Notes for the Maple Computer Algebra System

GETTING STARTED WITH MAPLE

Using Maple with *Linear Algebra and Its Applications*

The Maple commands for most linear algebra operations, including the manipulation of vectors and matrices, can be found in the `linalg` package that comes with Maple. The `laylinalg` package contains Maple implementations of the special MATLAB commands presented in the text, data for many problems in the text, and data for the Maple Projects. The current version of the `laylinalg` package can be used in either Maple 7 or Maple 8 and can be downloaded from `http://www.laylinalgebra.com/`.

To make the `laylinalg` package accessible within Maple it is necessary to expand the downloaded archive and to instruct Maple where to find these files. First, unzip (or unstuff if you are using a Macintosh computer) the archive containing the `laylinalg` package into a new folder named `laylinalg`. When this has been successfully completed, the `laylinalg` folder will contain three files: `maple.hdb`, `maple.ind`, and `maple.lib`. (The fourth file in the archive, `MapleReadMe.txt`, contains a more detailed version of the information in this paragraph.) To instruct Maple where to find the `laylinalg` package, the full pathname of this new directory must be added to the list of directories in `libname`. For example, if the full pathname is `C:\\laylinalg` then the command

```
> libname := "C:\\laylinalg", libname:
```

must be executed. To ensure these steps are performed every time you use Maple, this command should be placed in a Maple initialization file. The `MapleReadMe.txt` file contains additional information about Maple initialization files; see also the Maple help page produced by the command `?mapleinit`. If you experience any difficulties with this installation process, please check with your instructor for specific instructions for your site.

To load a Maple "package" into a Maple session, use the `with` command. The command

```
> with( laylinalg );
```

loads the `laylinalg` package and displays all of the commands in the package. This particular command also loads Maple's own `linalg` package and modifies the display of Maple vectors so they appear as $n \times 1$ column matrices.[1]

Each Maple command must end with a semicolon or colon. The semicolon tells Maple to display the result of the command; a colon suppresses the output. Maple is case-sensitive but ignores all whitespaces (blanks) around numbers, names, keywords, and operators in com-

[1] To have vectors displayed horizontally, execute the command `pvac := false;`. The command `pvac` stands for print vectors as columns.

mands. For example, Maple will not recognize `with( LayLinAlg );` but will understand `with(laylinalg);`. The commands in these notes include extra spaces to increase readability.

Entering Matrices in Maple

There are many ways to create matrices in Maple. In general, a matrix is created with the `matrix` command. The first two arguments of this command are the number of rows and columns in the matrix, respectively. If there are only two arguments, then the elements of the matrix are unassigned (see matrix A, below). The optional third argument is used to specify the entries of the matrix. If the third argument is a list or vector, these quantities are used to fill the matrix row-by-row from left-to-right (see matrices B and C). The third argument can also be a Maple function of two variables. In this case each element of the matrix is assigned value by evaluating the function with the corresponding row and column indices (see matrix E). Diagonal matrices are easily defined using the `diag` command from the `linalg` package (see matrix F).

```
> A := matrix( 2, 3 );
```

$$A := \text{array}(\ 1\ ..\ 2,\ 1\ ..\ 3,\ [\ ])$$

```
> evalm( A );
```

$$\begin{bmatrix} A_{1,1} & A_{1,2} & A_{1,3} \\ A_{2,1} & A_{2,2} & A_{2,3} \end{bmatrix}$$

```
> B := matrix( 2, 3, [1,2,3,4,5,6] );
```

$$B := \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

```
> C := matrix( 2, 3, [1,2,3,4] );
```

$$C := \begin{bmatrix} 1 & 2 & 3 \\ 4 & C_{2,2} & C_{2,3} \end{bmatrix}$$

```
> E := matrix( 2, 3, (i,j) -> 1/(i+j) );
```

$$E := \begin{bmatrix} \frac{1}{2} & \frac{1}{3} & \frac{1}{4} \\ \frac{1}{3} & \frac{1}{4} & \frac{1}{5} \end{bmatrix}$$

```
> F := diag( 1, 2, 3 );
```

$$F := \begin{bmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 3 \end{bmatrix}$$

The `entermatrix` command, from the `linalg` package, provides a somewhat more user-friendly interface for entering the entries of a matrix:

```
> G := matrix(2,3): # define the matrix G

> entermatrix(G);
enter element 1,1 > 1; # note that the semicolon is required
enter element 1,2 > 2;
enter element 1,3 > 3;
enter element 2,1 > 4;
enter element 2,2 > 5;
enter element 2,3 > 6;
```

A more graphical but seldom-used method for defining a matrix with between two and four rows and two and four columns is to use the Matrix Palette. You are strongly advised to not use the Matrix and Vector Palettes with the `laylinalg` package. The commands produced by the Matrix and Vector Palettes create Maple objects with type `Matrix` and `Vector`. These objects are designed for use with Maple's `LinearAlgebra` package, not the type `matrix` and `vector` objects used by the `linalg` and `laylinalg` packages.

Using Maple's Online Help

These notes are intended to be self-contained. If, however, you are interested in additional information, you should first consult Maple's online help. The command `help( laylinalg );` or, equivalently, `?laylinalg` accesses the help worksheet with an overview the `laylinalg` package. Help worksheets exist for all Maple commands, including all commands in the `linalg` and `laylinalg` packages. For example, the command `?gauss` accesses the help worksheet for the `gauss` command in the `laylinalg` package.

The `Help` menu on the Maple user interface can also be used to search for help on a specific keyword. A `Full Text Search` lists all help worksheets containing keywords that you specify. A `Topic Search` interactively shows all help worksheets whose topic matches the search string. For example, there is a long list of help topics that begin with the string `la`. When the search string is expanded to `lay` the list of matching help topics reduces to all of the help worksheets for the `laylinalg` package.

STUDY GUIDE NOTES

At the beginning of each Maple session for this linear algebra course, enter the command:
`> with( laylinalg );`
Notice the use of a colon to suppress the output.

