Differential Equations with Linear Algebra: *Mathematica* Help

Matthew R. Boelkins
Grand Valley State University

J. L. Goldberg
University of Michigan

Merle C. Potter
Michigan State University

October 13, 2009

Preface to Mathematica Help

The purpose of this supplement to *Differential Equations with Linear Algebra* is to provide some basic support in the use of *Mathematica*, analogous to the subsections of the text itself that offer similar guidance in the use of *Maple*. In the following pages, the user will find parallel sections to those in the text titled "Using *Maple* to . . .". In particular, the following sections can be found here:

1.2.1 Row Reduction using *Mathematica*

Obviously one of the problems with the process of row reducing a matrix is the potential for human arithmetic errors. Soon we will learn how to use computer software to execute all of these computations quickly; first, though, we can deepen our understanding of how the process works, and simultaneously eliminate arithmetic mistakes, by using a computer algebra system in a step-by-step fashion. In this supplement to the text, we use the software *Mathematica*. For now, we only assume that the user is familiar with *Mathematica*'s interface, and will introduce relevant commands with examples as we go. Recall that *Mathematica* provides a user prompt such as `In[1]:=`, and upon our entering code and pressing SHIFT+ENTER, generates output at the prompt `Out[1]:=`. In what follows, we will not include the prompts `In[1]:=` and `Out[1]:=`, nor will we remind the reader that holding the shift key while pressing the ENTER key is required.

To demonstrate various commands, we will revisit the system from Example 1.2.1. The reader should explore this code actively by entering and experimenting on his or her own. Recall that we were interested in row reducing the augmented matrix

$$\begin{bmatrix} 3 & 2 & -1 & 8 \\ 1 & -4 & 2 & -9 \\ -2 & 1 & 1 & -1 \end{bmatrix}$$

Mathematica stores vectors and matrices as lists and lists of lists, respectively. We enter the augmented matrix, say **A**, row-wise in *Mathematica* with the command

```
A = {{3, 2, -1, 8},{1, -4, 2, -9},{ -2, 1, 1, -1}}
```

to which the program responds with the somewhat unenlightening output

```
{{3, 2, -1, 8},{1, -4, 2, -9},{-2, 1, 1, -1}}
```

To see the output in matrix format, enter

```
MatrixForm[A]
```

The above can be accomplished in a single command line by entering

```
A = {{3, 2, -1, 8},{1, -4, 2, -9},{-2, 1, 1, -1}};
MatrixForm[A]
```

We can access row 1 of the matrix `A` with the syntax `A[[1]]`, and similarly work with any other row of the matrix. This allows us to execute elementary row operations.

To work on the row reduction in the desired example, we first want to swap rows 1 and 2; this is accomplished by using a temporary variable "`temp`" in the following way:

```
A1 = A;
temp = A1[[1]];
A1[[1]] = A1[[2]];
A1[[2]] = temp;
MatrixForm[A1]
```

Note that this stores the result of this row operation in the matrix $\mathbf{A}1$, which is convenient for use in the next step. The semicolon (`;`) suppresses the output of the command that precedes it. After executing the most recent set of commands, the following matrix will appear on the screen:

$$\begin{pmatrix} 1 & -4 & 2 & -9 \\ 3 & 2 & -1 & 8 \\ -2 & 1 & 1 & -1 \end{pmatrix}$$

To perform row replacement, our next step is to add $(-3) \cdot R_1$ to R_2 (where rows 1 and 2 are denoted R_1 and R_2) to generate a new second row; similarly, we will add $2 \cdot R_1$ to R_3 for an updated row 3. The commands that accomplish these steps are

```
A2 = A1;
A2[[2]] = -3*A1[[1]] + A1[[2]];
A2[[3]] = 2*A1[[1]] + A1[[3]];
MatrixForm[A2]
```

and lead to the following output:

$$\begin{pmatrix} 1 & -4 & 2 & -9 \\ 0 & 14 & -7 & 35 \\ 0 & -7 & 5 & -19 \end{pmatrix}$$

Next we will scale row 2 by a factor of $1/14$ and row 3 by $-1/7$ using the commands

```
A3 = A2;
A3[[2]] = 1/14 * A2[[2]];
A3[[3]] = -1/7 * A2[[3]];
MatrixForm[A3]
```

to find that $\mathbf{A}3$ has the entries

$$\begin{pmatrix} 1 & -4 & 2 & -9 \\ 0 & 1 & -\frac{1}{2} & \frac{5}{2} \\ 0 & -7 & 5 & -19 \end{pmatrix}$$

The remainder of the computations in this example involve slightly modified versions of the three versions of the commands demonstrated above, and are left as an exercise for the reader. Recall that the unique solution to the original system is $(1, 2, -1)$.

