 & 0 & 0 & K_{21}^{(3)} & K_{22}^{(3)} & K_{23}^{(3)} \\
0 & 0 & 0 & 0 & K_{31}^{(3)} & K_{32}^{(3)} & K_{33}^{(3)}
\end{bmatrix}
\begin{bmatrix}
\tilde{\varphi}_1 \\
\tilde{\varphi}_2 \\
\tilde{\varphi}_3 \\
\tilde{\varphi}_4 \\
\tilde{\varphi}_5 \\
\tilde{\varphi}_6 \\
\tilde{\varphi}_7
\end{bmatrix}
= 
\begin{bmatrix}
b_1^{(1)} \\
b_2^{(1)} \\
b_3^{(1)} + b_1^{(2)} \\
b_4^{(2)} \\
b_5^{(2)} + b_1^{(3)} \\
b_6^{(3)} \\
b_7^{(3)}
\end{bmatrix} + 
\begin{bmatrix}
g_1^{(1)} \\
g_2^{(2)} \\
g_3^{(3)}
\end{bmatrix}
\]

More explicit values for all entries in this equation except the unknowns in the vector \( \{\tilde{\varphi}\} \) can be inferred from the results in Exercise 15.11, though those values apply only if the interior nodes in each element are halfway between the end nodes of that element.
15.20 FDM 2D Inhomogeneous Helmholtz Equation

**Exercise:** The inhomogeneous Helmholtz equation in two-dimensional Cartesian coordinates is

\[
\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + k^2 u = r(x, y)
\]

where \( k^2 \) is a constant and \( r(x, y) \) is the inhomogeneity. Apply finite difference methods to show that

\[
u_{i,j} \approx u_{i+1,j} + u_{i-1,j} + u_{i,j+1} + u_{i,j-1} - \frac{\Delta x^2 r_{i,j}}{4 - k^2 \Delta x^2}
\]

Here, \( u_{i,j} = u(x_i, y_j) \), the spacing between consecutive values of \( x \) is \( \Delta x \), the spacing between consecutive values of \( y \) is \( \Delta y \) and \( \Delta y = \Delta x \). This result could be used in an iterative approach to solving the inhomogeneous Helmholtz equation. **Note:** If \( k^2 = 0 \), the equation of this exercise reduces to the inhomogeneous Laplace equation, i.e., to the Poisson equation. If, on the other hand, \( r(x, y) = 0 \), then this equation reduces to the (homogeneous) Helmholtz equation.

**Solution:** Paralleling our argument on Eq. (15.75) in the text, we infer that

\[
d\varphi^2(x, y) \bigg|_{x_i, y_j} \approx \frac{d\varphi(x, y)}{dx} \bigg|_{x_i + \Delta x/2, y_j} - \frac{d\varphi(x, y)}{dx} \bigg|_{x_i - \Delta x/2, y_j}\]

\[
\approx \frac{\varphi_{i+1,j} - \varphi_{i,j}}{\Delta x} - \frac{\varphi_{i,j} - \varphi_{i-1,j}}{\Delta x}
\]

\[
\approx \frac{\varphi_{i+1,j} - 2\varphi_{i,j} + \varphi_{i-1,j}}{\Delta x^2}
\]

Similarly, we conclude that

\[
d\varphi^2(x, y) \bigg|_{x_i, y_j} \approx \frac{\varphi_{i,j+1} - 2\varphi_{i,j} + \varphi_{i,j-1}}{\Delta x^2}
\]

Finally, evaluating the original differential equation at \((x_i, y_j)\), we conclude that

\[
\frac{\varphi_{i+1,j} - 2\varphi_{i,j} + \varphi_{i-1,j}}{\Delta x^2} + \frac{\varphi_{i,j+1} - 2\varphi_{i,j} + \varphi_{i,j-1}}{\Delta x^2} + k^2 u_{i,j} = r_{i,j}
\]

which, on solution for \( u_{i,j} \) yields that

\[
u_{i,j} \approx \frac{u_{i+1,j} + u_{i-1,j} + u_{i,j+1} + u_{i,j-1} - \Delta x^2 r_{i,j}}{4 - k^2 \Delta x^2}
\]

Q.E.D.
15.25 Finite Difference Formulas via Taylor Series

Exercise: (a) Starting with the Taylor series

\[ \tilde{\varphi}(x + \Delta x) = \tilde{\varphi}(x) + \Delta x \tilde{\varphi}'(x) + O(\Delta x^2) \]

evaluate

\[ \tilde{\varphi}(x + \Delta x) - \tilde{\varphi}(x) \]
to show that

\[ \tilde{\varphi}'(x) = \frac{\tilde{\varphi}(x + \Delta x) - \tilde{\varphi}(x)}{\Delta x} + O(\Delta x) \]

That is, derive the forward difference formula in Eq. (15.74). (b) Deduce the more accurate forward difference formula

\[ \tilde{\varphi}'(x) \approx -\frac{\tilde{\varphi}(x + 2\Delta x)}{2\Delta x} + \frac{4\tilde{\varphi}(x + \Delta x) - 3\tilde{\varphi}(x)}{2\Delta x} + O(\Delta x^2) \]

Hint: Start by using the Taylor expansion

\[ f(x + h) = f(x) + hf'(x) + \frac{1}{2} h^2 f''(x) + O(h^3) \]
to expand \( \tilde{\varphi}(x + \Delta x) \) and \( \tilde{\varphi}(x + 2\Delta x) \).

Solution: (a) This first part involves little more than a rearrangement of the terms in the Taylor expansion provided in part (a), first to

\[ \tilde{\varphi}(x + \Delta x) - \tilde{\varphi}(x) = \Delta x \tilde{\varphi}'(x) + O(\Delta x^2) \]

and then to

\[ \frac{\tilde{\varphi}(x + \Delta x) - \tilde{\varphi}(x)}{\Delta x} = \tilde{\varphi}'(x) + O(\Delta x^2) \]

and finally to\(^8\)

\[ \tilde{\varphi}'(x) = \frac{\tilde{\varphi}(x + \Delta x) - \tilde{\varphi}(x)}{\Delta x} + O(\Delta x) \]

Introducing \( x = x_i \) and \( x + \Delta x = x_{i+1} \) in the above Taylor expansion to find that

\[ \left. \frac{d\tilde{\varphi}(x)}{dx} \right|_{x_i} = \frac{\tilde{\varphi}_{x_{i+1}} - \tilde{\varphi}_i}{\Delta x} + O(\Delta x) \]

(b) Based on the Taylor series provided in part (b), we note that

\[ \tilde{\varphi}(x + \Delta x) = \tilde{\varphi}(x) + \Delta x \tilde{\varphi}'(x) + \frac{1}{2} \Delta x^2 \tilde{\varphi}''(x) + O(\Delta x^3) \]

and that

\[ \tilde{\varphi}(x + 2\Delta x) = \tilde{\varphi}(x) + 2\Delta x \tilde{\varphi}'(x) + \frac{1}{2} (2\Delta x)^2 \tilde{\varphi}''(x) + O(\Delta x^3) \]

Solving the first of these equations for \( \tilde{\varphi}''(x) \Delta x^2 \) yields

\[ \tilde{\varphi}''(x) \Delta x^2 = 2\tilde{\varphi}(x + \Delta x) - 2\tilde{\varphi}(x) - 2\Delta x \tilde{\varphi}'(x) + O(\Delta x^3) \]

Substituting this result into the second equation then yields that

\[ \tilde{\varphi}(x + 2\Delta x) = \tilde{\varphi}(x) + 2\Delta x \tilde{\varphi}'(x) + 2 \left( \frac{1}{2} \tilde{\varphi}(x + \Delta x) - \frac{1}{2} \tilde{\varphi}(x) - \frac{1}{2} \Delta x \tilde{\varphi}'(x) \right) + O(\Delta x^3) \]

\(^8\)We can retain the plus sign before \( O(\Delta x) \) because it is of no consequence in specifying the magnitude of the potential error.
Combining those terms that can be combined, we learn that

\[ \tilde{\varphi}(x + 2 \Delta x) = -3 \tilde{\varphi}(x) - 2 \Delta x \tilde{\varphi}'(x) + 4 \tilde{\varphi}(x + \Delta x) + O(\Delta x^3) \]

Finally, solving for \( \tilde{\varphi}'(x) \), we find that

\[ \tilde{\varphi}'(x) = \frac{-3 \tilde{\varphi}(x) + 4 \tilde{\varphi}(x + \Delta x) - \tilde{\varphi}(x + 2 \Delta x)}{2 \Delta x} + O(\Delta x^2) \]

With \( x = x_i \), \( x + \Delta x = x_{i+1} \), and \( x + 2 \Delta x = x_{i+2} \), this result has the expression

\[ \left. \frac{d\tilde{\varphi}(x)}{dx} \right|_{x_i} = \frac{-3 \tilde{\varphi}_i + 4 \tilde{\varphi}_{i+1} - \tilde{\varphi}_{i+2}}{2 \Delta x} + O(\Delta x^2) \]
15.26 Shape Functions for Two-Node Linear Element

**Exercise:** Starting with the first two numbered equations in Section 15.9.2, work through the details in that section to derive the results

\[
N_1^{(e)}(x) = \frac{x_2^{(e)} - x}{l^{(e)}} ; \quad N_2^{(e)}(x) = \frac{x - x_1^{(e)}}{l^{(e)}}
\]

for the two interpolation functions. Here, \(l^{(e)} = x_2^{(e)} - x_1^{(e)}\).

