72

Linear Algebra in Maple®

David. J. Jeffrey

Robert M. Corless

72.1 Introduction

Maple® is a general purpose computational system that combines symbolic computation with exact and approximate (floating-point)numerical computation and offers a comprehensive suite of scientific graphics as well. The main library of functions is written in the Maple programming language, a rich language designed to allow easy access to advanced mathematical algorithms. A special feature of Maple is user access to the source code for the library, including the ability to trace Maple's execution and see its internal workings; only the parts of Maple that are compiled, for example, the kernel, cannot be traced. Another feature is that users can link to LAPACK library routines transparently, and thereby benefit from fast and reliable floating-point computation. The development of Maple started in the early 80s, and the company **Maplesoft** was founded in 1988. A strategic partnership with NAG Inc. in 2000 brought highly efficient numerical routines to Maple, including LAPACK.

There are two linear algebra packages in Maple: `LinearAlgebra` and `linalg`. The `linalg` package is older and considered obsolete; it was replaced by `LinearAlgebra` in MAPLE 6. Here we describe only the `LinearAlgebra`

Facts:

1. Maple commands are typed after a prompt symbol, which by default is "greater than" (>). In examples below, keyboard input is simulated by prefixing the actual command typed with the prompt symbol.
2. In the examples below, some of the commands are too long to fit on one line. In such cases, the Maple continuation character backslash (\ ) is used to break the command across a line.
3. Maple commands are terminated by either semicolon (;) or colon (:). Before Maple 10, a terminator was required, but in the Maple 10 GUI it can be replaced by a carriage return. The semicolon terminator allows the output of a command to be displayed, while the colon suppresses the display (but the command still executes).
4. To access the commands described below, load the `LinearAlgebra` package by typing the command (after the prompt, as shown)
   ```
   > with( LinearAlgebra );
   ```
   If the package is not loaded, then either a typed command will not be recognized, or a different command with the same name will be used.
5. The results of a command can be assigned to one or more variables. Thus,
   ```
   > a := 1 ;
   ```
   assigns the value 1 to the variable *a*, while
   ```
   > (a,b,c) := 1,2,3 ;
   ```
   assigns *a* the value 1, *b* the value 2 and *c* the value 3. *Caution*: The operator colon-equals (:=) is assignment, while the operator equals (=) defines an equation with a left-hand side and a right-hand side.
6. A sequence of expressions separated by commas is an expression sequence in Maple, and some commands return expression sequences, which can be assigned as above.
7. Ranges in Maple are generally defined using a pair of periods (..). The rules for the ranges of subscripts are given below.

72.2 Vectors

3. A Gram–Schmidt exercise.

```
> u1 := <3|0|4>: u2 := <2|1|1>: w1n := u1/Norm( u1, 2 );
```

$$w1n := [3/5, 0, 4/5]$$

```
> w2 := u2 - (u2 . w1n) w1n; w2n := w2/Norm( w2, 2 );
```

$$w2n := \frac{2\sqrt{2}}{5}, \frac{\sqrt{2}}{2}, \frac{3\sqrt{2}}{10}$$

4. Vectors with complex elements. Define column vectors $\mathbf{u_c}$,$\mathbf{v_c}$ and row vectors $\mathbf{u_r}$,$\mathbf{v_r}$.

```
> uc := <1 I,2>: ur := Transpose( uc ): vc := <5,2 . 3*I>:
vr := Transpose( vc ):
```

The inner product of column vectors conjugates the first vector in the product, and the inner product of row vectors conjugates the second.

```
> inner1 := uc . vc; inner2 := ur . vr;
                inner1 := 9-11 I , inner2 := 9+11 I
```

Maple computes the product of two similar vectors, i.e., both rows or both columns, as a true mathematical inner product, since that is the definition possible; in contrast, if the user mixes row and column vectors, then Maple does not conjugate:

```
> but := ur . vc;
```

$$but := 9 . I$$

Caution: The use of period (.) with complex row and column vectors together differs from the use of period (.) with complex $1 \times m$ and $m \times 1$ matrices. In case of doubt, use matrices and conjugate explicitly where desired.

72.3 Matrices

Facts:

1. One-column matrices and vectors are not interchangeable in Maple.
2. Matrices and two-dimensional arrays are not interchangeable in Maple.