Mathematica is certainly capable of performing all of these steps at once. After completing each step-by-step command above in the row-reduction process, the result can be checked by executing the command

```
RowReduce[A];
MatrixForm[%]
```

The corresponding output should be

$$\begin{pmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 2 \\ 0 & 0 & 1 & -1 \end{pmatrix}$$

which clearly reveals the unique solution to the system, $(1, 2, -1)$.

1.3.2 Matrix Products using *Mathematica*

After becoming comfortable with computing elementary matrix products by hand, it is useful to see how *Mathematica* can assist us with more complicated computations. Here we demonstrate the relevant command.

Revisiting Example 1.3.2, to compute the product $\mathbf{Ax}$, we first enter $\mathbf{A}$ and $\mathbf{x}$ using the familiar commands

```
A = {{1, -3},{-4, 1},{2, 5}}
x = {-5, 2}
```

Next, we use the "period" symbol to inform *Mathematica* that we want to multiply. Entering

```
b = A.x;
MatrixForm[b]
```

yields the expected output that

$$\begin{pmatrix} -11 \\ 22 \\ 0 \end{pmatrix}$$

Note: *Mathematica* will obviously only perform the multiplication when it is defined. If, say, we were to attempt to multiply $\mathbf{A}$ and $\mathbf{x}$ where

```
A = {{1, -4, 2},{-3, 1, 5}}
x = {-5, 2},
```

then *Mathematica* would report the following:

```
Dot::dotsh:  Tensors{{1,-4,2},{-3,1,5}} and {-5,2} have incompatible shapes.
```

1.7.1 Matrix Algebra using *Mathematica*

While it is important that we first learn to add and multiply matrices by hand to understand how these processes work, just like with row reduction it is reasonable to expect that we'll often use available technology to perform tedious computations like multiplying a 4×5 and 5×7 matrix. Moreover, in real world applications, it is not uncommon to have to deal with matrices that have thousands of rows and columns, or more. Here we introduce a few *Mathematica* commands that are useful in performing some of the algebraic manipulations we have studied in this section.

Let's consider some of the matrices defined in earlier examples[1]:

$$\mathbf{A} = \begin{bmatrix} 1 & 3 & -4 \\ 0 & -7 & 2 \end{bmatrix}, \mathbf{B} = \begin{bmatrix} -6 & 10 \\ 3 & 2 \end{bmatrix}, \mathbf{M} = \begin{bmatrix} -6 & 10 & -1 \\ 3 & 2 & 11 \end{bmatrix}$$

After defining each of these three matrices with the usual commands in *Mathematica*, such as

```
A = {{1, 3, -4},{0, -7, 2}}
```

we can execute the sum of $\mathbf{A}$ and $\mathbf{M}$ and the scalar multiple $-3\mathbf{B}$ and see the results in matrix form with the commands

```
MatrixForm[A + M]
MatrixForm[-3*B]
```

for which *Mathematica* will report the outputs

$$\begin{pmatrix} -5 & 13 & -5 \\ 3 & -5 & 13 \end{pmatrix} \text{ and } \begin{pmatrix} 18 & -30 \\ -9 & -6 \end{pmatrix}$$

We have previously seen that to compute a matrix-vector product, the "period" is used to indicate multiplication, as in `A.x;`. The same syntax holds for matrix multiplication, where defined. For example, if we wish to compute the product $\mathbf{BA}$, we enter

```
MatrixForm[B.A]
```

which yields the output

$$\begin{pmatrix} -6 & -88 & 44 \\ 3 & -5 & -8 \end{pmatrix}$$

If we try to have *Mathematica* compute an undefined product, such as $\mathbf{AB}$ through the command `MatrixForm[A.B]`, we get the error message

```
Dot::dotsh:  Tensors {{1,3,-4},{0,-7,2}} and {{-6,10},{3,2}} have incompatible
shapes.
```

In the event that we need to execute computations involving an identity matrix, rather than tediously enter all the 1's and 0's, we can use the built-in *Mathematica* command `IdentityMatrix[n];` where n is the number of rows and columns in the matrix. For example, entering

```
Id := IdentityMatrix[4];
```

[1] We use $\mathbf{M}$ rather than $\mathbf{C}$ because "$\mathbf{C}$" is a protected symbol in *Mathematica*.

results in the output

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Note: "Id" is the name we are using to store this identity matrix. We do not use the letter "I" because in *Mathematica* "i" is reserved to represent $\sqrt{-1}$.