**Solution:** We begin by presuming that the solution in the interval \(x_1^{(e)} \leq x \leq x_2^{(e)}\) can be approximated with the linear function

\[
\tilde{\varphi}^{(e)} = a^{(e)} + b^{(e)} x
\]

We require that this function yield the known values at the ends of the interval, i.e., that

\[
\tilde{\varphi}_1^{(e)} = a^{(e)} + b^{(e)} x_1^{(e)}
\]
\[
\tilde{\varphi}_2^{(e)} = a^{(e)} + b^{(e)} x_2^{(e)}
\]

Subtracting the first of these equations from the second yields that

\[
\tilde{\varphi}_2^{(e)} - \tilde{\varphi}_1^{(e)} = b^{(e)} (x_2^{(e)} - x_1^{(e)}) \implies b^{(e)} = \frac{\tilde{\varphi}_2^{(e)} - \tilde{\varphi}_1^{(e)}}{x_2^{(e)} - x_1^{(e)}}
\]

Then, the first equation yields that

\[
\tilde{\varphi}_1^{(e)} = a^{(e)} + \left(\frac{\tilde{\varphi}_2^{(e)} - \tilde{\varphi}_1^{(e)}}{x_2^{(e)} - x_1^{(e)}}\right) x_1^{(e)} \implies a^{(e)} = \tilde{\varphi}_1^{(e)} - \left(\frac{\tilde{\varphi}_2^{(e)} - \tilde{\varphi}_1^{(e)}}{x_2^{(e)} - x_1^{(e)}}\right) x_1^{(e)}
\]

\[
\implies a^{(e)} = \tilde{\varphi}_1^{(e)} \frac{x_2^{(e)} - x_1^{(e)}}{x_2^{(e)} - x_1^{(e)}} = \tilde{\varphi}_1^{(e)} \frac{x_2^{(e)} - x_1^{(e)}}{l^{(e)}}
\]

Substituting these results for the two constants into the first equation in this solution, we learn that

\[
\tilde{\varphi}^{(e)} = \tilde{\varphi}_1^{(e)} \frac{x_2^{(e)} - x_1^{(e)}}{l^{(e)}} + \left(\frac{\tilde{\varphi}_2^{(e)} - \tilde{\varphi}_1^{(e)}}{l^{(e)}}\right) x = \tilde{\varphi}_1^{(e)} \left(\frac{x_2^{(e)} - x}{l^{(e)}}\right) + \tilde{\varphi}_2^{(e)} \left(\frac{x - x_1^{(e)}}{l^{(e)}}\right)
\]

from which we learn, finally, that the interpolation functions are

\[
N_1^{(e)}(x) = \frac{x_2^{(e)} - x}{l^{(e)}} ; \quad N_2^{(e)}(x) = \frac{x - x_1^{(e)}}{l^{(e)}}
\]
15.1 FDM 1D Inhomogeneous Equation (MATLAB)

Exercise: Recast \texttt{fdm1d} to solve the ODE of Eq. (15.67) when \( f(x) = kx \), the boundary conditions are
\[
\frac{\partial \varphi}{\partial x}(0) = p \quad ; \quad \varphi(L) = q
\]
and \( \alpha, \beta, k, p, q \), and \( L \) are constants whose values are to be read in at execution time. Then, using your command file, explore the solution to the equation for various values of \( k \) when
\[
\alpha = -4.0 \quad ; \quad \beta = 4.0 \quad ; \quad L = 10.0 \quad ; \quad p = q = 0
\]

Optional: (a) Find an analytic solution to the equation in this exercise and compare the exact results with the approximate results generated by your modification of \texttt{fdm1d}. (b) Find the points at which the solution has the value zero, both starting with the solution obtained in the main exercise and working from the exact solution obtained in optional part (a).

Solution: We start with the ODE of Eq. (15.67)
\[
-\alpha(x) \frac{d^2 \varphi(x)}{dx^2} - \alpha'(x) \frac{d\varphi(x)}{dx} + \beta(x) \varphi(x) = f(x) \tag{E15.1.1}
\]
Because \( \alpha, \beta, k, p, q \), and \( L \) are all constants, we can drop their dependence on \( x \), and also note that \( \alpha'(x) \) is zero. We are left with the equation
\[
-\alpha \frac{d^2 \varphi(x)}{dx^2} + \beta \varphi(x) = f(x) \tag{E15.1.2}
\]
The boundary conditions given in the problem are
\[
\frac{\partial \varphi}{\partial x}(0) = p \quad ; \quad \varphi(L) = q \tag{E15.1.3}
\]
We want to find the solution to Eq. (E15.1.2) in the interval \( 0 \leq x \leq L \). First, we divide the interval into \( N \) segments, each of length \( \Delta x = L/N \), introducing \( x_i = i \Delta x, i = 0, 1, 2, \ldots, N \) and noting that \( x_0 = 0 \), and \( x_N = L \). Then, we evaluate Eq. (E15.1.2) at the point \( x_i \) to find that
\[
-\alpha \frac{d^2 \varphi(x)}{dx^2} \bigg|_{x_i} + \beta \varphi(x_i) = f(x_i) \tag{E15.1.4}
\]
Next, we approximate the derivatives by invoking finite differences. We approximate the first derivative with a forward difference, a backward difference and a central difference, the results being
\[
\frac{d\varphi(x)}{dx} \bigg|_{x_i} \approx \frac{\varphi_{i+1} - \varphi_i}{\Delta x} \quad ; \quad \frac{d\varphi(x)}{dx} \bigg|_{x_i} \approx \frac{\varphi_i - \varphi_{i-1}}{\Delta x} \quad ; \quad \frac{d\varphi(x)}{dx} \bigg|_{x_i} \approx \frac{\varphi_{i+1} - \varphi_{i-1}}{2\Delta x} \tag{E15.1.5}
\]
We are looking for the second derivative of \( \varphi(x) \), so we use both the forward and backward approximations of the first derivative. The expression
\[
\frac{d^2 \varphi(x)}{dx^2} \bigg|_{x_i} \approx \frac{\varphi_{i+1} - 2\varphi_i + \varphi_{i-1}}{\Delta x^2} \tag{E15.1.6}
\]
is the same as that obtained in the text as Eq. (15.75). Now we substitute this approximation into Eq. (E15.1.4) to find that
\[
-\alpha \left[ \frac{\varphi_{i+1} - 2\varphi_i + \varphi_{i-1}}{\Delta x^2} \right] + \beta \varphi_i = f_i \tag{E15.1.7}
\]
We multiply by \( \Delta x^2 \) and collect the terms with each index to find finally that
\[
-\alpha \varphi_{i-1} + (2\alpha + \beta \Delta x^2)\varphi_i - \alpha \varphi_{i+1} = f_i \Delta x^2 \tag{E15.1.8}
\]
Unfortunately this equation cannot be solved for \( i = 0 \) or \( i = N \) because \( \varphi_{-1} \) and \( \varphi_{N+1} \) are not within the domain of the problem. To resolve those special cases, we look at the boundary conditions, which are stated in Eq. (E15.1.3). The first condition is a Neumann condition or a condition of the second kind, while the second condition is a Dirichlet condition, or a condition of the first kind. Let’s first look at the Dirichlet condition. We abandon the result given by Eq. (E15.1.8) for \( i = N \), and we replace it with the equation given by the boundary condition, namely

\[ \varphi_N = q \]  

(E15.1.9)

For the Neumann condition, we use the central difference approximation to the derivative at the boundary, finding that

\[ \left. \frac{d\varphi}{dx} \right|_{x=0} \approx \frac{\varphi_{i+1} - \varphi_{i-1}}{2 \Delta x} = p \]  

(E15.1.10)

Then set \( i = 0 \), simplify, and solve for \( \varphi_{-1} \), finding that

\[ \varphi_{-1} = \varphi_1 - 2p \Delta x \]  

(E15.1.11)

We write out Eq. (E15.1.8) for \( i = 0 \) and substitute from Eq. (E15.1.11) to eliminate \( \varphi_{-1} \), finding that

\[ (2\alpha + \beta \Delta x^2)\varphi_0 - 2\alpha \varphi_1 = f_0 \Delta x^2 - 2p\alpha \Delta x \]  

(E15.1.12)

Now we are able fully to define the set of \( N + 1 \) equations for the \( N + 1 \) unknowns \((\varphi_0, \varphi_1, \varphi_2, \ldots, \varphi_N)\). The

\[ (2\alpha + \beta \Delta x^2)\varphi_0 - 2\alpha \varphi_1 = f_0 \Delta x^2 - 2p\alpha \Delta x \]

\[ -\alpha \varphi_{i-1} + (2\alpha + \beta \Delta x^2)\varphi_i - \alpha \varphi_{i+1} = f_i \Delta x^2, \quad 1 \leq i \leq N - 1 \]  

(E15.1.13)

\[ \varphi_N = q \]

We have deduced a set of linear algebraic equations to be solved for the unknowns. We note that this set can be seen in matrix form, which would be tridiagonal, since each equation involves no more than three consecutive indices.

The final step in solving Eq. (E15.1.13) in MATLAB is to write a MATLAB program that will construct the matrix form of Eq. (E15.1.13) and then solve for \( \varphi_i \). In our case, we want to recast the program \textit{fdm1d.m}. In order to adapt the program to this specific problem, we will need to (1) represent \( f \) by an \( N + 1 \) element vector, with values given for each element (because \( f \) is dependent on \( x \) rather than being constant), (2) change the vector containing the inhomogeniety, (3) recalculate the matrix of coefficients (which is still tri-diagonal), and (4) remove \( \gamma \) from the program, since that variable doesn’t appear in the problem at hand. First we include a prompt that asks the user to input a value for \( k \) with the statement,

\[ k = \text{input( 'Enter k: ' )}; \]

Then we have to determine a vector to represent \( f \) that will contain all of the values of \( f_i \), by inserting the statement\(^9\)

\[ f = k*x'; \]

after \( x \) has been calculated. Finally, we have to correct the vector of inhomogenieties, and the elements of the coefficient matrix. We replace the statements calculating these quantities in \textit{fdm1d.m} with the statements

---

\(^9\)The vector \( x \) must be transposed so it is a column vector rather than a row vector for compatibility with the rest of the calculation.
inhomo = f*dx2;
inhomo(1) = inhomos(1) - 2.0*alpha*dx*p;
inhomo(N+1) = q;

cf = zeros(N+1,N+1);
for i=1:N+1 cf(i,i) = 2.0*alpha + beta*dx2; end
cf(N+1,N+1) = 1.0;

for i=1:N cf(i+1,i) = -alpha; end
cf(N+1,N) = 0.0;

for i=1:N cf(i,i+1) = -alpha; end
cf(1,2) = -2.0*alpha;

Remember (1) that, in MATLAB, the first index of an array is the row index and the second index
is the column index and (2) that the indices of a vector with \( N + 1 \) elements run in MATLAB from
1 to \( N + 1 \), e.g., what is \( \phi_i \) in the above discussion will be \( \phi(i+1) \) in the program.