Commands:

1. Generation of Matrices.

- `Matrix( [[a,b,...],[c,d,...],...] )` Construct a matrix row-by-row, using a list of lists.
- `a|b|...>,<c|d|...>,...>...>ol4sta4-174.4(l5st)-174-1(o)D thes.`

- `Matrix( n, m, (i,j) >f(i,j) )` Construct a matrix $n \times m$ using a function $f(i, j)$ to define the elements. $f(i, j)$ is evaluated sequentially for i from 1 to n and j from 1 to m. The notation `(i,j) >f(i,j)` is Maple syntax for a bivariate function $f(i, j)$.
- `Matrix( n, m, fill=a )` An $n \times m$ matrix with each element equal to a.
- `Matrix( n, m, symbol=a )` An $n \times m$ matrix containing subscripted entries a_{ij}.
- `map( x >f(x), M )` A matrix obtained by applying $f(x)$ to each element of M.

  [*Caution*: the command is `map` *not* `Map`.]
- `< < A | B>, < C | D> >` Construct a partitioned or block matrix from matrices A, B, C, D. Note that `< A|B >` ä

Examples:

1. A matrix product.

The rules for mixed products are

`Vector[row](`

72.4 Arrays

Before describing Maple's `Array` structure, it is useful to say why Maple distinguishes between an `Array` and a `Vector` or `Matrix`, when other books and software systems do not. In linear algebra, two different types of operations are performed with vectors or matrices. The first type is described in sections 72.2 and 72.3, and comprises operations derived from the mathematical structure of vector spaces. The other type comprises operations that treat vectors or matrices as data arrays; they manipulate the individual elements directly. As an example, consider dividing the elements of `Array` [1, 3, 5] by the elements of [7, 11, 13] to obtain [1/7, 3/11, 5/13].

The distinction between the operations can be made in two places: In the name of the operation or the name of the object. In other words we can overload the data objects or overload the operators. Systems such as MATLAB choose to leave the data object unchanged, and define separate operators. Thus, in MATLAB the statements [1, 3, 5]/[7, 11, 13] and [1, 3, 5]./[7, 11, 13] are different because of the operators. In contrast, Maple chooses to make the distinction in the data object, as will now be described.

Facts:

1. The Maple `Array` is a general data structure akin to arrays in other programming languages.
2. An array can have up to 63 indexes and each index can lie in any integer range.
3. The description here only addresses the overlap between Maple `Array` and `Vector`.

Cautions:

1. A Maple `Array` might look the same as a vector or matrix when printed.

Commands:

1. Generation of arrays.
   - `Array`([$x_1, x_2, \ldots$

2. Getting `Vectors` and `Arrays` to do the same thing.
```
> Transpose( map( x->x x, <1,2,3> ) ) -convert( Array( [1,2,3] )◢2,
Vector );
```
$$[0, 0, 0]$$

72.5 Equation Solving and Matrix Factoring

Cautions:

1. If a matrix contains exact numerical entries, typically integers or rationals, then the material studied in introductory textbooks transfers to a computer algebra system without special considerations. However, if a matrix contains symbolic entries, then the fact that computations are completed without the user seeing the intermediate steps can lead to unexpected results.
2. Some of the most popular matrix functions are discontinuous when applied to matrices containing symbolic entries. Examples are given below.
3. Some algorithms taught to educate students about the concepts of linear algebra often turn out to be ill-advised in practice: Computing the characteristic polynomial and then solving it to find eigenvalues, for example; using Gaussian elimination without pivoting on a matrix containing floating-point entries, for another.

Commands:

1. `LinearSolve( A, B )` The vector or matrix X satisfying $AX \quad B$.
2. `BackwardSubstitute( A, B ), ForwardSubstitute( A, B )` The vector or matrix X satisfying $AX \quad B$ when A is upper or lower triangular (echelon) form respectively.
3. `ReducedRowEchelonForm( A )`. The reduced row-echelon form (RREF) of the matrix A. For matrices with symbolic entries, see the examples below for recommended usage.
4. `Rank( A )` The rank of the matrix A. *Caution*: If A has floating-point entries, see the section below on Numerical Linear Algebra. On the other hand, if A contains symbolic entries, then the rank may change discontinuously and the generic answer returned by `Rank` may be incorrect for some specializations of the parameters.
5. `NullSpace( A )` The nullspace (kernel) of the matrix A. *Caution*: If A has floating-point entries, see the section below on Numerical Linear Algebra. Again on the other hand, if A contains symbolic entries, the nullspace may change discontinuously and the generic answer returned by `NullSpace` may be incorrect for some specializations of the parameters.
6. `( P, L, U, R ) := LUDecomposition( A, method='RREF' )` The $PLUR$, or Turing, factors of the matrix A. See examples for usage.
7. `( P, L, U ) := LUDecomposition( A )` The PLU factors of a matrix A, when the RREF R is not needed. This is usually the case for a Turing factoring where R is guaranteed (or known a priori) to be I, the identity matrix, for all values of the parameters.
8. `( Q, R ) := QRDecomposition( A, fullspan )` The QR factors of the matrix A. The option `fullspan` ensures that Q is square.
9. `SingularValues( A )` See section 72.8 Numerical Linear Algebra.
10. `ConditionNumber( A )` See section 72.8 Numerical Linear Algebra.