SECTION 1.1 Row Operations

The data for the exercises in this section are contained in the `laylinalg` package. The list of exercises within chapter 1 section 1 for which Maple data is available is displayed by executing the command: `c1s1();`. If Maple simply repeats your input, it is likely that the `with( laylinalg );` command has not been executed.

The data for a specific problem, such as Exercise 13 in Section 1.1, can be loaded with the command: `c1s1( 13 );` or `c1s1(13);`. (The spaces around `13` are optional.) The output from this command displays each assignment that Maple has made. In this section the data for each exercise are stored in a matrix M. To see the current contents of the matrix M, execute the command: `evalm( M );`.

Row operations on M can be performed using commands from either the `laylinalg` or the `linalg` package. Table 1 summarize the commands for the three elementary row operations.

`laylinalg` Command Syntax	Description
`replace( M, r, m, s );`	row $r \leftarrow$ row $r + m$ * row s
`swap( M, r, s );`	row $r \leftrightarrow$ row s
`scale( M, r, c );`	row $r \leftarrow c$ * row r of M

`linalg` Command Syntax	Description
`addrow( M, s, r, m );`	row $r \leftarrow$ row $r + m$ * row s
`swaprow( M, r, s );`	row $r \leftrightarrow$ row s
`mulrow( M, r, c );`	row $r \leftarrow c$ * row r of M

Table 1: Comparison of elementary row operations in the `laylinalg` and `linalg` packages.

Notes:

- The syntax and functionality of the commands in the `linalg` and `laylinalg` packages are almost identical. The names in the `linalg` package are used throughout Maple; those used in `laylinalg` more closely match those used elsewhere in the textbook. The `laylinalg` commands will be used throughout this appendix.
- The name of any matrix in your current Maple session can be inserted in place of `M` in the commands above.

- The letters r and s are positive integers corresponding to row numbers. The names m and c are scalar expressions (often numbers, but sometimes symbolic expressions) that you choose.

If you enter one of the row operation commands, say,

```
> swap( M, 1, 3 );
```

then the new matrix produced from `M` is not given a name. You can refer to this result using `%`, the history operator. If, instead, you type

```
> M1 := swap( M, 1, 3 );
```

then the answer is stored in the matrix `M1`. If the next operation is

```
> M2 := replace( M1, 2, 5, 1 );
```

then the result after performing this row operation on `M1` is placed in `M2`, and so on. Note the use of `:=` in *assignments*; the symbol `=` is used to create an *equation*.

One advantage of giving a new name to each new matrix is that you can easily go back a step if you do not like what you just did to a matrix. If you type

```
> M := replace( M, 2, 5, 1 );
```

then the result is placed back in `M` and the "old" `M` is lost. Of course, the "reverse" operation

```
> M := replace( M, 2, -5, 1 );
```

will recreate the original matrix `M`.

Notes:

- For the problems in this section and the next, the multiple m needed in the `replace` or `addrow` command will usually be a small integer or fraction that you can compute in your head. The next two paragraphs describe how to use Maple in situations where m is not easily computed mentally.

  The entry in row r and column c of a Maple matrix `M` is denoted by `M[r,c]`. If the number stored in `M[r,c]` is a floating-point number, that is, it is displayed with a decimal point, then the number may be accurate to only about eight (8) digits. It is generally more accurate — and simpler — to use `M[r,c]` instead of re-typing the displayed values in computations.

- For instance, to use the entry `M[s,c]` to change `M[r,c]` to 0, enter the commands:
  `> m := -M[r,c] / M[s,c];` # mult of row s to add to row r
  `> M1 := replace( M, r, m, s );` # adds m * row s to row r
  Or, you could use the single command:
  `> M1 := replace( M, r, -M[r,c]/M[s,c], s );`

- The environment variable `Digits` controls the number of significant digits Maple uses in floating-point computations. The default is 10. While this should be more than sufficient for the problems in this text, you can change this by assigning a new value to `Digits`. In general, you should expect a floating-point computation to be accurate to approximately `Digits`-2 digits.

SECTION 1.1 Symbolic Row Operations

A powerful feature of Maple is its ability to work with matrices and vectors that involve unspecified parameters. For example, Exercise 25 in Section 1.1 can be solved using Maple and the `laylinalg` package in the following steps

```
> c1s1( 25 );
> evalm( M );                      # display the augmented matrix
> M1 := replace( M, 3, 2, 1 );     # add 2 × row 1 to row 3
> M2 := replace( M1, 3, 1, 2 );    # add row 2 to row 3
> M2[3,4] = 0;                     # eqn to make system consistent
```

Symbolic multipliers can be used in `replace` and `scale`, for example: `replace( A, 3, a, 1 );`, but all row numbers must be explicit integers.

SECTION 1.3 Constructing a Matrix

To see the exercises with data for Section 1.3, execute the command:

```
> c1s3( );
```

The data for Exercise 25, for example, consists of the matrix A, the right-hand side vector $\mathbf{b}$, and the columns of the matrix, $\mathbf{a}_1$, $\mathbf{a}_2$, $\mathbf{a}_3$. The corresponding Maple names, `A`, `b`, `a1`, `a2`, and `a3`, can be assigned by executing the command:

```
> c1s1( 25 );
```

The command

```
> M := augment( a1, a2, a3, b );
```

creates a matrix with the given vectors as its columns. The same matrix could also be created by

```
> M := augment( A, b );
```

Exercises 11–14, 24–28, and 31 can be solved using the `laylinalg` commands `replace`, `swap`, and (occasionally) `scale` (or the similar `linalg` commands: `addrow`, `swaprow`, and `mulrow`), described in the Maple Note for Section 1.1 on page MP-4.

SECTION 1.4 `gauss` and `bgauss`

To solve $A\mathbf{x} = \mathbf{b}$, row reduce the augmented matrix

```
> M := augment( A, b );
```

The command

```
> x := vector( [5,3,-7] );
```

creates a column vector, `x`, with entries 5, 3, and -7. Matrix-vector multiplication is

```
> evalm( A &* x );
```

The `laylinalg` package contains two commands, `gauss` and `bgauss`, that can be used to speed up row reduction of a matrix. Use the command `with( laylinalg );` to load the `laylinalg` package. A ZIP file containing the `laylinalg` package can be downloaded from the WWW.