Finally, if we desire to compute the transpose of a matrix $\mathbf{A}$, such as

$$\mathbf{A} = \begin{bmatrix} 1 & 3 & -4 \\ 0 & -7 & 2 \end{bmatrix}$$

the relevant command is

```
Transpose[A] // MatrixForm
```

which generates the output

$$\begin{pmatrix} 1 & 0 \\ 3 & -7 \\ -4 & 2 \end{pmatrix}$$

1.8.2 Matrix Inverses using *Mathematica*

Certainly we can use *Mathematica*'s row reduction commands to find inverses of matrices. However, an even simpler command exists that enables us to avoid having to enter the corresponding identity matrix. Let us consider the two matrices from Examples 1.8.2 and 1.8.3. Let

$$\mathbf{A} = \begin{bmatrix} 2 & 1 & -2 \\ 1 & 1 & -1 \\ -2 & -1 & 3 \end{bmatrix}$$

be defined in *Mathematica*. If we enter the command

```
Inverse[A] // MatrixForm
```

we see the resulting output which is indeed $\mathbf{A}^{-1}$,

$$\begin{pmatrix} 2 & -1 & 1 \\ -1 & 2 & 0 \\ 1 & 0 & 1 \end{pmatrix}$$

For the matrix

$$\mathbf{A} = \begin{pmatrix} 2 & 1 \\ -6 & -3 \end{pmatrix}$$

executing the command `Inverse[A]` produces the output

```
Inverse::sing:  Matrix {{2,1},{-6,-3}} is singular.
```

which is *Mathematica's* way of saying "$\mathbf{A}$ is not invertible."

1.9.1 Determinants using *Mathematica*

Obviously for most square matrices of size greater than 3×3, the computations necessary to find determinants are tedious and present potential for error. As with other concepts that require large numbers of arithmetic operations, *Mathematica* offers a single command that enables us to take advantage of the program's computational powers. Given a square matrix $\mathbf{A}$ of any size, we simply enter

```
Det[A]
```

As we explore properties of determinants in the exercises of this section, it will prove useful to be able to generate random matrices. In *Mathematica*, one accomplishes this for a 3×3 matrix with integer entries between -99 and 99 by the command

```
A = Table[ Random[Integer, -99, 99], 3, 3]
```

From here, we can compute the determinant of this random matrix simply by entering

```
Det[A];
```

See Exercise 11 in Section 1.9 for a particular instance where this code will be useful.

1.10.2 Using *Mathematica* to Find Eigenvalues and Eigenvectors

Due to its reliance upon determinants and the solution of polynomial equations, the eigenvalue problem is computationally difficult for any case larger than 3×3. Sophisticated algorithms have been developed to compute eigenvalues and eigenvectors efficiently and accurately. One of these is the so-called QR algorithm, which through an iterative technique produces excellent approximations to eigenvalues and eigenvectors simultaneously.

While *Mathematica* implements these algorithms and can find both eigenvalues and eigenvectors, it is essential that we not only understand what the program is attempting to compute, but also how to interpret the resulting output.

Given an $n \times n$ matrix $\mathbf{A}$, we can compute the eigenvalues of $\mathbf{A}$ with the command

```
Eigenvalues[A]
```

Doing so for the matrix

$$\mathbf{A} = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}$$

from Example 1.10.2 yields the *Mathematica* output

$$\{3, 1\}$$

and thus the two eigenvalues of the matrix $\mathbf{A}$ are 3 and 1. If we desire the eigenvectors, we can use the command

```
Eigenvectors[A]
```

which leads to the output

$$\{\{1, 1\}, \{1, -1\}\}$$

which tells us the eigenvectors are $[1 \;\; 1]^T$ and $[-1 \;\; 1]^T$. It is likely best in these computations to use a command that links the eigenvectors explicitly to the corresponding eigenvalues. To accomplish this, we use the syntax

```
Eigensystem[A]
```

which results in the output

$$\{\{3, 1\}, \{\{1, 1\}, \{-1, 1\}\}\}$$

where we see the eigenvalues listed first, and then the corresponding eigenvectors listed in order.

Mathematica is extremely powerful. It is not at all bothered by complex numbers. So, if we enter a matrix like the one in Example 1.10.3 that has no real eigenvalues, *Mathematica* will find complex eigenvalues and eigenvectors. To see how this appears, we enter the matrix

$$\mathbf{R} = \begin{bmatrix} \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{bmatrix}$$

and execute the command

```
Eigensystem(R)
```

The resulting output is

$$\{\{\frac{1+i}{\sqrt{2}}, \frac{1-i}{\sqrt{2}}\}, \{\{i, 1\}, \{-i, 1\}\}\}$$

Note that here *Mathematica* is using "i" to denote $\sqrt{-1}$. Just as we saw in Example 1.10.3, $\mathbf{R}$ does not have any real eigenvalues. We can use familiar properties of complex numbers (most importantly, $i^2 = 1$) to actually check that the equation $\mathbf{Ax} = \lambda\mathbf{x}$ holds for the listed complex eigenvalues and complex eigenvectors above. However, at this point in our study, these complex eigenvectors are of less importance, so we defer further details on them until later work with systems of differential equations.