Examples:

1. Need for Turing factoring.
   One of the strengths of Maple is computation with symbolic quantities. When standard linear algebra methods are applied to matrices containing symbolic entries, the user must be aware of new mathematical features that can arise. The main feature is the discontinuity of standard matrix

functions, such as the reduced row-echelon form and the rank, both of which can be discontinuous. For example, the matrix

$$B \quad A \; \lambda I \quad \begin{bmatrix} 7 \; \lambda & 4 \\ 6 & 2 \; \lambda \end{bmatrix}$$

has the reduced row-echelon form

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \qquad \lambda \; 1, 10$$

$$\text{ReducedRowEchelonForm}(B) \quad \begin{bmatrix} \\ \\ \end{bmatrix}$$

2. QR factoring.
   Maple does not offer column pivoting, so in pathological cases the factoring may not be unique, and will vary between software systems. For example,

```
> A := <0,0>|<5,12> : QRDecomposition( A, fullspan )
```

$$\begin{bmatrix} 5/13 & 12/13 \\ 12/13 & 5/13 \end{bmatrix}, \quad \begin{bmatrix} 0 & 13 \\ 0 & 0 \end{bmatrix}$$

72.6 Eigenvalues and Eigenvectors

Facts:

1. In exact arithmetic, explicit expressions are not possible in general for the eigenvalues of a matrix of dimension 5 or higher.

```
> L;
```

$$\_Z^4 \quad 13\,\_Z^3 \quad 4\,\_Z^2 \quad 319\,\_Z \quad 386,$$

`ForwardSubstitute, Identity, Inverse, LUApply, LUDecomposition, LinIntSolve, MatBasis, MatGcd, Mod, Multiply, Permute, Random, Rank, RankProfile, RowEchelonTransform, RowReduce, Swap, Transpose, ZigZag]`

2. Arithmetic can be done modulo a prime p or, in some cases, a composite modulus m.
3. The relevant matrix and vector datatypes are `integer[4]`, `integer[8]`, `integer[]`, and `float[8]`. Use of the correct datatype can improve efficiency.

Examples:

```
> p :  13;
> A :  Mod( p, Matrix([[1,2,3],[4,5,6],[7,8,-9]]), integer[4] );
```

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 4 \end{bmatrix}$$

```
> Mod( p, MatrixInverse( A ), integer[4] );
```

$$\begin{bmatrix} 12 & 8 & 5 \\ 0 & 11 & 3 \\ 5 & 3 & 5 \end{bmatrix}$$

Cautions:

1. This is not to be confused with the `mod` utilities, which together with the inert `Inverse` command, can also be used to calculate inverses in a modular way.
2. One must always specify the datatype in `Modular` commands, or a cryptic error message will be generated.

72.8 Numerical Linear Algebra in Maple

The above sections have covered the use of Maple for exact computations of the types met during a standard first course on linear algebra. However, in addition to exact computation, Maple offers a variety of floating-point numerical linear algebra support.

Facts:

1. Maple can compute with either "hardware floats" or "software floats,"
2. A hardware float is IEEE double precision, with a mantissa of (approximately) 15 decimal digits.
3. A software float has a mantissa whose length is set by the Maple variable `Digits`.

Cautions:

1. If an integer is typed with a decimal point, then Maple treats it as a software float.
2. Software floats are significantly slower that hardware floats, even for the same precision.

Commands:

1. `Matrix( n, m, datatype=float[8] )` An $n \times m$ matrix of hardware floats (initialization data not shown). The elements must be real numbers. The 8 refers to the number of bytes used to store the floating point real number.
2. `Matrix( n, m, datatype=complex(float[8] ))` An $n \times m$ matrix of hardware floats, including complex hardware floats. A complex hardware float takes two 8-byte storage locations.

3. `Matrix( n, m, datatype=sfloat )` An $n \times m$ matrix of software floats. The entries must be real and the precision is determined by the value of `Digits`.
4. `Matrix( n, m, datatype=complex(sfloat) )` As before with complex software floats.
5.