To make row reduction of $M = [\ A \quad \mathbf{b}\ ]$ more efficient, the command `gauss( M, r );` chooses the leading entry in row r of M as a pivot and uses row replacements to create zeros in the pivot column below this pivot entry. The result can be assigned to a name, otherwise the only way to access the result is with the history operator (`%`). In either case, the result is displayed to the screen.

For the backward phase of row reduction, use `bgauss( M, r );` which selects the leading entry in row r of M as the pivot, and creates zeros in the column above the pivot. Use `scale` to create leading 1's in the pivot positions.

Note:

- Exercise 19 of this section can be solved using symbolic row operations as discussed in the Maple Note for Section 1.1.

SECTION 1.5 Zero Matrices

The command `vector( m, 0 );` creates the zero vector in R^m and `matrix( m, n, 0 );` creates an $m \times n$ matrix of zeros. To solve an equation $A\mathbf{x} = \mathbf{0}$, use the command

```
> M := augment( A, vector( m, 0 ) );
```

to augment A with the column vector containing m zeros, then use `gauss`, `swap`, `bgauss`, and `scale` to complete the row-reduction of M.

SECTION 1.6 Rational and Floating-Point Format

Chemical equation-balance problems are studied best using exact or symbolic arithmetic. This is because the balance variables must be whole numbers (with no round-off allowed). When the chemical equations have integer coefficients — as they do here — Maple will perform exact computations. That is, when `laylinalg` row operations are applied to an integer- or rational-valued matrix, the resulting matrix will have integer or rational entries. No floating-point approximations will be used. Once you find a rational solution of a chemical equation-balance problem, you can multiply the entries in the solution vector by a suitable integer to produce a solution that involves only whole numbers.

Notes:

- To convert the entries of a Maple matrix, M, to floating-point numbers, use either `evalf( evalm(M) );` or `map( convert, M, float );`.
- The command `map( convert, M, rational );` returns the matrix in which each entry of the matrix is replaced with an approximately equal rational number.

SECTION 1.10 Generating a Sequence

The data for Exercises 9–13 in Section 1.10 consists of the migration matrix, M, and the initial population vector $\mathbf{x}_0$. The following set of commands generates the first ten (10) terms in the sequence defined by $\mathbf{x}_{k+1} = M\mathbf{x}_k$:

```
> x||0 := evalm(x0);                    # display initial population
> for k from 0 to 9 do                  # execute loop 10 times
>   x||(k+1) := evalm( M &* x||k ); # compute x_{k+1} from x_k
> end do;
```

Note that the first command reassigns `x0` to itself. This command can be omitted if you do not wish to include the initial population in the output.

Numbers are entered in Maple without commas. Maple uses scientific notation for numbers larger than 1,000,000 (10^6) or smaller than 10^{-6}. In Maple, numbers in scientific notation can be entered as `4.0 * 10^5` or `2.46 * 10^(-3)`. Note that the parentheses around a negative exponent are required. When the mantissa is entered as a floating point number, the number is displayed (and computed) with `Digits` digits. The default setting of `Digits`, 10, should be more than sufficient for all computations in this text.

SECTION 2.1 Matrix Notation and Operations

Several methods for defining a matrix in Maple have been described previously in this appendix. Refer to these discussions for information about creating your own matrices.

Recall that, in Maple, the (i, j)-entry of A is `A[i,j]`. One or more columns or rows of A can be obtained with the `col` or `row` command. For example, `col(A,3)` is column 3 of A and `row(A,2)` is row 2 of A. To extract the first two columns of A, use `col(A,1..2)`. (The expression `a..b` represents the integers from a to b.) Be aware that both `row` and `col` return a sequence of one or more (column) vectors.

Multiple vectors can be assembled as the columns of a new matrix with `augment`, as described in Section 1.4. For example, `augment( col(A,1..2) );`. To create a matrix from the rows of an existing matrix, either take the transpose of the result from `augment` or use the `stackmatrix` command.

The command `submatrix( A, a..b, c..d );` returns a matrix whose entries come from rows a through b and columns c through d of A. The number of rows in A is given by `rowdim(A)` and the number of columns by `coldim(A)`.

Maple uses `+` and `-` to denote matrix addition and subtraction, respectively. Multiplication of a matrix or vector by a scalar can be obtained with the commutative multiplication operator `*`; the non-commutative operator `&*` must be used for matrix multiplication, including matrix–vector products. If A is square and k is an integer, `A^k` denotes the k^{th} power of A. The `evalm` command is needed to force Maple to display the result of commands that produce vectors or matrices.

The transpose of A is `transpose(A)`. The outer product of $\mathbf{u}$ and $\mathbf{v}$ can be obtained with the command `evalm( u &* transpose(v) );`. The command `innerprod(u,v);` gives the inner

product of **u** and **v**, and this scalar is displayed without using `evalm`. (If the entries of **u** and/or **v** are complex, then you probably want to use `dotprod` instead of `innerprod`; see also the Maple Note for Section 6.1.)

Maple commands for the construction of many special matrices are given below:

```
> M := matrix( 5, 6, 0 );      # a 5 × 6 matrix of zeros
> M := matrix( 3, 5, 1 );      # a 3 × 5 matrix of ones
> M := diag( 3, 5, 7, 2, 4 ); # a 5 × 5 diagonal matrix
> M := diag( 1$6 );            # the 6 × 6 identity matrix
> M := randomint( 6, 4 );      # a 6 × 4 matrix (-9 ≤ entries ≤ 9)
```

SECTION 2.2 The Identity Matrix and A^{-1}

When A is 5×5, the command `M := augment( A, diag( 1$5 ) );` creates the augmented matrix $[\ A \quad I\ ]$. Use `gauss`, `swap`, `bgauss`, and `scale` to reduce the augmented matrix to $[\ I \quad A^{-1}\ ]$, if possible. See the Maple Note for Section 1.4 on page MP-6.