One final example is relevant here to see how *Mathematica* deals with repeated eigenvalues and "missing" eigenvectors. If we enter the 3×3 matrix $\mathbf{A}$ from Example 1.10.4 and then execute the `Eigensystem` command, we receive the output

$$\{\{-4, 3, 3\}, \{\{-6, 8, 3\}, \{5, -2, 1\}, \{0, 0, 0\}\}\}$$

Here we see that 3 is a repeated eigenvalue of $\mathbf{A}$ with multiplicity 2. The first vector in the output matrix is the eigenvector corresponding to $\lambda = -4$. The final vector of all zeros indicates that $\mathbf{A}$ has only one linearly independent eigenvector corresponding to the eigenvalue $\lambda = 3$. This vector of all zeros also demonstrates that $\mathbb{R}^3$ does not have a linearly independent spanning set that consists of eigenvectors of $\mathbf{A}$.

2.2.1 Plotting Slope Fields using *Mathematica*

Mathematica does not have a single command that we can use to plot the slope field for a differential equation; however, with a bit of effort, we can manipulate its ability to plot a general vector field to generate the visual representation of a slope field that we week. It is first necessary to load the `VectorFieldPlots` package with the command

```
Needs["VectorFieldPlots`"]
```

To plot a vector field, we need a function of two variables x and y with two components u and v that will generate the direction of a vector in the field at any point. Said differently, given a point (x, y), we want to generate a vector $[u \;\; v]^T$ to plot emanating from the point (x, y). This is precisely what the differential equation

$$\frac{dy}{dx} = f(x, y)$$

accomplishes. With $f(x, y)$ being the slope of the tangent line at point (x, y), we can realize a vector at that point by considering the vector $[1 \;\; f(x, y)]^T$, whose slope will be exactly $f(x, y)$.

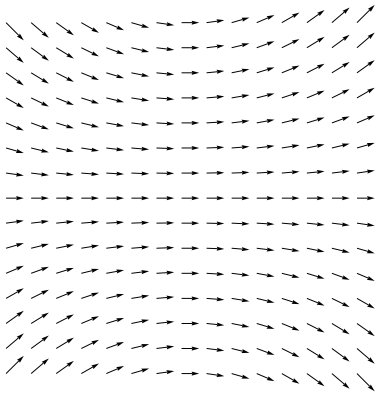
Thus, to plot the direction field associated with a given differential equation, say $\frac{dy}{dx} = xy$, on the window $[-1, 1] \times [-1, 1]$, first introduce variables x and y with

```
var('t')
```

and then use the syntax

```
VectorFieldPlot[{1, x*y},{x, -1, 1},{y, -1, 1}]
```

which generates the output.



It is often ideal to have all vectors in the slope field of the same length. If we normalize the vector field by making the function that generates it one whose output is a unit vector, *Mathematica* will take care of the rest. We do this in the command below, plus name the plot "`SlopeField`" for use in a moment:

```
SlopeField = VectorFieldPlot[1/Norm[{1, x*y}] * {1, x*y}, {x, -1, 1},{y,
-1, 1}]
```

Also, it is often helpful to have axes included. To do so, we plot (and store) axes with the command

```
axes = Plot[ 0, {x, -1, 1}, PlotRange -> {-1,1}, AxesLabel -> {x,y}]
```

Finally, we can show the slope field with the axes present via the `Show` command as follows:

```
Show[axes, SlopeField]
```

which results in the output shown next.

Without a well-chosen window selected by the user, the plot *Mathematica* generates may not be very insightful. For example, if the above command were changed so that the range of y values is `-10 .. 10`, almost no information can be gained from the slope field. As such, we will strive to learn to analyze the expected behavior of a differential equation from its form so that we can choose windows well in related plots; we may often have to experiment and explore to find graphs that are useful.

Finally, if we are interested in one or more related initial-value problems, some additional effort enables us to sketch the graph of each corresponding solution. Say we are interested in the IVP solution that satisfies the given DE along with the initial condition $y(0) = -0.5$. We may use *Mathematica* to solve the IVP and then plot this result using the following syntax:

```
soln = DSolve[ {y'[x] == x*y[x], y[0] == -0.5}, y[x], x];
solnPlot = Plot[y[x] /.  soln[[1]], {x, -1, 1}, PlotStyle -> Thick ];
show[axes, solnPlot, SlopeField]
```

This results in the image shown below that demonstrates the slope field, the coordinate axes, and the curve that represents the solution to the IVP.

It is often easier for the user to simply plot an IVP's solution by hand, just by using the initial condition and following the "directions" provided by the slope field.