We make a floating-point version of *A* by

```
> Af := Matrix( A, datatype=float[8] );
```

and then take the `NullSpace` of both `A` and `Af`. The nullspace of `A` is correctly returned as the empty set—*A* is not singular (in fact, its determinant is 1). The nullspace of `Af` is correctly returned as

$$\left\{ \begin{bmatrix} 0.707637442412755612 \\ 0.706575721416702662 \end{bmatrix} \right\}$$

The answers are different — quite different — even though the matrices differ only in datatype.

Commands:

1. `SmithForm( B, output=['S','U','V'] )` Smith form of B.

Examples:

The Smith form of

$$B \;\begin{vmatrix} 0 & 4y^2 & 4y & 1 \\ 4y^2 & 4y & 1 & 0 \\ 4y & 1 & 4y^2+2y & 4y^2+y^2+2y^2+2 \\ 1 & 0 & 4y^2+y^2+2y^2+2 & 4y^2+y \end{vmatrix}$$

is

$$S \;\begin{vmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1/4\,y^2+y+1 & 0 \\ 0 & 0 & 0 & (1/64)\,y^2+y+1 \end{vmatrix}$$

Maple also returns two unimodular (over the domain $\mathbb{Q}[y]$) matrices **u** and **v** for which $A = U.S.V$.

72.10 Structured Matrices

Facts:

1. Computer algebra systems are particularly useful for computations with structured matrices.
2. User-defined structures may be programmed using *index functions*. See the help pages for details.
3. Examples of built-in structures include *symmetric*, *skew-symmetric*, *Hermitian*, *Vandermonde*, and *Circulant* matrices.

Examples:

Generalized Companion Matrices. Maple can deal with several kinds of generalized companion matrices. A generalized companion matrix[2] *pencil* of a polynomial $p(x)$ is a pair of matrices C_0, C_1 such that $\det(xC_1 + C_0) = 0$ precisely when $p(x) = 0$. Usually, in fact, $\det(xC_1 + C_0) = p(x)$, though in some definitions proportionality is all that is needed. In the case $C_1 = I$, the identity matrix, we have $C_0 = C(p(x))$ is *the companion matrix* of $p(x)$. MATLAB's `roots` function computes roots of polynomials by first computing the eigenvalues of the companion matrix, a venerable procedure only recently proved stable.

The generalizations allow direct use of alternative polynomial bases, such as the Chebyshev polynomials, Lagrange polynomials, Bernstein (Bézier) polynomials, and many more. Further, the generalizations allow the construction of generalized companion matrix pencils for *matrix polynomials*, allowing one to easily solve *nonlinear eigenvalue problems*.

We give three examples below.

If $p := 3 - 2x - x^2$, then `CompanionMatrix( p, x )` produces "the" (standard) companion matrix (also called Frobenius form companion matrix):

$$\begin{bmatrix} 0 & -3 \\ 1 & -2 \end{bmatrix}$$

[2] Sometimes known as "colleague" or "comrade" matrices, an unfortunate terminology that inhibits keyword search.

and it is easy to see that $\det(tI - C) = p(t)$. If instead

$$p: \quad B_0^3(x) \quad 2B_1^3(x) \quad 3B_2^3(x) \quad 4B_3^3(x)$$

where $B_k^n(x) = \binom{n}{k}(1-x)^{n-k}(x-1)^k$ is the kth Bernstein (Bézier) polynomial of degree n on the interval $-1 \le x \le 1$, then `CompanionMatrix( p, x )` produces the pencil (note that this is not in Frobenius form)

$$C_0 = \begin{bmatrix} 3/2 & 0 & -4 \\ 1/2 & 1/2 & 8 \\ 0 & 1/2 & -\frac{52}{3} \end{bmatrix}$$

$$C_1 = \begin{bmatrix} 3/2 & 0 & -4 \\ 1/2 & 1/2 & 8 \\ 0 & 1/2 & -\frac{20}{3} \end{bmatrix}$$

(from a formula by Jonsson & Vavasis [JV05] and independently by J. Winkler [Win04]), and we have $p(x) = \det(xC_1 - C_0)$. Note that the program does not change the basis of the polynomial $p(x)$ of equation (72.9) to the monomial basis (it turns out that $p(x) = 20 - 12x$ in the monomial basis, in this case: note that C_1 is singular). It is well-known that changing polynomial bases can be ill-conditioned, and this is why the routine avoids making the change.