The complete program, which we name pde_ex01.m, is given at the end of the solution. We run
the program in MATLAB, choosing the constants arbitrarily to be

\[
N = 100 \quad ; \quad \alpha = -4.0 \quad ; \quad \beta = 4.0 \quad ; \quad L = 10.0 \quad ; \quad p = q = 0.0
\]

We’ve chosen to evaluate the equation at \( k = 1.0, 2.0, 4.0, 8.0 \). This is accomplished, along with
producing graphs of each solution, with the statements,

```
pde_ex01
    Enter number of segments (N): 100
    Enter alpha: -4.0
    Enter beta: 4.0
    Enter k: 1.0
    Enter L: 10.0
    Enter p: 0.0
    Enter q: 0.0
    x1 = x ; phi1 = phi;
pde_ex01
    Enter k: 2.0
    (rest same as for k = 1.0)
x2 = x ; phi2 = phi;
pde_ex01
    Enter k: 4.0
    (rest same as for k = 1.0)
x4 = x ; phi4 = phi;
pde_ex01
    Enter k: 8.0
    (rest same as for k = 1.0)
x8 = x ; phi8 = phi;
```

```
subplot(221)
plot(x1,phi1, 'linewidth',3, 'color','black')
grid on
set(gca, 'fontsize',12)
axis([0.0,10.0,-20.0,40.0])
title('k = 1.0', 'fontsize',16)
```
The graphs are shown in Fig. E15.7.

To be sure that we have chosen a sufficient number of points, and to test the accuracy of our solution, we look at graphs with \( N = 50, 100, 200 \) with the following statements

```
pde_ex01
Enter number of segments (N): 100
Enter alpha:  -4.0
Enter beta:  4.0
Enter k:  4.0
Enter L:  10.0
Enter p:  0.0
Enter q:  0.0
x100 = x ; phi100 = phi;
pde_ex01
Enter number of segments (N): 50
(repeat same as for N = 100)
x050 = x ; phi050 = phi;
pde_ex01
Enter number of segments (N): 200
(repeat same as for N = 100)
x200 = x ; phi200 = phi;
```
Exercise 15.1 (MATLAB)

Figure E15.7: Graphs of $\varphi(x)$ versus $x$ for $k = 1.0, 2.0, 4.0, 8.0$. This figure was produced with MATLAB.

plot(x100,phi100, 'linewidth',6, 'color','black')
plot(x200,phi200, 'linewidth',10, 'color','black')

This graph is shown in Fig. E15.8. Since these three graphs all lie on top of one another,$^{10}$ we can safely conclude that the solution is the same for all values of $N$, i.e., that the solution we have found is accurate at least to the resolution of the graph.

We can also test the accuracy of the solution by comparing the values at corresponding points for two of the solutions just generated. More specifically, if we execute the statements

```matlab
% Compute the difference between the two solutions
diff = zeros([101,1]);
for i = 1:101
    diff(i) = phi200(2*i-1) - phi100(i);
end
max(diff), min(diff)
0.0437 -0.0360
max(phi200), min(phi200)
18.8566 -9.5916
```

We see here that the solution for $N = 100$ is accurate to an absolute error of about $\pm 0.04$ and that the solution falls in the range $-9.6 \leq \varphi \leq 18.9$.

$^{10}$That the graphs lie on top of one another is easily seen if you pause after each plot statement and watch how the thicker lines overlay as the graph unfolds on the screen.
Figure E15.8: Graph of solution for \( N = 50, N = 100, \) and \( N = 200 \) for \( k = 4.0. \) This figure was produced with MATLAB.

Optional Parts

(a) With the explicit values

\[
\alpha = -4.0 \quad ; \quad \beta = 4.0 \quad ; \quad L = 10.0 \quad ; \quad p = q = 0.0
\]

chosen above, we seek to solve the equation

\[
4 \frac{d^2 \varphi}{dx^2} + 4 \varphi = k x \quad \text{or} \quad \frac{d^2 \varphi}{dx^2} + \varphi = \frac{k}{4} x
\]

subject to the boundary conditions

\[
\frac{\partial \varphi}{\partial x}(0) = 0 \quad ; \quad \varphi(10) = 0
\]

The standard strategy for analytic solution of equations of this sort is to add a particular solution that generates the inhomogeneity to a general solution to the homogeneous equation. That approach results in the tentative solution

\[
\varphi(x) = A \sin(x) + B \cos(x) + \frac{k}{4} x
\]

The boundary conditions then require that

\[
\frac{\partial \varphi}{\partial x}(0) = 0 \quad \implies \quad A \cos(0) - B \sin(0) + \frac{k}{4} = 0 \quad \implies \quad A = -\frac{k}{4}
\]

and

\[
\varphi(10) = 0 \quad \implies \quad -\frac{k}{4} \sin(10) + B \cos(10) + \frac{10k}{4} = 0 \quad \implies \quad B = k \frac{\sin(10)/4 - 10/4}{\cos(10)} = 3.1416 k
\]
Here, we understand the arguments of the trigonometric functions to be in radians. The general solutions to the original equation thus is
\[ \varphi(x) = -0.25k \sin(x) + 3.1416k \cos(x) + 0.25kx \]

The statements

```matlab
k = [1.0,2.0,4.0,8.0];
x = [0.0:0.1:10.0];
for i=1:4
    phi = -0.25*k(i)*sin(x) + 3.1416*k(i)*cos(x) + 0.25*k(i)*x;
    subplot(2,2,i)
    plot(x,phi, 'linewidth',3, 'color','black')
    grid on
    set(gca, 'fontsize',12)
    axis([0.0,10.0,-20.0,40.0])
    title(['k = ',num2str(k(i))], 'fontsize',16)
    xlabel('x', 'fontsize',16); ylabel('
\varphi', 'fontsize',16);
end
```

quickly produce a graph that looks exactly like that in Fig. E15.7, so it will not be reproduced here. Instead, we compare the solution here obtained for \( k = 8.0 \) with that previously obtained and denoted \( x8 \) and \( \varphi8 \) with the statements\footnote{Note that the immediately executed loop exits with the value \( k = 8 \). The transposition of \( \varphi \) is needed because it is a column vector while \( \varphi8 \) is a row vector.}

```matlab
diff= phi8-phi';
[max(diff), min(diff)]
an = 0.0957  -0.1163
```

These results reveal that the numerical solution obtained for \( k = 8 \) differs from the exact solution by about \( \pm 0.1 \), well within the resolution of a graph whose scaling ranges over the interval \( -20 \leq \varphi \leq 40 \).

(b) Based on Fig. E15.7, we infer that the zeros for \( k = 8 \) occur approximately zero at about \( x = 1.7, 4.2, \) and \( 8.5 \), i.e, near \( x(18), x(43), \) and \( x(86) \). The statement

```matlab
for i = 16:19
    [i, x8(i), phi8(i)]
end
```

display values around \( x(18) \), from which we conclude that the first zero occurs between \( x = 1.6 \) and \( x = 1.7 \). Linear interpolation provides the estimate
\[ \varphi(x) = 0 \text{ at } x = 1.6 + 0.1 \times \frac{0.4454}{0.4454 + 1.8517} = 1.6194 \]

Similarly, we find that the second zero lies between \( x = 4.2 \) (where \( \varphi = -2.1774 \)) and \( x = 4.3 \) (where \( \varphi = 0.3730 \)), so
\[ \varphi(x) = 0 \text{ at } x = 4.2 + 0.1 \times \frac{2.1774}{2.1774 + 0.3730} = 4.2854 \]
Finally, we find that the third zero lies between \( x = 8.5 \) (where \( \varphi = 0.1567 \)) and \( x = 8.6 \) (where \( \varphi = -1.4432 \)), so

\[
\varphi(x) = 0 \at \begin{array}{c}
\text{at } x = 8.5 + 0.1 \\
0.1567 \\
0.1567 + 1.4432
\end{array} = 8.5098
\]

Since we have an analytic solution in hand, we can alternatively invoke OCTAVE's function \texttt{fzero} to find these roots. As its first argument, \texttt{fzero} needs the MATLAB function

\begin{verbatim}
function fct = phi( x )
global kk
fct = -0.25*kk*sin(x) + 3.1416*kk*cos(x) + 0.25*kk*x;
endfunction
\end{verbatim}

which we store in the default directory with the name \texttt{phiex.m}. We then find the roots with the statements

\begin{verbatim}
global kk
kk=8.0;
fzero( @phiex, [1.6,1.7] )
ans = 1.6203
fzero( @phiex, [4.2,4.3] )
ans = 4.2861
fzero( @phiex, [8.5,8.56] )
ans = 8.5163
\end{verbatim}

These results are exact to four digits after the decimal point; the approximate roots obtained by interpolation from the values in \texttt{phi8} agree with these "exact" roots to two places after the decimal point.

A similar approach can be used to find the roots of \( \varphi \) for any other value of \( k \).

\textbf{Listing of pde\_ex01.m}

\begin{verbatim}
% ***** Command file fdm1d.m *****

% ***** Note that, when this command file has completed executing,
% all variables to which it assigns values---and in
% particular x and phi---will be accessible at MATLAB's main
% command level.

% ***** Request input of necessary parameters.

N   = input( 'Enter number of segments (N): ' );
alpha = input( 'Enter alpha: ' );
beta = input( 'Enter beta: ' );
k   = input( 'Enter k: ' );
L   = input( 'Enter L: ' );
p   = input( 'Enter p: ' );
q   = input( 'Enter q: ' );

% ***** Calculate segment size, square of segment size, and
% coordinates of nodes and vector of f .

dx = L/N; dx2 = dx^2;
\end{verbatim}
\( x = dx*[0:N]; \)
\( f = k*x'; \) % Transpose x to get column vector

% ***** Determine vector of inhomogenieties and coefficient matrix.

\( \text{inhomo} = f*dx2; \)
\( \text{inhomo}(1) = \text{inhomo}(1) - 2.0*alpha*dx*p; \)
\( \text{inhomo}(N+1) = q; \)

\( \text{cf} = \text{zeros}(N+1,N+1); \)
\( \text{for } i=1:N+1 \text{ cf}(i,i) = 2.0*alpha + beta*dx2; \) end
\( \text{cf}(N+1,N+1) = 1.0; \)

\( \text{for } i=1:N \text{ cf}(i+1,i) = -alpha; \) end
\( \text{cf}(N+1,N) = 0.0; \)

\( \text{for } i=1:N \text{ cf}(i,i+1) = -alpha; \) end
\( \text{cf}(1,2) = -2.0*alpha; \)

% ***** Solve system using MATLAB's operator \.

\( \text{phi} = \text{cf}\backslash\text{inhomo}; \)
15.4 FDM 1D \(u'' + kxu = 0\) (MATLAB)

**Exercise:** Recast `fdm1d` to solve the equation

\[
\frac{d^2u}{dx^2} + kxu = 0
\]

over the interval \(0 \leq x \leq L\) subject to the boundary conditions \(u(0) = 0\) and \(du(L)/dx = 1.0\), arranging your program so the value of \(k\) is entered at execution time. Then, explore the solution in some detail for various values of \(k\), making sure to assess the accuracy of your solution.