There are other Maple commands that row reduce matrices, invert matrices, and solve systems $A\mathbf{x} = \mathbf{b}$. They will be introduced later, after you have studied the concepts and algorithms of this section.

SECTION 2.3 `inverse`, `cond`, and `hilbert`

Determining whether a matrix is invertible is not always a simple matter. A fast and fairly reliable method is to use the command `inverse( A );`, which computes the inverse of A. The error message `Error, (in inverse) singular matrix` is displayed if the matrix is singular.

In Exercises 41–44 of this section, the Maple command `cond( A );` computes the condition number of any matrix A using quantities called the singular values of A (discussed in Section 7.4). To perform the experiment described in Exercise 42 you can use the following Maple commands:

```
> x := randvector( 4 );              # random 4-d vector
> b := evalm( A &* x );              # compute exact RHS
> x1 := evalm( inverse(A) &* b );    # solution using A^-1
> evalm( x - x1 );                   # exact - computed
```

Because Maple uses exact arithmetic for integers, **x** and **x**1 should be the same. To force Maple to use floating-point computations, replace the definition of **b** with `b := evalf( evalm( A &* x ) );`.

For Exercise 45, the command `hilbert( n );` produces the $n \times n$ Hilbert matrix. To obtain the $n \times n$ Hilbert matrix with floating-point entries use `evalf( hilbert( n ) );`.

SECTION 2.4 Partitioned Matrices

Partitioned matrices can be created in Maple with the `blockmatrix` command. For example, if A, B, C, E, F, and G are matrices of appropriate sizes, then the command

```
> M := blockmatrix( 2, 3, [A,B,C, E,F,G] );
```

creates a larger matrix of the form

$$M = \begin{bmatrix} A & B & C \\ E & F & G \end{bmatrix}.$$

Once M is formed, there is no record of the partition that was used to create M. For instance, although B is the (1,2)-block used to create M, the value of `M[1,2]` is the same as the (1,2)-entry of A. You should be aware, however, that if `A[1,2]` is an unevaluated name at the time the `blockmatrix` command is executed and is subsequently assigned a value, this assignment to `A[1,2]` changes *all* instances of `A[1,2]` in the current Maple session. In this case, an assignment of a value to `A[1,2]` will also change the (1,2)-entry of M.

Other `linalg` commands that can be useful in the creation of new matrices from existing matrices include: `stackmatrix`, `augment`, `concat`, `delrows`, `delcols`, `diag`, `extend`, `minor`, and `submatrix`.

SECTION 2.5 LU Factorization and the `linsolve` Command

Row reduction of A using the command `gauss` will produce the intermediate matrices needed for an LU factorization of A. You can try this on the matrix in Example 2, stored as Exercise 33 of this section in the `laylinalg` package. The matrices in (5) for Example 2 (page 145 of the text) are produced by the commands:

```
> c2s5( 33 );       # load matrix A
> U = gauss(A,1); # U has 0's below the first pivot
> U = guass(U,2); # now U has 0's below pivots 1 and 2
> U = gauss(U,3); # the echelon form
```

You can copy the information from the screen onto your paper, and divide by the pivot entries to produce L as in the text.

To construct a permuted LU factorization, use the Maple command

```
> U := gauss( U, r, v );
```

where `r` is the row index of the pivot and `v` is a row vector that lists the rows to be changed by replacement operations. For example, if A has 5 rows and the first pivot is in row 4, use `U := gauss( A, 4, [1,2,3,5] );`. If the next pivot is in row 2, use `U := gauss( U, 2, [1,3,5] );`. To build the permuted matrix L, use full columns from A or the partially reduced U, divided by the corresponding pivot. Then change entries to zero if they are in a row already selected as a "pivot row".

The Maple command `U := LUdecomp( A, L='L', P='P' );` produces a permuted LU factorization, $A = PLU$, for any matrix A.

When A is invertible, the simplest way to solve $A\mathbf{x} = \mathbf{b}$ with Maple is to use the `linsolve` command: `x := linsolve(A,b);`. The command `x := inverse(A) &* b;` is less efficient and can be less accurate.

SECTION 2.8 `rref`

The Maple command `rref(A);` produces the reduced echelon form of the matrix A. From this matrix you can write a basis for Col A or write the homogeneous equations that describe Nul A. (Do not forget that A is a coefficient matrix, not an augmented matrix.)

SECTION 2.9 `rank`

You can use `rref( A );` to check the rank of a matrix A. For a matrix with floating-point entries, roundoff error or an extremely small pivot can produce an incorrect echelon form. A more reliable command for computing the rank of a matrix A is `rank( A );`.

SECTION 3.2 Computing Determinants

To compute $\det A$, define `U:=A;` and then repeatedly use the commands `U:=gauss(U,r);` and `U:=swap(U,r,s);` as needed to reduce A to an echelon form U (see the Maple Note for Section 1.4 on page MP-6). Then, the determinant of A is given by the command

```
> detA := (-1)^k * mul( U[i,i], i=1..n );
```

where A is an $n \times n$ matrix and k is the number of row swaps made while reducing A to U.

The command `det( A );` can be used to check your work, but the longer sequence of commands is preferred — at this time — because it emphasizes the *process* of computing determinants.

SECTION 4.1 Graphing Functions

The following command can be used to graph the function f in Exercise 37:

```
> plot( f, t=0..2*Pi );
```

In Maple's `plot` command, the first argument is the function (or list of functions) to be plotted and the second argument contains the name of the independent variable (here, t) and an interval on which the plot is to be created. Note that `Pi` is Maple's name for the constant π. Consult Maple's online help for the `plot` command and `plots` package for optional arguments and additional examples.

SECTION 4.3 `rref` and `genmatrix`

The Maple command `rref(A)` produces the reduced row echelon form of the matrix A. From this matrix you can write a basis for Col A or write the homogeneous equations that describe Nul A. (Do not forget that A is a coefficient matrix, not an augmented matrix.)