3.4.1 Plotting Direction Fields for Systems using *Mathematica*

As we noted in section 2.2.1, *Mathematica* does not have a single command that we can use to plot the slope field for a differential equation; however, with a bit of effort, we can manipulate its ability to plot a general vector field to generate the visual representation of a slope field that we week. It is first necessary to load the `VectorFieldPlots` package with the command

```
Needs["VectorFieldPlots`"]
```

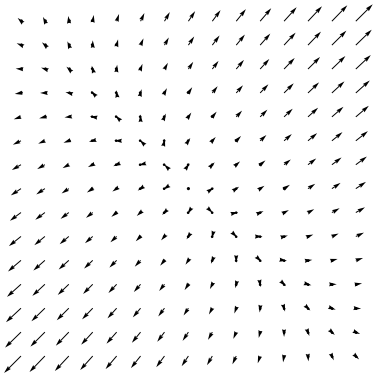
To plot a vector field, we need a function $f(x, y)$ with two components u and v that will generate the direction of a vector in the field at any point. Said differently, given a point (x, y), we want to generate a vector $[u \;\; v]^T$ to plot emanating from the point (x, y).

In the context of plotting the direction field for the system of DEs given by $\mathbf{x}' = \mathbf{A}\mathbf{x}$, the vector we wish to plot at (x, y) is $\mathbf{x}'$. Viewing $\mathbf{x} = [x(t) \;\; y(t)]^T$, it follows $\mathbf{x}' = [x'(t) \;\; y'(t)]^T$. Thus, the vector field we desire is given by $u = x'$ and $v = y'$. This precisely is the vector given by taking the product $\mathbf{A}\mathbf{x}$.

Therefore, to plot the direction field associated with the system of DEs $\mathbf{x}' = \mathbf{A}\mathbf{x}$ given by the matrix $\mathbf{A} = \begin{bmatrix} 3 & 2 \\ 2 & 3 \end{bmatrix}$ from Example 3.4.1, we use the following syntax:

```
A = {{3, 2}, {2, 3}}
VectorFieldPlot[A.{x, y}, {x, -4, 4}, {y, -4, 4}]
```

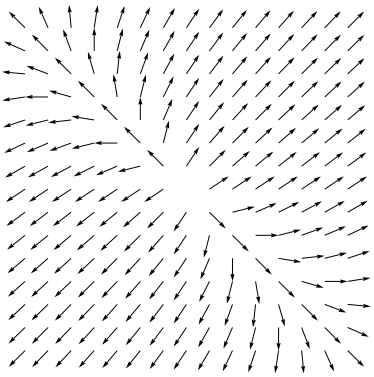
This command produces the output shown below.



Because *Mathematica* does not automatically scale the vectors in the vector field to be of consistent length, it is more helpful to make each vector in the field of the same length. This is accomplished by the command

```
VectorFieldPlot[1/Norm[A.{x,y}] * A.{x, y}, {x, -4, 4}, {y, -4, 4}]
```

which produces the output



Mathematica does object slightly to this command, since at the origin, the zero vector is generated; since the length of the zero vector is zero, in the command above when we divide by `Norm[A.{x,y}]`, we are technically dividing by zero. The program reports this, but generates the resulting plot anyway.

From here, it is a straightforward exercise to sketch trajectories by hand; while one can also generate these trajectories in *Mathematica*, doing so is relatively complicated. The user is instead encouraged to think about how the direction field aligns with the eigenvalues and eigenvectors of the matrix of the system, to plot the straight-line solutions by hand, and then think about how the nonlinear trajectories should appear, especially as $t \to \infty$. As shown in section 2.2.1, it is also possible to superimpose the coordinate axes on the plot, and the directions stated in 2.2.1 apply in the exact same way in this context.

As always, the user can experiment some with the window in which the plot is displayed: the range of x- and y-values will affect how clearly the direction field is revealed.

3.7.1 Applying Variation of Parameters Using *Mathematica*

Here we address how *Mathematica* can be used to execute the computations in a problem such as the one posed in Example 3.7.2, where we are interested in solving the nonhomogeneous linear system of equations given by

$$\mathbf{x}' = \begin{bmatrix} 2 & -1 \\ 3 & -2 \end{bmatrix} \mathbf{x} + \begin{bmatrix} 1/(e^t+1) \\ 1 \end{bmatrix}$$

Because we already know how to find the complementary solution, we focus on determining $\mathbf{x}_p$ by variation of parameters. First, we use the complementary solution,

$$\mathbf{x}_h = c_1 e^{-t} \begin{bmatrix} 1 \\ 3 \end{bmatrix} + c_2 e^{t} \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

to define the fundamental matrix $\mathbf{\Phi}(t)$. To define and view this matrix in *Mathematica*, we enter

```
Phi := {{Exp[-t], Exp[t]},{3 Exp[-t], Exp[t]}};
MatrixForm[Phi]
```