**Solution:** To cast Eq. (15.67) in the form of this problem, we must interpret \(\varphi(x)\) as \(u(x)\). Then, we must set \(\alpha(x) = -1\), \(\beta(x) = kx\), and \(f(x) = 0\). The remaining quantity \(\alpha'(x)\) will then automatically be zero. Further, to match the boundary conditions, we must set \(p = 0\), \(\gamma = 0\), and \(q = -1\). (Remember that \(\alpha\) is negative!). The command file `fdm1d.m` for MATLAB requires only one adjustment to accommodate this change, specifically, we must adapt the program so that \(\beta(x)\) is a function of \(x\) rather than a constant. Thus, we must

- Replace the entry of \(\beta\) with the entry of \(k\) at execution time, i.e., we replace the statement requesting \(\beta\) with the statement
  
  ```matlab
  k = input( 'Enter k: ' );
  ```

- Create a vector `beta` having \(N+1\) elements and store in that vector the values of \(kx\) by adding the statement
  
  ```matlab
  beta = k * x
  ```

  after the vector \(x\) has been created.

- In the loop that evaluates \(c(i,i)\), replace `beta` with `beta(i)` so the proper value of \(beta\) will be used at each step in that loop.

- Remove all references to \(\gamma\) (\(gamma\)), since that quantity plays no role in the computation and repeated multiplications by zero aren’t really necessary. This removal reduces the statement adjusting the element `diag[N]` to a tautology, so we can remove that statement as well.

No further changes are necessary.

For a first trial, we take \(L = 10\), \(N = 50\), and \(k = 1\) (and remember that \(\alpha = -1\) and \(f = p = q = 0\)). Then, we run this command file, save the output for later reference, and produce an on-screen plot with the input statements

```matlab
pde_ex04
Enter number of segments (N): 50
Enter alpha: -1.0
Enter k: 1.0
Enter f: 0.0
Enter L: 10.0
Enter p: 0.0
Enter q: -1.0
x050 = x; phi050 = phi;
plot( x050, phi050, 'color','black', 'linewidth',2 )
xlabel('x', 'fontsize',12); ylabel('u', 'fontsize',12)
grid on
hold on
```
The resulting plot looks a bit jagged, and we conclude that we need a larger $N$ for smoothness of the graph if not for the sake of accuracy. Thus, with a bit of trial and error, we decide to rerun the command file with $N = 100$, $N = 200$, and $N = 500$ with the statements

```matlab
pde_ex04
Enter number of segments (N): 100
(other parameters the same as above)
x100 = x ; phi100 = phi;
plot( x100, phi100, 'linewidth',2, 'color','black' )
pde_ex04
Enter number of segments (N): 200
(other parameters the same as above)
x200 = x ; phi200 = phi;
plot( x200, phi200, 'linewidth',2, 'color','black' )
pde_ex04
Enter number of segments (N): 500
(other parameters the same as above)
x500 = x ; phi500 = phi;
plot( x500, phi500, 'linewidth',2, 'color','black' )
```

The resulting graph is shown in Fig. E15.9. Since the graphs for $N = 200$ and $N = 500$ essentially coincide, we conclude that $N = 500$ gives accuracy at least to the resolution of the graph, and we will use $N = 500$ in all subsequent analyses.

Having dealt with an assessment of accuracy, we turn to the point of the problem—to examine the dependence of the solution on $k$. As a start, we again look to the solution for $k = 1.0$ and, after some experimentation, select also $k = 0.5$ and $k = 0.2$. The statements

```matlab
pde_ex04
Enter number of segments (N): 500
Enter alpha: -1.0
Enter k: 1.0
Enter f: 0.0
Enter L: 10.0
Enter p: 0.0
Enter q: -1.0
x10 = x ; phi10 = phi;
pde_ex04
Enter k: 0.5
(other parameters the same as above)
x05 = x ; phi05 = phi;
pde_ex04
Enter k: 0.2
(other parameters the same as above)
x02 = x ; phi02 = phi;
hold off
plot( x02, phi02, 'linestyle','-.', 'linewidth',3, 'color','black')
xlabel('x', 'fontsize',12); ylabel('u', 'fontsize',12)
grid on; hold on
plot( x05, phi05, 'linestyle','--', 'linewidth',3, 'color','black')
plot( x10, phi10, 'linewidth',3, 'color','black')
generate those three solutions and display them in the graph that is reproduced in Fig. E15.10. (The order of the plot statements was determined so as to make sure the vertical scaling was set to
Figure E15.9: Test of accuracy using \( N = 50, 100, 200, \) and 500. The upper peaks become progressively higher as \( N \) is increased. This graph was produced with MATLAB.

The solution appears to be oscillatory and to decay in amplitude as \( x \) increases. Further, as \( k \) decreases, the amplitude of the solution decreases and the “wavelength” of the solution increases.

Finally, we allow \( k \) to increase above 1, ultimately deciding to include \( k = 1.0, k = 2.0, \) and \( k = 4.0 \) by creating Fig. E15.11 with the statements

```matlab
pde_ex04
Enter number of segments (N): 500
Enter alpha: -1.0
Enter k: 1.0
Enter f: 0.0
Enter L: 10.0
Enter p: 0.0
Enter q: -1.0
x10 = x & phi10 = phi
pde_ex04
Enter k: 2.0
(other parameters the same as above)
x20 = x ; phi20 = phi;
pde_ex04
Enter k: 4.0
(other parameters the same as above)
x40 = x ; phi40 = phi;
```
Figure E15.10: The solution for $k = 1.0$ (solid line), $k = 0.5$ (dashed line), and $k = 0.2$ (dash-dot line). This graph was produced with MATLAB.

```
hold off
plot(x40, phi40, 'linewidth',3, 'color','black')
xlabel('x', 'fontsize',12); ylabel('u', 'fontsize',12)
grid on; hold on
plot(x20, phi20, 'linestyle','--', 'linewidth',3, 'color','black')
plot(x10, phi10, 'linestyle','-.', 'linewidth',3, 'color','black')
```

These results reinforce the conclusions about the oscillatory nature of the solution, about the increasing amplitude as $k$ increases, and about the decreasing “wavelength” as $k$ increases.
Figure E15.11: The solution for $k = 4.0$ (solid line), $k = 2.0$ (dashed line), and $k = 1.0$ (dash-dot line). This graph was produced with MATLAB.
Listing of pde_ex04.m

% ***** Command file pde-ex04.m *****

% ***** Note that, when this command file has completed executing, %
% all variables to which it assigns values---and in %
% particular x and phi---will be accessible at MATLAB’s main %
% command level.

% ***** Request input of necessary parameters.
N = input( 'Enter number of segments (N): ' );
alpha = input( 'Enter alpha: ' );
k = input( 'Enter k: ' );
f = input( 'Enter f: ' );
L = input( 'Enter L: ' );
p = input( 'Enter p: ' );
q = input( 'Enter q: ' );

% ***** Calculate segment size, square of segment size, and %
% coordinates of nodes.

dx = L/N; dx2 = dx^2;
x = dx*[ 0 : N ];
beta = k*x;

% ***** Determine vector of inhomogenieties and coefficient matrix.

inhomo = zeros(N+1,1) + 1.0;
inhomo = f*dx2*inhomo;
inhomo(1) = p;
inhomo(N+1) = inhomo(N+1) + 2.0*q*dx;

cf = zeros(N+1,N+1);
for i=1:N+1 cf(i,i) = 2.0*alpha + beta(i)*dx2; end
cf(1,1) = 1.0;
cf(N+1,N+1) = cf(N+1,N+1);
for i=1:N cf(i,i+1) = -alpha; end
cf(1,2) = 0.0;

for i=1:N cf(i+1,i)= -alpha; end
cf(N+1,N) = -2.0*alpha;

% ***** Solve system using MATLAB’s operator \.

phi = cf\inhomo;
15.13 FEM 1D Thermal Equilibrium (MATLAB)

Exercise: Recast `fem1d` so it will generate a solution to

\[-\alpha(x) \frac{d^2 \varphi(x)}{dx^2} - \alpha'(x) \frac{d \varphi(x)}{dx} + \beta(x) \varphi(x) = f(x)\]

when

- \(\alpha\) is a (positive) constant and \(\beta = 0\),
- \(f(x)\) varies with position in accordance with \(f(x) = Ae^{-\sigma(x-L/2)^2}\), with \(A\) and \(\sigma\) constants,
- the solution is to have the fixed value \(\varphi = 0\) at \(x = 0\) and the fixed value \(\varphi = 100\) at \(x = L\), i.e., we have a source that is concentrated near the middle of the region of interest and we impose Dirichlet boundary conditions on both ends.

Then, explore the character of the solutions for various values of \(\sigma\). Use two-noded elements of equal size.

Solution: With \(\alpha\) constant and \(\beta = 0\), we seek here a solution to the problem defined by the differential equation

\[-\alpha \frac{d^2 \varphi}{dx^2} = f = Ae^{-\sigma(x-L/2)^2}\]

subject to the boundary conditions

\[\varphi(0) = p_0 \quad ; \quad \varphi(L) = p_L\]

The eighth and ninth numbered equations in Section 15.9.4 with \(\beta = 0\) apply to this case, giving

\[K_{11} = K_{NN} = \frac{\alpha}{l} \quad ; \quad K_{ii} = \frac{2\alpha}{l} \quad (i = 2, 3, \ldots, N-1) \quad ; \quad K_{i+1,i} = K_{i,i+1} = -\frac{\alpha}{l} \quad (i = 1, 2, \ldots, N-1)\]

for the non-zero elements of \([K]\) and

\[b_1 = \frac{1}{2} f(1) l \quad ; \quad b_N = \frac{1}{2} f(M) l \quad ; \quad b_i = \frac{1}{2} (f(i-1) + f(i)) l \quad (i = 2, 3, \ldots, N - 1)\]

for the elements of \([b]\). (We have here supposed a division of the interval \(0 \leq x \leq L\) into \(M = N - 1\) elements of equal length \(l = L/M\).) In terms of these quantities, the basic problem then is conveyed by last numbered equation in Section 15.9.4, which we write out for five terms:

\[
\begin{bmatrix}
K_{11} & K_{12} & K_{13} & K_{14} & K_{15} \\
K_{21} & K_{22} & K_{23} & K_{24} & K_{25} \\
K_{31} & K_{32} & K_{33} & K_{34} & K_{35} \\
K_{41} & K_{42} & K_{43} & K_{44} & K_{45} \\
K_{51} & K_{52} & K_{53} & K_{54} & K_{55}
\end{bmatrix}
\begin{bmatrix}
\varphi_1 \\
\varphi_2 \\
\varphi_3 \\
\varphi_4 \\
\varphi_5
\end{bmatrix}
= 
\begin{bmatrix}
b_1 \\
b_2 \\
b_3 \\
b_4 \\
b_5
\end{bmatrix}
+ 
\begin{bmatrix}
g_1 \\
0 \\
0 \\
0 \\
g_5
\end{bmatrix}
\]

As in the example in the text, continuity of the solution and its derivative across the boundary between elements guarantees that all but the first and last entries in \([g]\) will be zero. We have not bothered to write out the values of \(g_1\) and \(g_5\) because the Dirichlet boundary conditions to be imposed renders their values irrelevant. Indeed, imposing those boundary conditions means that,
among other things, we replace the first equation with the equation $\varphi_1 = p_0$, converting the above equation to

$$
\begin{bmatrix}
1 & 0 & 0 & 0 & 0 \\
K_{21} & K_{22} & K_{23} & K_{24} & K_{25} \\
K_{31} & K_{32} & K_{33} & K_{34} & K_{35} \\
K_{41} & K_{42} & K_{43} & K_{44} & K_{45} \\
K_{51} & K_{52} & K_{53} & K_{54} & K_{55}
\end{bmatrix}
\begin{bmatrix}
\varphi_1 \\
\varphi_2 \\
\varphi_3 \\
\varphi_4 \\
\varphi_5
\end{bmatrix}
= 
\begin{bmatrix}
p_0 \\
\hat{b}_2 \\
\hat{b}_3 \\
\hat{b}_4 \\
\hat{b}_5 + \hat{g}_5
\end{bmatrix}
$$

To restore the symmetry, we recognize that the first term in each of the remaining equations can be transferred to the other side of the equations, yielding

$$
\begin{bmatrix}
1 & 0 & 0 & 0 & 0 \\
0 & K_{22} & K_{23} & K_{24} & K_{25} \\
0 & K_{32} & K_{33} & K_{34} & K_{35} \\
0 & K_{42} & K_{43} & K_{44} & K_{45} \\
0 & K_{52} & K_{53} & K_{54} & K_{55}
\end{bmatrix}
\begin{bmatrix}
\varphi_1 \\
\varphi_2 \\
\varphi_3 \\
\varphi_4 \\
\varphi_5
\end{bmatrix}
= 
\begin{bmatrix}
p_0 \\
\hat{b}_2 - K_{21} p_0 \\
\hat{b}_3 - K_{31} p_0 \\
\hat{b}_4 - K_{41} p_0 \\
\hat{b}_5 - K_{51} p_0 + \hat{g}_5
\end{bmatrix}
$$

A similar incorporation of the boundary condition $\varphi_5 = p_L$ leads to

$$
\begin{bmatrix}
1 & 0 & 0 & 0 & 0 \\
0 & K_{22} & K_{23} & K_{24} & 0 \\
0 & K_{32} & K_{33} & K_{34} & 0 \\
0 & K_{42} & K_{43} & K_{44} & 0 \\
0 & 0 & 0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
\varphi_1 \\
\varphi_2 \\
\varphi_3 \\
\varphi_4 \\
\varphi_5
\end{bmatrix}
= 
\begin{bmatrix}
p_0 \\
b_2 - K_{21} p_0 - K_{25} p_L \\
b_3 - K_{31} p_0 - K_{35} p_L \\
b_4 - K_{41} p_0 - K_{45} p_L \\
p_L
\end{bmatrix}
$$

With the elements of $[K]$ and $\{b\}$ given by earlier equations, this last equation is the equation we must set up and solve to find the desired steady state heat distribution.

With this background, we are ready to describe the how fem1d must be altered to accomplish this solution.

- In BLOCK 1, we must enter values of $M = M$, $\alpha = \alpha$, $A = A$ and $\sigma = \sigma$, $l = L/M = 1$, $p_0 = p0$, and $p_L = pL$. Further, we add a calculation of the actual length $\text{length}$ of the rod.

- In BLOCK 2, we create the initial stiffness matrix with the commands