For Exercises such as 37 and 38 the `genmatrix` command can be used to generate the coefficient matrix A from a list or set of linear equations involving a set of unknowns, say `c[1]`, `c[2]`, ... For example, the coefficient matrix for Exercise 38 can be obtained by the following commands:

```
> EQN := add( c[j] * cos(t)^j, j=0..6 ) = 0; # genl form of eqn
> eq[0] := eval( EQN, t=0 );                 # eqn w/ t = 0
⋮
> eq[6] := eval( EQN, t=Pi );                # eqn w/ t = π
> A := genmatrix( [ eq[k] $ k=0..6 ],        # coef matrix
>                 [ c[k] $ k=0..6 ] );
```

SECTION 4.4 `linsolve`

The Maple command `linsolve( A, b );` produces the solution of $A\mathbf{x} = \mathbf{b}$ when A is invertible. The `linsolve` command has additional capabilities that will be introduced later, after you have the appropriate background.

SECTION 4.6 `rref`, `rank` and `randomint`

In this course, you can use either `rref( A );` or `rank( A );` to check the rank of a matrix A. In practical work, particularly with floating-point matrices, you should use the more reliable `rank( A );`.

The `laylinalg` command `randomint( m, n );` creates an $m \times n$ matrix of random integers between -9 and 9.

Note:

- In earlier editions the `randomint` command was called `randint`. The name change was necessary because MATLAB now has a `randint` command in its Communications Toolbox.

SECTION 4.7 Change-of-Coordinates Matrix

The `laylinalg` package includes data for Exercises 7–10 and 17–19. The Maple command `rref` can be used to row reduce a matrix such as

$$[\ \mathbf{c}_1 \quad \mathbf{c}_2 \quad \mathbf{b}_1 \quad \mathbf{b}_2\ ]$$

to the desired form. The command `augment`, introduced in the Maple Note for Section 1.3 on page MP-6 can be used to create a matrix from a collection of column vectors.

SECTION 4.8 `solve` and `polyroots`

In Exercises 7–16 and 25–28, the coefficients of the polynomial in the auxiliary equation are stored in a vector $\mathbf{p}$. The coefficients are stored in descending order. For example, the polynomial $x^2 + 6x + 9$ would be represented with `p := vector(3,[1,6,9]);`.

The `polyroots` command in the `laylinalg` package computes floating-point approximations to all (real and complex) roots of the polynomial whose coefficients are stored in the vector **p**. The syntax is `polyroots( p );`.

Given the coefficient vector **p** the corresponding polynomial can be created with the following commands:

```
> n := vectdim(p);                    # degree of polynomial
> P := add( p[k]*x^(n-k), k=1..n );   # polynomial w/ coefs p
```

The exact roots of the polynomial can now be found using

```
> solve( P=0, x );
```

Note, however, that it might not be possible to find all roots for a polynomial of degree 5 or higher.

SECTION 4.9

The Maple box for Section 1.10 contains information that is useful for the Exercises in this section.

SECTION 5.1 Finding Eigenvectors

The `laylinalg` package contains a command that will simplify your homework by automatically producing a basis for an eigenspace when you know the eigenvalue. For example, if A is a 3×3 matrix with eigenvalue $\lambda = 7$, then the command `nulbasis( A-7*diag(1$3) );` returns a matrix whose columns form a basis for the corresponding eigenspace. The command `nulbasis( C );` returns a matrix whose columns form a basis for Nul C (the same basis you would get if you started with `rref(C);` and computed the basis by hand).

Additional Maple commands for computing eigenvalues and eigenvectors will be discussed later. You should use `nulbasis` now, to reinforce the basic concepts of this chapter.

For Exercises 37–40 use the `eigenvalues` command to obtain the eigenvalues of a matrix. For example, the commands

```
> lambda := eigenvalues( A );                 # eigenvalues of A
> nulbasis( A - lambda[2]*diag(1$5) );        # basis for e-space of λ2
```

compute a basis for the eigenspace corresponding to the second eigenvalue of a 5×5 matrix A. To avoid typing or numerical errors, you are encouraged to avoid looking at the list of eigenvalues and explicitly typing the eigenvalue in the argument of the `nulbasis` command. It is much safer — and faster — to reference the eigenvalues as in the commands above.

SECTION 5.2 `charpoly`

The Maple command `charpoly` can be used to check your answers in Exercises 9–14. Note that if A is $n \times n$, `charpoly(A,lambda)` is $\det(\lambda I - A)$. If n is even, this is the characteristic polynomial of A. When n is odd, this is the negative of the characteristic polynomial of A (as defined in the text).

For Exercises 28 and 29, create a 4×4 matrix with random integer entries with the command `randomint( 4 );`. For Exercise 29, use `gauss` and perhaps `swap` to create the echelon form

without row scaling. See also the Maple Note for Section 1.4 on page MP-6.

The following commands create the graph of the characteristic polynomial of the matrix A in Exercise 30 for all values of the parameter a in a list `a_vals`:

```
> p := charpoly( A, lambda ); # char poly in terms of parameter a
> P := [seq( p, a=a_vals )];  # char poly for each value of a
> plot( P, lambda=0..3 );     # graph all char polys on same axes
```

SECTION 5.3 Diagonalization and `eigenvalues`

To practice the diagonalization procedure in this section, you should use the `nulbasis` command to produce eigenvectors. For Exercises 33–36, the command `ev := eigenvalues( A );` can be used to provide the eigenvalues. See also the Maple Note for Section 5.1 on page MP-13.

SECTION 5.5 Diagonalization and `eigenvectors`

Once you become familiar with the diagonalization process described in the Maple Note for Section 5.3, you can use other Maple commands to simplify the process of determining the eigenvectors for a matrix.