Next, if we choose nodes $[-1, -1/3, 1/3, 1]$ and look at the degree 3 polynomial taking the values $[1, -1, 1, -1]$ on these four nodes, then `CompanionMatrix( values, nodes )` gives C_0 and C_1 where C_1 is the 5×5 identity matrix with the $(5,5)$ entry replaced by 0, and

$$C_0 = \begin{bmatrix} 1 & 0 & 0 & 0 & -1 \\ 0 & 1/3 & 0 & 0 & 1 \\ 0 & 0 & -1/3 & 0 & -1 \\ 0 & 0 & 0 & -1 & 1 \\ -\frac{9}{16} & \frac{27}{16} & -\frac{27}{16} & \frac{9}{16} & 0 \end{bmatrix}$$

and

$$C1 = \begin{vmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2/5 & 1/3 & 2/7 \\ 0 & 0 & 0 & 1/3 & 2/7 & 1/4 \\ 0 & 0 & 0 & 2/ & & \end{vmatrix}$$

72.11 Functions of Matrices

The exponential of the matrix A is computed in the `MatrixExponential` command of Maple by *polynomial interpolation* (see Chapter 11.1) of the exponential at each of the eigenvalues of A, including multiplicities. In an exact computation context, this method is not so "dubious" [George Labahn, personal communication]. This approach is also used by the general `MatrixFunction` command.

Au: Add G. Labahn to ref. list.

Examples:

```
> A :=  Matrix( 3, 3, [[-7,-4,-3],[10,6,4],[6,3,3]] ):
> MatrixExponential( A );
```

$$\begin{bmatrix} 6 e^{1} - 7 & 4 e^{1} - 3 & 3 e^{1} - 2 \\ 10 e^{1} - 10 & 6 e^{1} - 3 & 4 e^{1} - 2 \\ 6 e^{1} - 6 & 3 e^{1} - 3 & 3 e^{1} - 2 \end{bmatrix} \tag{72.1}$$

Now a square root: `> MatrixFunction( A, sqrt(x), x ):`

$$\begin{bmatrix} 6 & 7/2 & 5/2 \\ 8 & 5 & 3 \\ 6 & 3 & 3 \end{bmatrix} \tag{72.2}$$

Another matrix square root example, for a matrix close to one that has no square root:

```
> A := Matrix( 2, 2, [[epsilon^2, 1], [0, delta^2] ] ):
> S := MatrixFunction( A, sqrt(x), x ):
> simplify( S ) assuming positive;
```

$$\begin{bmatrix} \varepsilon & \dfrac{1}{\delta + \varepsilon} \\ 0 & \delta \end{bmatrix} \tag{72.3}$$

If ε and δ both approach zero, we see that the square root has an entry that approaches infinity. Calling `MatrixFunction` on the above matrix with $\varepsilon = 0$ yields an error message, `Matrix function x^(1/2) is not defined for this Matrix`, which is correct.

Now for the matrix logarithm. `> Pascal := Matrix( 4, 4, (i,j)->binomial(j-1,i-1) );`

$$\begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & 1 & 2 & 3 \\ 0 & 0 & 1 & 3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{72.4}$$

```
> MatrixFunction( Pascal, log(x), x );
```

$$\begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 3 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{72.5}$$

Now a function not covered in Chapter 11, instead of redoing the sine and cosine examples: `> A := Matrix( 2, 2, [[-1/5, 1], [0, -1/5]] ):`

```
> W := MatrixFunction( A, LambertW(-1,x), x );
```

$$W := \begin{bmatrix} \mathrm{LambertW}(-1, -1/5) & 5\,\dfrac{\mathrm{LambertW}(-1, -1/5)}{1 + \mathrm{LambertW}(-1, -1/5)} \\ 0 & \mathrm{LambertW}(-1, -1/5) \end{bmatrix} \tag{72.6}$$

```
> evalf( W );
```

$$\begin{bmatrix} -2.542641358 & 8.241194055 \\ 0.0 & -2.542641358 \end{bmatrix} \tag{72.7}$$

That matrix satisfies $W\exp(W) = A$, and is a *primary matrix function*

Examples:

Negative of `gallery(3)` from MATLAB.

```
> A := -Matrix( [[-149,-50,-154], [537,180,546], [-27,-9,-25]] ):
> E := Matrix( [[130, -390, 0], [43, -129, 0], [133,-399,0]] ):
> AtE := A - t*E;
```

$$\begin{bmatrix} 149 + 130\,t & 50 - 390\,t & 154 \\ -537 + 43\,t & -180 + 129\,t & -546 \\ 27 + 133\,t & 9 - 399\,t & 25 \end{bmatrix} \tag{72.9}$$

For which t is that matrix stable?

```
> p :  CharacteristicPolynomial( AtE, lambda );
> PolynomialTools[Hurwitz]( p, lambda, 's', 'g' );
```

This command returns “FAIL,” meaning that it cannot tell whether p