```matlab
K = zeros(M+1,M+1); # Create (M+1)x(M+1) array of zeros
S = alpha/l; # Evaluate common quantities
S2 = 2.0*S;
T = -S;
K(1,1) = S; # Set diagonal elements of K
for i = 2:M K(i,i) = S2; end
K(M+1,M+1) = S;
for i = 1:M K(i+1,i) = T; end # Set elements above and below
for i = 1:M K(i,i+1) = T; end # main diagonal of K
```

- In BLOCK 3, we create the initial vector of inhomogenieties:
b = zeros(M+1,1);  # Create M+1 element vector of zeros
x = l*[0:M];  # Create array of values of x

f = A*exp(-sigma*(x-length/2.0).^2);

b(1) = 0.5*f(1)*l;  # Set elements of b
for i = 2:M b(i) = 0.5*(f(i-1)+f(i))*l; end
b(M+1) = f(M)*l;

• BLOCK 4 is deleted, since there are no mixed boundary conditions.
• To incorporate the Dirichlet boundary condition at the end \( x = 0 \), we include in BLOCK 5 the statements

\[
K(1,1) = 1.0; \quad \text{# Set first row of K to 1,0,0,...}
\]
for j = 2:M+1 K(1,j) = 0.0; end
b(1) = p0;  # Set first element of b to p0
# Adjust other elements of b,K
for i = 2:M+1 b(i) = b(i) - K(i,1)*p0; end
for i = 2:M+1 K(i,1) = 0.0; end

• To incorporate the Dirichlet boundary condition at the end \( x = L \), we include in BLOCK 5 the statements

\[
K(M+1,M+1) = 1.0; \quad \text{# Set last row of K to 0,...,0,0,1}
\]
for j = 1:M K(M+1,j) = 0.0; end
b(M+1) = pL;  # Set last element of b to pL
# Adjust other elements of b,K
for i = 2:M b(i) = b(i) - K(i,M+1)*pL; end
for i = 1:M K(i,M+1) = 0.0; end

• Finally, we are ready to solve the system of equations that we have created as in BLOCK 6 of the prototype program:

\[
\phi = K\backslash b;
\]

A full listing of the program is contained on the pages at the end of this solution.

Preliminary exploration led to the conclusions that \( M = 50 \) yielded adequately smooth graphs, and that \( A = 2000 \) (substantial heating in the middle of the system) and \( \sigma = 40.0 \) (heating confined to the middle third or so of the system) yield reasonable results. We produce graphs of solution and of the heating function \( f(x) \) for these parameters with the statements

\[
pde_ex13
\]
\[
Enter \text{number of segments (M)}: 50
Enter \text{alpha}: 1.0
Enter \text{A}: 2000.0
Enter \text{sigma}: 40.0
Enter \text{l}: 0.02
Enter \text{p0}: 0.0
Enter \text{pL}: 100.0
subplot(211)
plot(x,phi, 'color','black', 'linewidth',3)
grid on
set(gca, 'fontsize',16)
Graphs of the solution for other parameters are shown in the remainder of Fig. E15.12. Remember that $A$ controls the rate of heating in the center of the rod and $\sigma$ controls the width of the region near the center of the rod that is most strongly heated. For low rates of heating (Fig. E15.12(a)), the temperature distribution is close to a straight line between the temperatures at the two ends. For higher rates of heating, the temperature near the center of the rod rises above that straight line, the more so as the rate of heating is increased. At the highest illustrated rate of heating, the temperature near the center of the rod rises above that at the high-temperature end. When the rate of heating is kept high but the width of the heated region is shrunk (Fig. E15.12(d)), the bump near the center remains but the temperature at that bump is reduced. Somewhat surprisingly, the peak of the temperature in segments (c) and (d) appears to be displaced to the right from the peak in the heating function.

Listing of pde_ex13.m

```matlab
% ***** Command file fem1d.m *****
% ***** Note that, when this command file has completed executing,
% all variables to which it assigns values---and in
% particular x and phi---will be accessible at MATLAB's main
% command level.
%
% BLOCK 1 Request input of necessary parameters and assure that each
% is stored with the proper data type.
%
M = input( 'Enter number of segments (M): ' );
alpha = input( 'Enter alpha: ' );
A = input( 'Enter A: ' );
sigma = input( 'Enter sigma: ' );
l = input( 'Enter l: ' );
p0 = input( 'Enter p0: ' );
pL = input( 'Enter pL: ' );
length = l*M;

% BLOCK 2 Determine coefficient (stiffness) matrix before incorporation
% of boundary conditions.

K = zeros(M+1,M+1); % Create (M+1)x(M+1) array of zeros
S = alpha/l; % Evaluate common quantities
S2 = 2.0*S;
T = -S;
K(1,1) = S; % Set diagonal elements of K
```
Figure E15.12: Solution and inhomogeneity for labeled values of $A$ and $\sigma$. Note the different vertical scales in the graphs of temperature and of $f(x)$. These graphs were produced with MATLAB.

for $i = 2$ to $M$ 
    $K(i,i) = S2$; 
end 
$K(M+1,M+1) = S$;

for $i = 1$ to $M$ 
    $K(i+1,i) = T$; 
end % Set elements above and below
for $i = 1$ to $M$ 
    $K(i,i+1) = T$; 
end % main diagonal of $K$

% BLOCK 3 Create vector of inhomogenieties.

$b = \text{zeros}(M+1,1);$ % Create $M+1$ element vector of zeros
$x = l*[0:M]$; % Create array of values of $x$

$f = A*\exp(-\sigma*(x-\text{length}/2.0)^2);$ 

$b(1) = 0.5*f(1)*l;$ % Set elements of $b$
for $i = 2$ to $M$ 
    $b(i) = 0.5*(f(i-1)+f(i))*l$; 
end 
$b(M+1) = f(M)*l$;
% BLOCK 4 deleted

% BLOCK 5 Incorporate boundary conditions.

\[ K(1,1) = 1.0; \quad \text{% Set first row of } K \text{ to } 1,0,0,... \]
for \( j = 2:M+1 \) \( K(1,j) = 0.0; \) end
\[ b(1) = p_0; \quad \text{% Set first element of } b \text{ to } p_0 \]
% Adjust other elements of b,K
for \( i = 2:M+1 \) \( b(i) = b(i) - K(i,1)*p_0; \) end
for \( i = 2:M+1 \) \( K(i,1) = 0.0; \) end
\[ K(M+1,M+1) = 1.0; \quad \text{% Set last row of } K \text{ to } 0,...,0,0,1 \]
for \( j = 1:M \) \( K(M+1,j) = 0.0; \) end
\[ b(M+1) = p_L; \quad \text{% Set last element of } b \text{ to } p_L \]
% Adjust other elements of b,K
for \( i = 2:M \) \( b(i) = b(i) - K(i,M+1)*p_L; \) end
for \( i = 1:M \) \( K(i,M+1) = 0.0; \) end

% BLOCK 6 Solve system using LU decomposition.

\[ \phi = K\backslash b; \]
15.23 FDM 2D Laplace Equation: Alternate Convergence Test (MATLAB)

**Exercise:** Recast fdmlap2d so that iteration stops when the largest change occurring at any single node from one iterate to the next does not exceed an externally prescribed tolerance. To avoid all possibility of an infinite loop, you should halt iteration *either* when the tolerance has been reached (or exceeded) *or* when some prescribed number of iterations has taken place. Code so that your program displays the actual tolerance achieved. Further, if execution terminates because the prescribed tolerance is *not* achieved, your program should print a message that alerts you to the fact that the prescribed tolerance was not achieved. Test your program with the same example as was used in the text. *Hints:* Before starting an iteration, set a variable, say `maxch`, equal to zero. As you calculate a new value for each node in the iteration, store the result in a temporary variable so you can compare that value with the old value it will replace, updating `maxch` to the absolute value of the difference between the new and the old values *if and only if* that difference exceeds the difference already stored in `maxch`. Then, substitute the new value for the old in the array containing the evolving solution and go on to the next node. Once the iteration is completed, `maxch` will contain the absolute value of the largest change at any node during that single iteration. If `maxch` is less than the prespecified convergence criterion, stop the iteration; otherwise conduct one more iteration.

**Solution:** The changes to be made to `fdmlap2d.pro` in MATLAB to reflect the requirements in this exercise involve

- Reading in a desired tolerance (i.e., maximum amount by which values in successive iterates can change before terminating iteration) by adding the statement