The command `eigenvectors(A);` returns, for each eigenvalue of A, a list containing three pieces of information: the eigenvalue, the (algebraic) multiplicity of the eigenvalue, and a basis for the corresponding eigenspace. The data structure is a little complicated, but the following sequence of commands illustrates how to find the matrices P and D such that $AP = PD$:

```
> EV := [ eigenvectors( A ) ];              # e-values and e-spaces
> lambda := seq( op(1,e)$op(2,e), e=EV );   # e-values (w/repetition)
> espace := seq( op(op(3,e)), e=EV );       # e-space basis
> DD := diag( lambda );                     # diag e-value matrix
> P := augment( espace );                   # e-vector matrix
```

Note that the matrix P may not be the same as the one you find by hand. A quick check that your answer is correct is: `iszero( A&*P - P&*DD );`. The result will be *true* if the argument is a zero matrix. When the matrices involve floating-point entries, Maple may return *false* when one entry is very small (but not exactly zero). In any case, a visual inspection should be performed on the difference $AP - PD$.

SECTION 5.5 Complex Eigenvalues, `Re`, and `Im`

If A is a 2×2 real matrix and if `eigenvalues( A );` returns a pair of complex conjugate eigenvalues, you can use the `nulbasis` command to find a complex eigenvector $\mathbf{v}$ corresponding to one of these eigenvalues. Then, build a 2×2 real matrix P from the real and imaginary parts of $\mathbf{v}$ and compute $C = P^{-1}AP$. This can be done with the following commands:

```
> V := nulbasis( A - lambda[1]*diag(1$2) ); # complex e-vector
> Vr := evalm( Re( V ) );                   # real part
> Vi := evalm( Im( V ) );                   # imaginary part
> P := augment( Vr, Vi );
> C := evalm( inverse(P) &* A &* P );
```

In general, the real and imaginary parts of a complex-valued vector $\mathbf{v}$ are obtained with the commands `evalm( Re(v) );` and `evalm( Im(v) );`, respectively.

SECTION 5.6 Plotting Discrete Trajectories

The following sequence of commands creates the sequence of vectors $\mathbf{x}$, $A\mathbf{x}$, $A^2\mathbf{x}$, ..., $A^{15}\mathbf{x}$ for a given initial vector $\mathbf{x}$:

```
> x := vector( [1,0] );          # initial vector (change as needed)
> T := convert( x, list ):       # initialize list of points
> for j from 1 to 15 do          # change 15 to number of iterations
>   x := evalm( A&*x );          # compute the "next" point
>   T := T, convert( x, list ); # add new point to list
> end do;
```

These points can be plotted with the command

```
> plot( [T], style=POINT );
```

Omitting the second argument of the `plot` command will "connect the dots" between the points. Other optional arguments can be used to further customize the plot. The `display` command — in the `plots` package — can be used to combine multiple trajectories into a single plot. (Do not forget to execute the command `with(plots);` to load the `plots` package.)

For Exercise 17, the following commands create a graph of the first component in each data point in the list `T` constructed above

```
> T1 := [seq( [i-1,T[i,1]], i=1..nops([T]) )];
> plot( T1 );
```

To graph the sum of the first two components of each data point in `T` replace `T1` with `T2` defined by

```
> T2 := [seq( [i-1,T[i,1]+T[i,2]], i=1..nops([T]) )];
```

Similarly, to plot the quotient of the first two components in each data point in `T` replace `T1` with `T2` defined by

```
> T3 := [seq( [i-1,T[i,1]/T[i,2]], i=1..nops([T]) )];
```

SECTION 5.7 Plotting Trajectories for Differential Equations

The following template of Maple commands can be used to create a plot of the direction field and a solution curve for the 2×2 system of differential equations in Example 1.

```
> with( DEtools );                              # load DEtools pkg
> A := matrix( 2, 2,
>              [ [-1.5, 0.5], [1, -1] ] );      # enter coef matrix
> SYS := [ diff(y1(t),t)                        # DO NOT CHANGE
>           = A[1,1]*y1(t) + A[1,2]*y2(t),      # DO NOT CHANGE
>          diff(y2(t),t)                        # DO NOT CHANGE
>           = A[2,1]*y1(t) + A[2,2]*y2(t) ];    # DO NOT CHANGE
> IC := [ y1(0)=5, y2(0)=4 ];                   # enter init cond
> RANGE := t = 0..10;                           # enter time interval
> WINDOW := y1 = -5..5, y2 = -4..4;             # enter plot window
> DEplot( SYS, [y1(t),y2(t)],                   # DO NOT CHANGE
>         RANGE, [IC], WINDOW );                # DO NOT CHANGE
```

To create plots for other systems, simply modify the coefficient matrix and the initial condition then modify the time interval and window to create the desired plot.

Note:

- The `with( DEtools );` command needs to be entered only once — anytime before the first use of `DEplot`.

SECTION 5.8 Power Method and Inverse Power Method

A Maple implementation of the Power Method (page 363 of the text) assumes that A is an $n \times n$ matrix with a strictly dominant eigenvalue and an initial vector $\mathbf{x}_0$:

```
> x := evalm( x0 );           # initial vector
> for k from 1 to 15 do       # change 15 to number of iterations
>   mu := norm( x );          # largest element in x
>   x := evalm( A &* x/mu );  # next iterate in power method
> end do;
```

The value of μ should approach the dominant eigenvalue and the vector $\mathbf{x}$ should approach a corresponding eigenvector.

One possible implementation of the Inverse Power Method is:

```
> I3 := diag(1$3);                    # 3 × 3 identity matrix
> alpha := 3.3;                       # initial guess at e-value
> x := vector( [1,1,1] );             # initial guess at e-vector
> for k from 1 to 15 do               # change bounds as needed
>   y := linsolve( A-alpha*I3, x );   # solve (A − αI)y_k = x_k
>   mu := norm( y );                  # largest element in abs value
>   nu := alpha + 1/mu;               # updated approx to e-value
>   x := evalm( y/mu );               # updated approx to e-vector
> end do;
```

Notes:

- The initial value of α should be reasonably close to the desired eigenvalue and $\mathbf{x}$ is a vector whose largest element (in absolute value) is 1.
- Recall that `norm(x)` returns the maximum of the absolute values of the entries in $\mathbf{x}$.