We next use the `Inverse` command to find $\mathbf{\Phi}^{-1}$ by entering

```
PhiInv := Simplify[Inverse[Phi]];
MatrixForm[PhiInv]
```

The `Simplify` command will be used repeatedly in this context to help ensure that the program doesn't unnecessarily complicate the algebraic expressions. At this stage, the resulting output ($\mathbf{\Phi}^{-1}$) is

$$\begin{pmatrix} -\frac{e^t}{2} & \frac{e^t}{2} \\ \frac{3e^{-t}}{2} & -\frac{e^{-t}}{2} \end{pmatrix}$$

Next, in order to compute $\mathbf{\Phi}^{-1}(t)\mathbf{b}(t)$, we must enter the function $\mathbf{b}(t)$. We enter

```
b := {1/(Exp[t] + 1), 1}
```

and then

```
y := Simplify[PhiInv.b];
MatrixForm[y]
```

At this point, y is a 2×1 array that holds the vector function $\mathbf{\Phi}^{-1}(t)\mathbf{b}(t)$. Specifically, the output for y displayed by *Mathematica* is

$$\begin{pmatrix} \frac{e^{2t}}{2+2e^t} \\ -\frac{1-2e^{-t}}{2+2e^t} \end{pmatrix}$$

To access the components in y, we reference them with the commands `y[[1]]` and `y[[2]]`. In particular, since we have to integrate $\mathbf{\Phi}^{-1}(t)\mathbf{b}(t)$ component-wise, we enter

```
Y := { Integrate[y[[1]], t], Integrate[y[[2]], t]};
MatrixForm[Y]
```

These last commands produce the output

$$\begin{pmatrix} \frac{e^t}{2} - \frac{1}{2}\ln(1+e^t) \\ -e^{-t} + \frac{3}{2}\ln(1+e^{-t}) \end{pmatrix}$$

and obviously stores $\mathbf{\Phi}^{-1}(t)\mathbf{b}(t)$ in $\mathbf{Y}$. We add that we write "ln" where *Mathematica* writes "Log." The second component is slightly different, but algebraically equal, to what *Maple* reports, and thus what is presented here (and below) appears different from what is reported in Section 3.7 of the textbook. The reader can verify that the functions present, however, are in fact the same.

Finally, in order to compute $\mathbf{\Phi}(t)\int \mathbf{\Phi}^{-1}(t)\mathbf{b}(t)\,dt$, we need to enter `Phi.Y`. Of course, we again want to simplify, so we use

```
Simplify[Phi.Y];
MatrixForm[%]
```

which produces the output

$$\begin{pmatrix} -\frac{1}{2} - \frac{1}{2}(-1 + 3e^t\ln(1+e^{-t}) - e^{-t}\ln(1+e^t)) \\ \frac{1}{2}(1 + 3e^t\ln(1+e^{-t}) - 3e^{-t}\ln(1+e^t)) \end{pmatrix}$$

This last result is an equivalent version of the particular solution $\mathbf{x}_p$ to the original system of nonhomogeneous equations given in Example 3.7.2.

4.6.1 Solving Characteristic Equations using *Mathematica*

While solving linear differential equations of order n requires nearly identical methods to DEs of order 2, there is one added challenge from the outset: solving the characteristic equation. The characteristic equation is a polynomial equation of degree n; while every such equation of degree 2 can be solved using the quadratic formula, equations of higher order can be much more difficult, and (for equations of degree 5 and higher) often impossible, to solve by algebraic means.

Computer algebra systems like *Mathematica* provide useful assistance in this matter with commands for solving equations exactly and approximately. For example, say we have the characteristic equation

$$r^4 - r^3 - 7r^2 + r + 6 = 0$$

To solve this exactly in *Mathematica*, we enter

```
Solve[r^4 - r^3 - 7 r^2 + r + 6 == 0, r]
```

Mathematica produces the output

$$\{\{r \to -2\}, \{r \to -1\}, \{r \to 1\}, \{r \to 3\}\}$$

showing that these are the four roots of the characteristic equation.

Of course, not all polynomial equations will have all integer solutions, much less all real solutions. For example, if we consider the equation

$$r^4 + r^3 + r^2 + r + 1 = 0$$

and use the `Solve` command, we see that

```
Solve[r^4 + r^3 + r^2 + r + 1 == 0, r]
```

results in the output

$$\{\{r \to -(-1)^{1/5}\}, \{r \to (-1)^{2/5}\}, \{r \to -(-1)^{3/5}\}, \{r \to (-1)^{4/5}\}\}$$

which is not very helpful.