  ```matlab
tol = input( 'Tolerance (tol): ' );
```

- Changing the outermost `for` loop to a `while` loop so that the second stopping criterion can be incorporated more easily. Leaving that second criterion aside for a moment, we replace the outermost loop with the loop

  ```matlab
  itcnt = 0;
  while itcnt < maxits
    .
    .
    itcnt=itcnt+1;
  endwhile
  ```

  Here, we have recognized that the `while` loop requires us also to initialize the iteration counter outside the loop and then increment it in the appropriate point inside the loop. Note also that this counter is incremented after the counted iteration has been completed so, when we return to start the next path through the loop, `itcnt` contains the number of iterations completed. Thus, the while loop should stop when `itcnt` has the value `maxits`. If we expressed the criterion as `itcnt le maxits`, the loop would be executed one too many times.

- (to keep track of the change from one iteration to the next) Saving the old value every time a new value of `u[i,j]` is calculated and then compare the old value with the new value and adjust a record—we suppose that record to be kept in a variable `change`, which must be initialized appropriately—of the largest value if the new difference is larger than the old difference, i.e., we must replace every calculation of a new entry for `u(i,j)` with

  ```matlab
tmp = u(i,j);
  u(i,j) = 0.25*(u(i+1,j) + u(i-1,j) + u(i,j+1) + u(i,j-1));
  chg = abs( tmp-u(i,j) );
  if chg > change change = chg; end
  ```
and every calculation of a new entry for $u(i,N+1)$ with

\[
\text{tmp} = u(i,N+1);
\]

\[
u(i,N+1) = 0.25 \times (2.0 \times u(i,N) + u(i-1,N+1) + u(i+1,N+1));
\]

\[
\text{chg} = \text{abs}(\text{tmp} - u(i,N+1));
\]

\[
\text{if chg > change change = chg; end}
\]

Every time we start a new iteration, we must initialize change to a value smaller than any likely actual change; we choose 0.0, so the statement

\[
\text{change}=0.0
\]

must be inserted immediately after the while loop is entered.

- Adding the second criterion to the condition in the while statement by editing that statement to read

\[
\text{while itcnt < maxits & change > tol}
\]

This addition means that, before the loop is entered, change needs to be initialized to a value that is larger than tol so that the loop will not be terminated immediately. We therefore add the statement

\[
\text{change} = 2.0 \times \text{tol};
\]

immediately ahead of the while statement.

- Finally, when the coding exits from the while loop, we need to display a message conveying whether convergence was achieved or not and indicating the number of iterations completed. Further, since convergence may be achieved at an iteration that is otherwise not printed, we need to print the final iterate outside the loop to be sure we see it is printed no matter how many iterations are completed. Thus, we add the statements

\[
\text{fprintf(’Number of iterations completed \%d’, itcnt )}
\]

\[
\text{fprintf(’\nTolerance achieved = \%f’, change)}
\]

\[
\text{if change > tol}
\]

\[
\text{fprintf(’\nSpecified tolerance \%f not achieved’, tol )}
\]

\[
\text{else}
\]

\[
\text{fprintf(’\nSpecified tolerance \%f achieved’, tol )}
\]

\[
\text{end}
\]

\[
\text{fprintf(’\nFinal Solution = \n’)}
\]

\[
u
\]

after the while loop.

The command file is named pde_ex23.pro and is listed at the end of the solution to this exercise.

A full test of this program will make sure that it stops properly both when convergence is achieved and when convergence fails. As a start, we run it in MATLAB with the input

\[
pde_ex23
\]

\[
\text{Number of segments: } 5
\]

\[
\text{Length of side: } 10
\]

\[
\text{Maximum number of iterations: } 10
\]

\[
\text{Display frequency: } 5
\]

\[
\text{Tolerance: } 2
\]
which generates the output

**Iteration 0**

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>80</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>60</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>40</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>20</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

**Iteration 5**

|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 100.0000 | 100.0000 | 100.0000 | 100.0000 | 100.0000 | 100.0000 | 80.0000 | 73.9017 | 70.1803 | 68.4650 | 68.2961 | 69.2642 |
| 60.0000 | 51.7524 | 46.5277 | 43.9893 | 43.6158 | 44.8891 |
| 40.0000 | 33.0073 | 28.4468 | 26.1477 | 25.7376 | 26.7981 |
| 0   | 0   | 0   | 0   | 0   | 0   |

Number of iterations completed 9
Tolerance achieved = 1.712425
Specified tolerance 2.000000 achieved
Final Solution =

|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 100.0000 | 100.0000 | 100.0000 | 100.0000 | 100.0000 | 100.0000 | 80.0000 | 78.0059 | 76.6798 | 75.9948 | 75.8662 | 76.1778 |
| 60.0000 | 57.1748 | 55.2876 | 54.3078 | 54.1200 | 54.5610 |
| 40.0000 | 37.5224 | 35.8621 | 34.9969 | 34.8288 | 35.2153 |
| 20.0000 | 18.6549 | 17.7518 | 17.2802 | 17.1879 | 17.3978 |
| 0   | 0   | 0   | 0   | 0   | 0   |

We have deliberately picked a large tolerance, expecting that convergence would be achieved in a very few iterations. Note that (1) the program has worked correctly to display the final solution even though it didn’t occur at a multiple of 5 iterations, (2) the actual tolerance achieved is indeed smaller than the specified desired tolerance, and (3) the proper message regarding the achievement of tolerance has been output.

As a second test, let us suppress all intermediate output by making the display frequency larger than the maximum number of iterations and run the program with a smaller tolerance, e.g., with the input

```
pde_ex23
```

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of segments:</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Length of side:</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Maximum number of iterations:</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Display frequency:</td>
<td>20</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tolerance:</td>
<td>0.1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

This time, we are presented with the output

**Iteration 0**

|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 100 | 100 | 100 | 100 | 100 | 100 | 80  | 0   | 0   | 0   | 0   | 0   |
| 60  | 0   | 0   | 0   | 0   | 0   |
| 40  | 0   | 0   | 0   | 0   | 0   |
Exercise 15.23 (MATLAB)

\[
\begin{bmatrix}
20 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 \\
\end{bmatrix}
\]

Number of iterations completed 10
Tolerance achieved = 1.325932
Specified tolerance 0.100000 not achieved
Final Solution =

\[
u = \begin{bmatrix}
100.0000 & 100.0000 & 100.0000 & 100.0000 & 100.0000 & 100.0000 \\
80.0000 & 78.4636 & 77.4365 & 76.9026 & 76.8001 & 77.0403 \\
60.0000 & 57.8184 & 56.3562 & 55.5939 & 55.4460 & 55.7869 \\
40.0000 & 38.0838 & 36.7972 & 36.1251 & 35.9936 & 36.2930 \\
20.0000 & 18.9589 & 18.2591 & 17.8930 & 17.8211 & 17.9838 \\
0 & 0 & 0 & 0 & 0 & 0 \\
\end{bmatrix}
\]

which shows only the initial guess and the final solution, indicates that the full ten iterations were used and that the specified tolerance was not achieved in those ten iterations.

Next, we set the maximum number of iterations quite high, specify a still smaller tolerance, and—with luck—will achieve convergence to a more accurate final solution. We use the input

\[
pde_ex23
\begin{align*}
\text{Number of segments:} & \quad 5 \\
\text{Length of side:} & \quad 10 \\
\text{Maximum number of iterations:} & \quad 100 \\
\text{Display frequency:} & \quad 200 \\
\text{Tolerance:} & \quad 0.01
\end{align*}
\]

and find the output

Iteration 0
\[
u =
\begin{bmatrix}
100 & 100 & 100 & 100 & 100 & 100 \\
80 & 0 & 0 & 0 & 0 & 0 \\
60 & 0 & 0 & 0 & 0 & 0 \\
40 & 0 & 0 & 0 & 0 & 0 \\
20 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 \\
\end{bmatrix}
\]

Number of iterations completed 30
Tolerance achieved = 0.007994
Specified tolerance 0.010000 achieved
Final Solution =

\[
u = \begin{bmatrix}
100.0000 & 100.0000 & 100.0000 & 100.0000 & 100.0000 & 100.0000 \\
80.0000 & 79.9908 & 79.9846 & 79.9814 & 79.9807 & 79.9822 \\
60.0000 & 59.9889 & 59.9781 & 59.9735 & 59.9726 & 59.9746 \\
0 & 0 & 0 & 0 & 0 & 0 \\
\end{bmatrix}
\]
The solution converged to within 0.01 in 30 iterations.

Finally, seeking a solution that coincides with the fully converged solution obtained in the text, we set the tolerance to 0.00001 with the input

```
pde_ex23
    Number of segments: 5
    Length of side: 10
    Maximum number of iterations: 100
    Display frequency: 200
    Tolerance: 0.00001
```

The resulting output is