SECTION 6.1 `innerprod` and `norm`

The Maple command `innerprod(u,v)` returns the inner product of two real-valued column vectors $\mathbf{u}$ and $\mathbf{v}$. The associated length of a vector $\mathbf{v}$ is `norm(v,2)`. (See also the Maple Note for Section 2.1 on page MP-8.)

SECTION 6.2 Orthogonality

In Exercises 1–9 and 17–22, a fast way to test the orthogonality of a set such as $\{\mathbf{u}_1, \mathbf{u}_2, \mathbf{u}_3\}$ is to create a matrix

```
> U := augment( u1, u2, u3 );
```

whose columns are the vectors from the set, and test whether

```
> evalm( transpose(U) &* U);
```

yields a diagonal matrix.

Notes:

- To understand how this works, see Theorem 6 on page 390.
- The `orthog` command could also be used, but this does not emphasize the essential properties of orthogonality.

SECTION 6.3 Orthogonal Projections

The orthogonal projection of a vector onto a single vector was described in the Maple Note for Section 6.2. The orthogonal projection onto the set spanned by an orthogonal set of vectors is the sum of the one-dimensional projections. Another way to construct this projection is to normalize the orthogonal vectors, place these orthonormal vectors in the columns of a matrix U, and apply Theorem 10. For instance, if $\{\mathbf{y}_1, \mathbf{y}_2, \mathbf{y}_3\}$ is an orthogonal set of nonzero vectors, then the matrix U defined by the Maple command:

```
> U := augment( y1/norm(y1,2),
>               y2/norm(y2,2),
>               y3/norm(y3,2) );
```

has orthonormal columns. The orthogonal projection of $\mathbf{y}$ onto $\text{Span}\{\mathbf{y}_1, \mathbf{y}_2, \mathbf{y}_3\}$ is obtained with the command:

```
> evalm( U &* (transpose(U) &* y) );
```

SECTION 6.4 The Gram-Schmidt Process, `proj`, and `gs`

If A has only two linearly independent columns, then the Gram-Schmidt process is:

```
> v1 := col(A,1);
> v2 := evalm( col(A,2)
>            - innerprod(col(A,2),v1)/innerprod(v1,v1) * v1 );
```

If A has three columns, add the command:

```
> v3 := evalm( col(A,3)
>            - innerprod(col(A,3),v1)/innerprod(v1,v1) * v1
>            - innerprod(col(A,3),v2)/innerprod(v2,v2) * v2 );
```

The generalization to larger matrices should now be apparent.

Notice that Maple commands in the form

```
> evalm( innerprod(y,u)/innerprod(u,u) * u );
```

provide the orthogonal projection of a vector $\mathbf{y}$ onto a vector $\mathbf{u}$. A simpler, but less illustrative, alternative is to use the `proj` command from the `laylinalg` package. The syntax is:

```
>  proj( y, convert(u,matrix) );
```

After you are familiar with the Gram-Schmidt process, use the command `proj( x, V );` to compute the projection of a vector $\mathbf{x}$ onto the subspace spanned by the columns of a matrix V. For example:

```
> v2 := evalm( col(A,2) - proj( col(A,2), augment(v1) ) );
> v3 := evalm( col(A,3) - proj( col(A,3), augment(v1,v2) ) );
```

Although you should construct the QR decomposition of a matrix A using the approach in the text, you can check your construction of the matrix Q with the command `Q := gs(A);`. The `gs` command from the `laylinalg` package provides an even faster method of applying the Gram-Schmidt process. The syntax is simply: `gs(A);`. After you are very familiar with the Gram-Schmidt process the use of `gs` saves time.

SECTION 6.5 `linsolve` and `leastsqrs`

When A has linearly dependent columns, you can write the general description of all least-squares solutions on paper after you row reduce the augmented matrix for the normal equations:

```
> rref( augment( transpose(A)&*A, transpose(A)&*b ) );
```

When A has linearly independent columns, a Maple command to solve the normal equations is:

```
> linsolve( transpose(A)&*A, transpose(A)&*b );
```

Other approaches to solving the normal equations include the use of `rref` as described above and the explicit use of $(A^TA)^{-1}$. For Exercises 15 and 16, see the Numerical Note on page 415 in the text and use `linsolve( R, transpose(Q)&*b );` to solve $R\mathbf{x} = Q^T\mathbf{b}$.

When A is not square but has linearly independent columns, a common algorithm for solving the overdetermined system $A\mathbf{x} = \mathbf{b}$ is to find Q and R and then solve $R\mathbf{x} = Q^T\mathbf{b}$. You should use the normal equations or the QR factorization. This will give you a solid conceptual background for applying least-squares techniques later in your career.

For Exercise 26, the command `A := stackmatrix( A1, A2 );` creates the (partitioned) matrix whose top block is A_1 and bottom block is A_2. Of course, each block must have the

same number of columns. (See also the Maple Note for Section 2.4 on page MP-10 for more information on partitioned matrices.)

SECTION 6.6 Computing Regression Coefficients

Maple can be used to construct the design and observation vectors from a list of data points. For example, for Exercise 1:

```
> c6s6( 1 );                          # load data from laylinalg
> n := vectdim( x );                  # number of data points
> X := augment( vector(n,1), x );     # design matrix
```

The least squares solution to $X\boldsymbol{\beta} = \mathbf{y}$ as described in the Maple Note for Section 6.5.

SECTION 6.6 `leastsqrs`, Functions of Vectors, and `map`

Once you create the design matrix X and the observation vector $\mathbf{y}$, your computation for least-squares solutions here are the same as those described in the Maple Note for Section 6.5. Here, A and $\mathbf{b}$ are replaced by X and $\mathbf{y}$, respectively. The Maple command

```
> rref( augment( transpose(X)&*X, transpose(X)&*y ) );
```

leads to the general description of all least-squares solutions. When X has linearly independent columns, the command

```
> linsolve( transpose(X)&*X, transpose(X)&*y );
```

creates the least-squares solution. In subsequent courses, you may choose simply to use `leastsqrs( X, y );`, which also produces all least-squares solutions.