In this case, rather than having exact results in terms of roots of negative numbers, we might prefer a decimal approximation to the zeros of the equation. One way to achieve this is to use the `NSolve` command:

```
NSolve[r^4 + r^3 + r^2 + r + 1 == 0, r]
```

which generates the result

$$\{\{r \to -0.809017 - 0.587785i\}, \{r \to -0.809017 + 0.587785i\}, \{r \to 0.309017 - 0.951057i\},$$

$$\{r \to 0.309017 + 0.951057i\}\}$$

For polynomial equations of degree 5 or more, the `NSolve` command is always the appropriate tool to use to determine accurate approximations of the equation's solutions.

5.4.3 The Heaviside and Dirac Functions in *Mathematica*

Both the Heaviside and Dirac functions belong to *Mathematica*'s library of basic functions. The syntax for the Heaviside function is `UnitStep[t]` (or equivalently, `HeavisideTheta[t]`). Similarly the Dirac function is given by `DiracDelta[t]`.

For work with the Heaviside function, we often denote the function by $u(t)$. In *Mathematica*, this can be accomplished with the command

```
u[t_] := UnitStep[t]
```

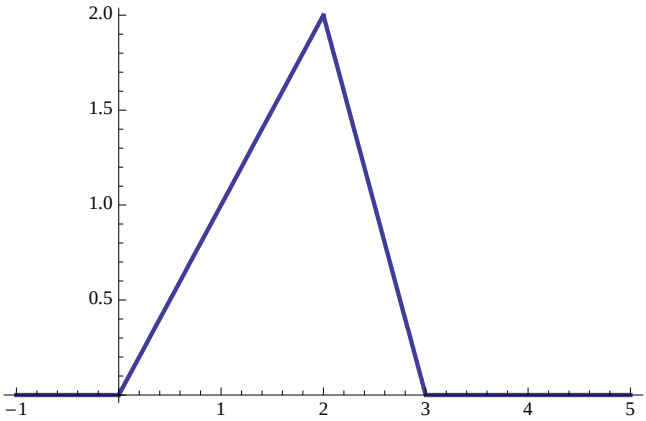
Then, to enter and plot a piecewise-defined function such as

$$f(t) = t(u(t) - u(t-2)) + (6-2t)(u(t-2) - u(t-3))$$

we may use the syntax

```
> f[t_] := t(u[t]-u[t-2]) + (6-2t)(u[t-2]-u[t-3])
> Plot[f[t], {t, -1, 5}, PlotStyle -> Thick]
```

to generate the plot shown below.



More on both the Heaviside function and the Dirac function in *Mathematica*, particularly related to their roles in solving initial-value problems with Laplace transforms, can be found in Section 5.6.1.

5.6.1 Laplace Transforms and Inverse Transforms using *Mathematica*

As we have noted, while we have computed Laplace transforms for a range of functions, there are many more examples we have not considered. Moreover, even for familiar functions, certain combinations of them can lead to tedious, involved calculations. Computer algebra systems such as *Mathematica* are fully capable of computing Laplace transforms of functions, as well as inverse transforms. Here we demonstrate the syntax required in the solution of the initial-value problem from Example 5.5.4:

$$y'' + 4y' + 13y = 2u(t-\pi)\sin 3t, \quad y(0) = 1,\ y'(0) = 0 \tag{1}$$

If, for example, we desire to use *Mathematica* to compute the Laplace transform of $2u(t-\pi)\sin 3t$, we use the syntax

```
LaplaceTransform[2 UnitStep[t-Pi] Sin[3t], t, s]
```

Note particularly that the "s" in the sine function must be capitalized, as is standard for any basic library function in *Mathematica*. The above command results in the output

$$-\frac{6e^{-s\pi}}{s^2+9}$$

which is precisely the transform we expect.

After computing by hand the transform of the left-hand side of (1) and solving for $Y(s)$, as shown in detail in Example 5.5.4, we have

$$Y(s) = \frac{s+4}{s^2+4s+13} - 2e^{-\pi s}\frac{3}{(s^2+9)(s^2+4s+13)}$$

Here we may use *Mathematica*'s `InverseLaplaceTransform` command to determine $\mathcal{L}^{-1}[Y(s)]$. While we could choose to do so all at once, for simplicity of display we do so in two steps. First,

```
y1 = InverseLaplaceTransform[(s+4)/(s^2 + 4s + 13), s, t]
```

results in the output

$$\frac{1}{6}e^{(-2-3i)t}((3+2i) + (3-2i)e^{6it}) \tag{2}$$

which is somewhat surprising, for we expect a real, not a complex, solution to the differential equation. It turns out that *Mathematica* is disguising the real solution here. Using the `ComplexExpand` and `Simplify` commands as follows,

```
ComplexExpand[y1]
Simplify[%]
```

results in the more expected output of

$$\frac{1}{3}e^{-2t}(3\cos 3t + 2\sin 3t)$$

Similarly, for the second term in $Y(s)$, we compute

```
y2 = InverseLaplaceTransform[2 exp(-Pi s) 3/((s^2 + 9) (s^2 + 4 s +
13)), s, t]
```