```
Iteration 0
u = 100 100 100 100 100 100  80  0  0  0  0  0  60  0  0  0  0  0
  40  0  0  0  0  0  20  0  0  0  0  0  0  0  0  0  0
Number of iterations completed 57
Tolerance achieved = 0.000008
Specified tolerance 0.000010 achieved
Final Solution =

u = 100.0000 100.0000 100.0000 100.0000 100.0000 100.0000 80.0000 80.0000 80.0000 80.0000 80.0000 80.0000 60.0000 60.0000 60.0000 60.0000 60.0000 60.0000 40.0000 40.0000 40.0000 40.0000 40.0000 40.0000 20.0000 20.0000 20.0000 20.0000 20.0000 20.0000 0 0 0 0 0 0
```

Convergence to what is, in fact, the exact solution for this problem, is achieved in 57 iterations.

The same embellishments described in Section 15.18.3 of the text could now be invoked to write a more refined solution to a file and then to create graphs of that solution. Those embellishments were not requested in this exercise.
% ***** Command file pde_ex23.m *****

% ***** Acquire controlling parameters *****

N = input( 'Enter number of segments (N): ' ); N = fix(N);
L = input( 'Enter length of side(L): ' );
maxits = input( 'Maximum number of iterations (maxits): ' );
maxits = fix( maxits );
f = input( 'Display frequency (f): ' ); f=fix(f);
tol = input( 'Tolerance (tol): ' );

% ***** Calculate grid spacing, values of x and y at grid points *****

dx = L/N;
x = dx*[0 : N ]; y = x;

% ***** Create and initialize array for solution *****

u = zeros(N+1, N+1);
u(1,:) = 100.0;
u(:,1) = 100.0 - 100.0*x/L;

% ***** Display solution on the screen *****

fprintf( '
Iteration 0' )
u

% ***** Iterate the specified number of times, displaying %
% every f-th iterate on the screen *****

itcnt = 0;
change = 2.0*tol;
while itcnt < maxits & change > tol
    change = 0.0;
    for i = 2:N
        for j = 2:N
            tmp = u(i,j);
            u(i,j) = 0.25*(u(i+1,j) + u(i-1,j) + u(i,j+1) + u(i,j-1));
            chg = abs( tmp-u(i,j) );
            if chg > change change = chg; end
        end
        tmp = u(i,N+1);
        u(i,N+1) = 0.25*(2.0*u(i,N) + u(i-1,N+1) + u(i+1,N+1));
        chg = abs( tmp-u(i,N+1) );
        if chg > change change = chg; end
    end
    itcnt = itcnt + 1;
    if f*fix(itcnt/f) == itcnt
        fprintf( '
Iteration %d', itcnt )
u
    end
end
fprintf('Number of iterations completed %d', itcnt )
fprintf('Tolerance achieved = %f', change)
if change > tol
    fprintf('Specified tolerance %f not achieved', tol )
else
    fprintf('Specified tolerance %f achieved', tol )
end
fprintf('Final Solution = 
')
u
15.24 FEM 2D Poisson Equation (MATLAB)

Exercise: Consider the problem defined by the two-dimensional Poisson equation
\[ \frac{\partial^2 U}{\partial x^2} + \frac{\partial^2 U}{\partial y^2} = -(1 - x^2)(1 - y^2) \]
to be solved in the square region \( R \) defined by \(-1 \leq x, y \leq 1 \) subject to the Dirichlet boundary conditions requiring \( U = 0 \) on the entire boundary of \( R \). Assume all variables are dimensionless. (a) Obtain a surface plot of this inhomogeneity over the \( x, y \) plane. (b) Recasting \texttt{fem2d} to incorporate the given boundary conditions and to address the non-zero inhomogeneity find, explore, and (using surface and contour plots as you deem appropriate) display the solution \( U(x, y) \) in the square region \( R \). Compare the resulting equation with Eq. (15.70) as a guide to interpreting the present problem as a modification of the problem addressed by \texttt{fem2d}. (c) Find the two-dimensional vector field \( \mathbf{V} = -\nabla U = -\nabla U \) in \( R \) and display that field graphically in whatever ways seem appropriate.

Solution: (a) The inhomogeneity is given by
\[ f(x, y) = (1 - x^2)(1 - y^2) \]

We create a surface plot of this function with the statements

```matlab
xx=[-1.0:0.1:1.0]; yy=[-1.0:0.1:1.0];
[x,y] = meshgrid(xx,yy);
sig = -(1.0-x.^2).*(1.0-y.^2);
mesh( x, y, sig, 'linewidth',2, 'edgecolor','black')
set(gca, 'fontsize',16 )
xlabel('x','fontsize',20); ylabel('y', 'fontsize',20)
zlabel('-(1-x^2)(1-y^2)', 'fontsize',20 )
title('Inhomogeneity','fontsize',20)
```

The resulting figure is shown in Fig. E15.13.

(b) Our task is to solve the equation
\[ \frac{\partial^2 U}{\partial x^2} + \frac{\partial^2 U}{\partial y^2} = -(1 - x^2)(1 - y^2) \] (E15.24.1)
in the region \(-1 \leq x, y \leq 1 \) subject to the boundary conditions
\[ U(-1, y) = U(1, y) = U(x, -1) = U(x, 1) = 0 \] (E15.24.2)

If in Eq. (15.70) in CPSUP we set
\[ \alpha_x = -1 ; \quad \alpha_y = -1 ; \quad \beta = 0 ; \quad f(x, y) = -(1 - x^2)(1 - y^2) \] (E15.24.3)
we can then justify using \texttt{fem2d.m} as a starting point in solving this problem.

To adapt \texttt{fem2d.m}, we need to make several adjustments. First, we must set the value of \( L \) to 1.0 and we can hard code the values of \( \alpha_x, \alpha_y, \) and \( \beta \) by replacing the statements requesting those values with the statements

\[ L = 1.0; \text{alpha}_x = -1.0; \text{alpha}_y = -1.0; \text{beta} = 0.0; \]

(Note that \( L \) in this exercise is half the side of the square region.) Further, we must change the calculation of \( \text{dx} \) to the value 2.0/\( d \) and the values of \( x \) and \( y \) to the range \(-1.0 \leq x, y \leq 1.0 \) involved in this exercise. To do so, we replace the line calculating \( \text{dx} \) and lines in the loop calculating the values of \( x \) and \( y \) with the lines
dx = 2.0/d; % Calculate dx
x(ct) = i*dx - L; % Find x coordinate
y(ct) = L - j*dx; % Find y coordinate

For this change to work well, we need also constrain \( d \) to be even, so that there will actually be a node at the center of the square, i.e., at \((x, y) = (0, 0)\).

Next, we must replace the constant value of \( f \) with a vector of the proper dimension, store in each of its elements the values of the inhomogeneity at the corresponding node, and replace \( f \) when it appears in the subsequent coding with the proper element of the vector. Thus, we

- Delete the line requesting input of the value of \( f \);
- Insert the line
  \[
  f = -(1.0-x.^2).*(1.0-y.^2);
  \]
  after the nodal coordinates have been inserted in the two vectors \( x \) and \( y \); and
- Replace \( f \) with \( f(cm(i,e)) \) in the one place in which it appears in the segment creating the stiffness matrix.

We need also to make some adjustments in the boundary conditions. The problem to which fem2d.m applies directly had the value \( p_1 \) on the top edge, \( p_2 \) on the bottom edge, \((p_1 - p_2)y/L + p_2\) along the left edge, and the condition \( \partial \phi/\partial x = q \) along the right edge. The current problem has the value zero on all four edges. We can accomplish that assignment on top, bottom, and left edges simply by

- Deleting the two lines requesting the entry of values for \( p_1 \) and \( p_2 \);
Exercise 15.24 (MATLAB)

- Deleting the line
  \[ p3 = (p1-p2)/L \ast y(u) + p2 \]
  \[ \text{; Find values of } \phi \text{ on left boundary} \]
  from the segment titled Incorporate Dirichlet boundary conditions; and
- Adding the lines
  \[
  p1 = 0.0 \quad \# \text{Set boundary values on top, bottom, and left} \\
  p2 = 0.0 \\
  p3 = 0.0
  \]

  near the beginning of the coding.

Replacing the Neumann condition in the template we are using with the required Dirichlet condition on the right edge takes a bit more effort. We

- Remove altogether the segment headed Incorporate Neumann boundary conditions;
- Remove the line requesting input of the value of \( q \), which enters only in the statement of the Neumann boundary conditions;
- Add the line \( p4 = 0.0 \) near the beginning of the coding;
- Add the lines
  \[
  v = d\ast(d+1) + i+1; \quad \% \text{Nodes on the right boundary} \\
  K(j,v) = 0; \\
  b(v) = p4; \\
  \text{if } j \neq v \quad \text{b(j) = b(j) - K(v,j)*p4;} \\
  \end{\text{if}} \\
  K(v,j) = 0; \\
  K[v,v] = 1;
  \]

  at the proper places in the segment headed Incorporate Dirichlet boundary conditions.
  (These additions parallel those already present in that segment and add proper treatment of the fourth edge. In particular, the first line sets \( v \) to the global node number for the nodes on the right edge.)

The file constructed from fem2d.pro with the above enumerated modifications is named pde_ex24.pro and is listed at the end of the solution to this exercise.

After a bit of exploration, selecting a solution with 32 segments on each side seems appropriate. The statements

\[
\text{pde_ex24} \\
\text{Enter number of segments (d): 32} \\
A32 = A;
\]

produces that solution and saves the result in \( A32 \). The statements

\[
[xx,yy] = \text{meshgrid}(-1.0:dx:1.0, -1.0:dx:1.0); \\
\text{mesh}(xx,yy,A32, 'edgecolor','black', 'linewidth',2) \\
\text{set(gca, 'fontsize',16 )} \\
\text{xlabel('x','fontsize',20); ylabel('y', 'fontsize',20) } \\
\text{zlabel('U', 'fontsize',20 )} \\
\text{title('Mesh Graph of U','fontsize',20)}
\]
Exercise 15.24 (MATLAB)

Figure E15.14: Mesh graph of $U(x,y)$ in Exercise 15.24. This graph was produced with MATLAB.

create the mesh graph of Fig. E15.14 and the statements

```matlab
levs = [0.025:0.025:0.2];
contour(xx, yy, A32, levs, 'k', 'linewidth', 3)
set(gca, 'fontsize', 16)
xlabel('x','fontsize',20); ylabel('y', 'fontsize', 20)
title('Contour Map of U','fontsize', 20)
axis('square')
A32(17,17)
ans = 0.2131
```

create the contour map of Fig. E15.15. The last statement reveals that the value at the center $(x,y) = (0,0)$ is $U(0,0)/U_0 = 0.2131$.

To assess convergence, we also generate a solution with 64 divisions of each side and compare the center potential from each solution with the statements