To construct the design matrix for an exercise in this section, you may need Maple's `map` command to apply a function to each element of a vector. If $\mathbf{x}$ is a vector and f is a Maple function, then `map( f, x );` is a vector the same size as $\mathbf{x}$ whose entries are the result of applying f to the entries in $\mathbf{x}$. For example, in Exercise 7 the $\mathbf{x}$ vector can be created as above and the vector of the square of each element is `x2 := map( u->u^2, x );`. The design matrix is now easily constructed with the use of `augment`. When f is a built-in Maple function, say the exponential function `exp`, the syntax is somewhat simpler: `map( exp, x );`.

SECTION 6.7 `int` and Inner Products on $C[0, 2\pi]$

The Maple command to evaluate the definite integral $\int_a^b f\ dt$ is `int( f, t=a..b );`. To solve Exercise 28, begin by creating a Maple procedure to compute the inner product of two functions in $C[0, 2\pi]$:

```
> ip := (f,g) -> int(f*g,t=0..2*Pi);
```

The inner product procedure, `ip`, and the four functions p_0, p_1, p_2, and p_3 can be defined by executing `c6s7(28);`. The Gram–Schmidt process described in the Maple Note for Section 6.4 on page MP-17 is applicable here. For example, the first two steps are:

```
> q0 := p0;
> q1 := p1 - ip(p1,q0)/ip(q0,q0) * q0;
```

Observe that the only difference between these commands and the ones introduced in Section 6.4 is that `innerprod` is replaced with the newly-defined inner product `ip`.

SECTION 6.8 Graphing Functions

The `plot` command can be used to plot one or more functions. For example, the two plots in Figure 3 for Example 4 on page 442 in the text can be created as follows:

```
> f := t;                          # define function f
> F3 := Pi - 2*sin(t) - sin(2*t)
>          - 2/3*sin(3*t);         # 3rd-order Fourier approx
> plot( [f,F3], t=0..2*Pi );       # Figure 3 (a)
> F4 := F3 - 2/4*sin(4*t);         # 4th-order Fourier approx
> plot( [f,F4], t=0..2*Pi );       # Figure 3 (b)
```

The Maple Note for Section 4.1 on page MP-11 contains additional information about Maple's `plot` command.

SECTION 7.1 Orthogonal Diagonalization

Use the `eigenvalues` and `nulbasis` commands to find the eigenvalues and corresponding eigenvectors as in Section 5.3 on page MP-14. If you encounter a two-dimensional eigenspace, with a basis $\{\mathbf{v}_1, \mathbf{v}_2\}$, use the command

```
> v2 := evalm( v2 - innerprod(v2,v1)/innerprod(v1,v1) * v1 );
```

or

```
> v2 := evalm( v2 - proj(v2,augment(v1)) );
```

to make the two eigenvectors orthogonal. See the Maple Note for Section 6.4 on page MP-17. After you normalize the vectors and create P, check whether $P^T P = I$ to verify that P is indeed an orthogonal matrix.

SECTION 7.4 The Singular Value Decomposition

The singular value decomposition of a matrix A is constructed from the orthogonal diagonalization of $A^T A$. The orthogonal diagonalization of a matrix is discussed in the Maple Note for Section 7.1 on page MP-20. Note that the eigenvalues of $A^T A$ may not be sorted in decreasing order. You may need to reorder some of the information from the orthogonal diagonalization when creating the diagonal matrix D and matrix of right singular vectors V. For example, if $A^T A$ is a 3×3 matrix and `lambda` is a list of the eigenvalues of $A^T A$ with, say, `lambda[3]` > `lambda[1]` > `lambda[2]`, then the command

```
> Sigma := diag( seq( sqrt(lambda[i]), i=[3,1,2] ) );
```

creates the diagonal matrix of singular values of A. Note that Σ must have the same dimensions as A; the `extend` command can be used to add one or more rows or columns of zeros to the diagonal matrix created above. For example,

```
> Sigma := extend( Sigma, 1, 0, 0 );
```

adds one row of zeros to the bottom of the diagonal matrix. The matrix V would then be created by:

```
> V := augment( seq( col(P,i), i=[3,1,2] ) );
```

The matrix of left singular vectors, U, can be formed by normalizing the nonzero columns of the product AV. If U is not square, the missing columns must be chosen to form an orthonormal

basis for Nul A^T. (See Figure 4 on page 479 in the text.) The command

```
> GramSchmidt( nullspace( transpose(A) ), normalized );
```

produces this orthonormal basis. Thus, the square matrix U defined by

```
> U := augment( U, op(GramSchmidt( nullspace( transpose(A) ),
>                                  normalized )) );
```

This construction helps you to think about the fundamental properties of the singular value decomposition. There are two additional Maple commands that can be used to directly obtain information about the singular values of a matrix. The `singularvals` command returns the singular values of A as a list but does not provide any information about the singular vectors. The `Svd` command computes floating-point approximations to the singular values and vectors of a numeric matrix A. The command

```
> SV := evalf( Svd( A, U, V ) );
```

returns the singular values in the table `SV`. The matrices U and V contain the right and left singular vectors, respectively. To create the diagonal matrix Σ from the table of singular values, use the command:

```
> Sigma := diag( SV[i] $ i=1..vectdim(SV) );
```

SECTION 7.5 Computing Principal Components

The p-dimensional vector containing the mean of each row of a $p \times N$ matrix X can be created with the command:

```
> Rmean := vector([ seq( add( x, x=convert(row(X,i),list) )
>                        / coldim(X), i=1..rowdim(X) ) ]);
```

The $p \times N$ matrix containing N copies of the row averages is obtained with:

```
> augment( Rmean $ coldim(X) );
```

Thus, to convert the data in X into mean-deviation form, use

```
> B := evalm( X - augment( Rmean $ coldim(X) ) );
```

The sample covariance matrix is produced by:

```
> S := evalm( B &* transpose(B) / (coldim(X)-1) );
```

The principal component analysis is completed by performing a singular value decomposition on the matrix $A = \frac{1}{\sqrt{N-1}} B^T$. (See the Numerical Note at the end of Section 7.5 of the text.)

INDEX OF MAPLE COMMANDS