Here, *Mathematica* produces the complex output

$$\text{HeavisideTheta}[-\pi+t]\left(\left(\frac{1}{240}-\frac{i}{240}\right)e^{(-2-3i)(-\pi+t)}\left((1+2i)+(2+i)e^{6i(-\pi+t)}\right)\right) \tag{3}$$

$$+\,\text{HeavisideTheta}[-\pi+t]\left(\frac{1}{120}(-3\cos[3(-\pi+t)]+\sin[3(-\pi+t)])\right)$$

From here, if we again perform the `ComplexExpand` and `Simplify` commands, we find that `y2` is the function

$$-\frac{1}{20}e^{-2t}\text{HeavisideTheta}[-\pi+t](3(e^{2\pi}-e^{2t})\cos[3t]+(e^{2\pi}+e^{2t})\sin[3t])$$

which (after a bit more algebraic rearrangement) corresponds to our work in Example 5.5.4. The sum of the two functions of t that have resulted from inverse transforms in (2) and (3) is precisely the solution to the IVP.

Note that in computing the inverse transform (3), *Mathematica* has implicitly executed the partial fraction decomposition of the expression

$$\frac{3}{(s^2+9)(s^2+4s+13)}$$

If we wish to find this explicitly, we can use the command

```
Apart[3/((s^2 + 9)*(s^2 + 4s + 13))]
```

which produces the output

$$-\frac{3(-1+s)}{40(9+s^2)}+\frac{3(3+s)}{40(13+4s+s^2)}$$

In general, we see that to compute the Laplace transform of $f(t)$ in *Mathematica* we use the syntax

```
LaplaceTransform[f(t), t, s]
```

whereas to compute the inverse transform of $F(s)$, we enter

```
InverseLaplaceTransform[F(s), s, t]
```

6.2.1 Plotting Direction Fields of Nonlinear Systems using *Mathematica*

The *Mathematica* syntax used to generate the plots in this section is essentially identical to that discussed for direction fields for linear systems in section 3.4.1.

Consider the system of differential equations from Example 6.2.1 given by

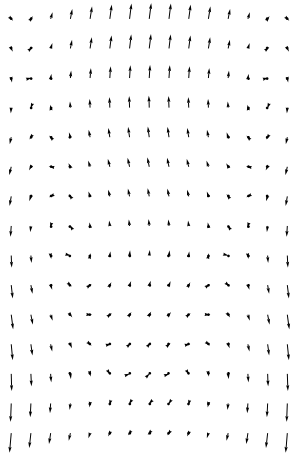
$$\begin{aligned} x' &= \sin(y) \\ y' &= y - x^2 \end{aligned}$$

We use x and y in place of x_1 and x_2 to simplify the syntax in *Mathematica*.

As in 3.4.1, we use the `VectorFieldPlot` command to plot the vector $\{x', y'\}$, and thus use the syntax

```
Needs["VectorFieldPlots`"]
VectorFieldPlot[{Sin[y], y - x^2}, {x, -3, 3}, {y, -1, 8}]
```

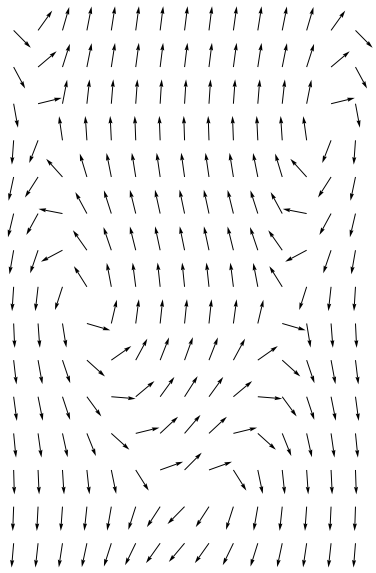
which produces the somewhat difficult-to-interpret output



We thus again scale the vectors being plotted to be of consistent length, using the syntax

```
VectorFieldPlot[1/Norm[{Sin[y], y - x^2}] * {Sin[y], y - x^2}, {x, -3,
3}, {y, -1, 8}]
```

which results in the more helpful plot



As we can see, while the plot provides some useful information, it principally shows that there is something interesting happening at some points along the parabola $y = x^2$, specifically at the equilibrium points discussed in Example 6.2.1. Further analysis to accompany the work of the computer is merited; some possibilities for this exploration are considered in Sections 6.3 and 6.4 of the text, after which it will be easier to plot trajectories by hand.

It is worth noting that while *Mathematica* has many excellent traits and capabilities, some other computer algebra systems are far easier to work with for plotting meaningful direction fields and trajectories. For example, see the discussion in the text regarding *Maple*.