```matlab
pde_ex24
Enter number of segments (d): 64
A64 = A;
A64(33,33)
ans = 0.2129
A32(17,17)-A64(33,33)
ans = 1.1713e-04 (i.e., 0.00011713)
```

We learn that the central value of the potential in the (presumably) more accurate solution decreases by only 0.00011713 (0.01%) when the grid is refined from 32 to 64 divisions. We can examine the
Figure E15.15: Contour map of \( U(x,y) \) in Exercise 15.24. Starting at the outside border of this map, the values of \( U/U_0 \) are 0.000, 0.025, 0.050, 0.075, ..., 0.200. The border of this region is at potential 0. This graph was produced with MATLAB.

\[
\begin{array}{c}
-1 & -0.5 & 0 & 0.5 & 1 \\
-1 & -0.8 & -0.6 & -0.4 & -0.2 & 0 & 0.2 & 0.4 & 0.6 & 0.8 & 1
\end{array}
\]

Differences at all points on the smaller grid by extracting the \( 33 \times 33 \) grid from the \( 65 \times 65 \) grid and then subtracting the one from the other with the statements

\[
\begin{align*}
A64red &= \text{zeros}(33,33); \\
\text{for } i &= 1:33 \\
\quad &\text{for } j = 1:33 \\
\quad &\quad A64red(i,j) = A64(2*i-1,2*j-1); \\
\end{align*}
\]

\[
df = A32-A64red;
\]

The difference at the center generated with these statements is given by

\[
df(17,17) \\
\text{ans} = 1.1713e-04 \quad \text{(i.e., 0.00011713)}
\]

which agrees with the previously calculated difference at the center. Finally we execute the statement

\[
[\min(\min(df)), \max(\max(df))]
\]

\[
\text{ans} = 1.0e-03 \times (0 \quad 0.1171) \quad \text{(i.e., 0.00000000 \quad 0.00011711)}
\]

The first value tells us that all differences are positive; the second tells us that the greatest difference occurs at the center. We conclude that the solution on the 32 \( \times \) 32 grid differs from that on a 64 \( \times \) 64 grid by no more than 0.00012. (Repeating the calculation with \( d = 128 \) yields the central value of
0.2129, in complete agreement to four decimal places with the result obtained with the 64 × 64 grid. We conclude that the solution on the 64 × 64 grid has converged to the exact solution to within four digits after the decimal point.

(c) The negative gradient of \( U \) can be evaluated with the statement

\[
[Vx32, Vy32] = \text{gradient}(-A32);
Vx16=\text{zeros}(17,17); Vy16= \text{zeros}(17,17);
\]

\[
\text{for } i = 1:17
\]
\[
\quad \text{for } j = 1:17
\]
\[
\quad \quad Vx16(i,j) = Vx32(2*i-1,2*j-1);
\]
\[
\quad \quad Vy16(i,j) = Vy32(2*i-1,2*j-1);
\]
\[
\text{end}
\]
\[
\text{end}
\]

Here, for the sake of a more transparent field map, we have elected to extract a 16 × 16 array of field values from the originally computed 32 × 32 array. Now, we plot the field and overlay a contour map with the statements

\[
dx = [-1.0:2.0/16.0:1.0];
[x16,y16]=\text{meshgrid}(dx,dx);
quiver(x16,y16,Vx16,Vy16, 'k', 'linewidth',2) \quad \% \text{Map field}
\]
\[
\text{hold on}
dx = [-1.0:2.0/32.0:1.0];
[x32,y32]=\text{meshgrid}(dx,dx);
levs = [0.025:0.025:0.2];
\text{contour}(x32, y32, A32, levs, 'k', 'linewidth',2) \quad \% \text{Overlay contour map}
\text{axis([-1.5,1.5,-1.5,1.5], 'square')}
\text{set(gca, 'fontsize',16 )}
\text{xlabel('x','fontsize',20); ylabel('y', 'fontsize',20)}
\quad \% \text{Add boundary at all points of which the solution is zero}
\text{plot([-1.0,-1.0,1.0,1.0,-1.0,1.0,-1.0,-1.0], ...}
\quad \quad 'color','black', 'linewidth',2)
\]

The resulting graph is shown in Fig. E15.16. Note particularly the expected invariance of the graph under 90°, 180° and 270° rotation about a vertical line through the center at \((x,y) = (0.0)\).
Figure E15.16: Field and overlaid contour map of the solution in Exercise 15.24. Starting at the outside border of this map, the values of $U$ are 0.000, 0.025, 0.050, 0.075, ..., 0.200. The border of this region is at potential 0. This graph was produced with MATLAB.

Listing of pde_ex24.m

```matlab
% ***** Command file pde_ex24.m *****

% ***** Enter parameters *****

d = input( 'Enter number of segments (d): ' ); d = fix(d);
L = 1.0; alpha_x = -1.0; alpha_y = -1.0; beta = 0.0;
p1 = 0.0; p2 = 0.0; p3 = 0.0; % Set boundary values on top, bottom, left
p4 = 0.0; % Set boundary value on right

% ***** Find number of elements M, nodes N, and segment side d *****

M = 2*d^2; % Calculate number of elements
N = (d+1)^2; % Calculate number of nodes
dx = 2.0/d; % Calculate segment size

% ***** Find coordinates of nodes *****

x = zeros(1,N); % Create array to store x values
y = zeros(1,N); % Create array to store y values
c = 0; % Initialize a counter variable
for i = 0:d % Start row number loop
    for j = 0:d % Start column number loop
```
ct=ct+1; % Increment counter
x(ct) = i*dx - L; % Find x coordinate
y(ct) = L - j*dx; % Find y coordinate
end % End loops
end
f = -(1.0-x.^2).*(1.0-y.^2); % Set inhomogeneity

% ***** Create connectivity matrix *****

cm = zeros(3,M); % Create 3 by M null array
for e = 1:M % Loop through all elements
    vl = fix((e-1)/(2*d)) + 1; % Find index of vertical line
    cm(1,e) = fix((e-1)/2) + 1 + vl; % Find global number of node 1
    if 2*fix(e/2) == e % If e is even
        cm(2,e) = cm(1,e) + d + 1; % find global number of node 1
        cm(3,e) = cm(1,e) + d; % find global number of node 2
    else % If e is odd
        cm(2,e) = cm(1,e) + d; % find global number of node 1
        cm(3,e) = cm(1,e) - 1; % find global number of node 2
    end
end

% ***** Find [K] for odd and even numbered elements *****

Kodd = zeros(3,3); % Create two 3 by 3 arrays
Keven = zeros(3,3);
bx = beta*dx^2; % Evaluate a common quantity
Keven(1,1) = (bx + 6*alpha_x)/12; % Assign the appropriate value to
Keven(2,2) = (bx + 6*alpha_x + 6*alpha_y)/12; % each K(i,j). Note that the array
Keven(3,3) = (bx + 6*alpha_y)/12; % are symmetric, and that Kodd
Keven(1,2) = (bx - 12*alpha_x)/24; % includes all of the same values as
Keven(2,1) = Keven(1,2); % Keven, but in different locations.
Keven(2,3) = (bx - 12*alpha_y)/24;
Keven(3,2) = Keven(2,3);
Keven(1,3) = bx/24;
Keven(3,1) = bx/24;

Kodd(1,1) = Keven(3,3);
Kodd(2,2) = Keven(1,1);
Kodd(3,3) = Keven(2,2);
Kodd(1,2) = Keven(1,3);
Kodd(2,1) = Kodd(1,2);
Kodd(2,3) = Keven(1,2);
Kodd(3,2) = Kodd(2,3);
Kodd(1,3) = Keven(2,3);
Kodd(3,1) = Kodd(1,3);

% ***** Create stiffness matrix and vector b *****

K = zeros(N,N); % Create arrays to store values of
b = zeros(N,1); % K(i,j) and b(i)
for e = 1:M
    for i = 1:3
        b(cm(i,e)) = b(cm(i,e)) + f(cm(i,e))*dx^2/6; % place its contributions at the
        for j = 1:3
            if 2*fix(e/2) ~= e
                K(cm(i,e),cm(j,e)) = K(cm(i,e),cm(j,e)) + Kodd(i,j);
            else
                K(cm(i,e),cm(j,e)) = K(cm(i,e),cm(j,e)) + Keven(i,j);
            end
        end
    end
end

% ***** Incorporate Dirichlet boundary conditions *****
for i = 0:d
    u = i + 1; % Nodes on the left boundary
    s = i*(d+1) + 1; % Nodes on the top boundary
    t = (i+1)*(d+1); % Nodes on the bottom boundary
    v = d*(d+1) + i+1; % Nodes on the right boundary
    for j = 1:N
        K(j,s) = 0; % Set rows in K to zero where value
        K(j,t) = 0; % of phi is known
        K(j,u) = 0;
        K(j,v) = 0;
        b(s) = p1; % Set values of b
        b(t) = p2;
        b(u) = p3;
        b(v) = p4;
        if j ~= s % Reflect influence
            b(j) = b(j) - K(s,j)*p1; % of known values of
        end % phi in the other
        if j ~= t
            b(j) = b(j) - K(t,j)*p2;
        end
        K(t,j) = 0;
        if j ~= u
            b(j) = b(j) - K(u,j)*p3;
        end
        K(u,j) = 0;
        if j ~= v
            b(j) = b(j) - K(v,j)*p4;
        end
        K(v,j) = 0;
    end
    K(s,s) = 1; % Set the appropriate entry to 1 in
    K(t,t) = 1; % the rows where phi is known
    K(u,u) = 1;
    K(v,v) = 1;
end

% ***** Solve the system of equations *****
\[ \phi = K_i \] \hspace{1cm} \% Solve the equation \( K \phi = b \)  
\hspace{1cm} \% writing the results to \( \phi \) 

\% ***** Store solution and node locations in an array *****

\begin{verbatim}
A = zeros(d+1,d+1); \hspace{1cm} \% Create a d+1 by d+1 array
ff = zeros(d+1,d+1); \hspace{1cm} \% Initialize a counter
cnt=0;
for i = 1:d+1 \hspace{1cm} \% Use nested for loops to write all
    for j = d+1:-1:1 \hspace{1cm} \% entries in \( A \)
        cnt=cnt+1; \hspace{1cm} \% Increment counter
        A(i,j) = phi(cnt);
        ff(i,j) = f(cnt);
    end \hspace{1cm} \% End inner loop
end \hspace{1cm} \% End outer loop
\end{verbatim}