

Maple Lecture 28. Linear Algebra

Maple has two package for linear algebra: **linalg** and **LinearAlgebra** (which is most recent). We will mainly work with the newer package **LinearAlgebra**.

```
[> with(LinearAlgebra);
```

While **LinearAlgebra** is more efficient and offers direct evaluation (not the problems with “last name evaluation”), the package **linalg** could still be useful for abstract linear algebra.

This lecture is a very appropriate transition to MATLAB (which is centered around matrices and linear algebra). As we now complete the “advanced Maple” part of the course, we use our skills in Maple freely.

For more on linear algebra and Maple, see [2, Chapter 18] and [1].

28.1 Creating Vectors and Matrices

We have seen how to make matrices with index functions. Here we see the connection with linear systems. A linear system has two representations: as a list of linear equations, or as a coefficient matrix A and a right hand side vector b . Let us generate a random linear system in matrix-vector representation:

```
[> A := RandomMatrix(3);      # coefficient matrix
[> b := RandomMatrix(3);      # right hand side vector
```

The matrix A and vector b define the linear system $Ax = b$. Before we can generate the list of equations of the system, we must define the list of variables:

```
[> X := [seq(x[i],i=1..3)];    # list of variables
[> eqs := GenerateEquations(A,X,b);
```

From a list of equations as in `eqs`, we can extract the matrix-vector representation as follows:

```
[> GenerateMatrix(eqs,X);
```

This command returns a sequence. We recognize as first element in the sequence the coefficient matrix A and on the second position we see the right hand side vector b .

28.2 Vector and Matrix Arithmetic

It is here where the operations in **LinearAlgebra** are significantly simpler than those in **linalg**. Recall that the dimensions of the matrices must agree for addition and multiplication.

```
[> A;
[> B := RandomMatrix(3);
```

We can add A to B in two ways:

```
[> A+B = MatrixAdd(A,B);
```

We could try the multiplication as

```
[> A*B;
```

but that doesn't work. Instead we work as

```
[> A.B = MatrixMatrixMultiply(A,B);
```

Observe that $A.B$ is usually not equal to $B.A$.

Note that we can stack matrices and vectors to create an augmented matrix:

```
[> <A|b>;
```

28.3 Basic Matrix Functions

In any linear algebra course, we encounter algorithms for computing a determinant, to solve linear systems, and to compute eigenvalues and eigenvectors.

```
[> Determinant(A);
```

One common way to compute the determinant is via the LU decomposition of the matrix:

```
[> P,L,U := LUdecomposition(A);
```

The command `LUdecomposition` returns three matrices: `P` is a permutation matrix, `L` is lower triangular with ones on its diagonal, and `U` is upper triangular. The following two commands illustrate the main properties of the LU decomposition:

```
[> P.A = L.U;
[> Determinant(A) = Determinant(U);
```

Another way to rewrite the matrix uses eigenvalues and eigenvectors:

```
[> Eigenvalues(A);
```

Isn't that ugly! What happened? Maple considers this matrix as exact and can for this small 3-by-3 matrix apply the exact formulas for computing the roots of a characteristic equation of degree 3. Let us force Maple to work numerically.

```
[> fA := convert(A,float);
[> lambda := Eigenvalues(fA);
[> V := Eigenvectors(fA);
```

If we ask for eigenvectors, we also get the eigenvalues as first element in the sequence `V`. An eigenvalue-eigenvector pair (λ, v) of A is characterized by the equation $Av = \lambda v$. Let us verify this property for the first eigenvector. The eigenvectors can be found in the columns of `V[2]`.

```
[> ev1 := Column(V[2],1);
[> fA.ev1 = lambda[1]*ev1;
```

We can verify this property for all eigenvectors all at once, if we put the eigenvalues on the diagonal of a diagonal matrix:

```
[> d := DiagonalMatrix(V[1]);
[> A.V[2] = V[2].d;
```

Multiplying the last equation by the inverse of `V[2]` from the right, we obtain a so-called spectral decomposition of the matrix A :

```
[> A = V[2].d.MatrixInverse(V[2]);
```

In general, we use a Jordan canonical form.

28.4 Vector Operations

From vector calculus, we remember the norm, dot and cross product. With the dot product we define orthogonality. To illustrate this, we will create an orthogonal basis for the space spanned by the columns of A (still the one from above, but we could just as well generate another random matrix). While the command `ColumnSpace` could be used, it is very instructive to proceed step by step, learning some linear algebra along the way.

We first extract the columns from A :

```
[> v := seq(Column(A,i),i=1..ColumnDimension(A));
```

Now we apply Gram-Schmidt orthogonalization on the *set* of vectors:

```
[> b := GramSchmidt({v});
```

The collection of vectors on return is orthogonal. Let us verify this with **DotProduct**:

```
[> Matrix(3,(i,j) -> DotProduct(b[i],b[j]));
```

We see that **b** is orthogonal but not orthonormal. To make **b** orthonormal, we must divide each vector by its 2-norm (and verify again):

```
[> ob := map(x->x/Norm(x,2),b);
[> Matrix(3,(i,j) -> DotProduct(ob[i],ob[j]));
```

In a linear algebra course, we have seen (or will see) that the Gram-Schmidt orthogonalization can be used to find the QR decomposition of a matrix:

```
[> Q,R := QRDecomposition(A);
```

The main property of the QR decomposition is

```
[> A = Q.R;
```

In numerical analysis we have seen (or will see) that QR decomposition is a numerically more stable alternative to solve linear systems than LU decomposition.

28.5 Smith Normal Form and SVD

While we are used to think about matrices over the real numbers, we can work with matrices over the integers. Matrices of polynomials are frequent models in control theory. For example, the discrete analogue to the Singular Value Decomposition is the Smith Normal Form.

```
[> S,U,V := SmithForm(A,output=['S','U','V']);
```

The property of the Smith normal form is verified as follows:

```
[> S = U.A.V;
```

For completeness, we show the Singular Value Decomposition:

```
[> U := 'U': S := 'S':
[> S,U,Vt := SingularValues(fA,output=['S','U','Vt']);
```

Notice that we first must free the variables **U** and **S** and use **fA**, obtained via **fA := convert(A,float)**. Then we verify the property of the decomposition as follows:

```
[> dS := DiagonalMatrix(S);
[> U.dS.Vt;
```

Modulo roundoff error, the output of the last command should coincide with the matrix **A**.

28.6 Assignments

1. The (i, j) -th entry in a Vandermonde matrix is defined as x_i^{j-1} .
Give the Maple command to make a 3-by-3 Vandermonde matrix.
2. The eigenvalues of the companion matrix of a polynomial in one variable are the roots of the polynomial. We will verify this property on a random polynomial.
 - (a) Generate a random polynomial of degree five. Use the command **CompanionMatrix** to generate the companion matrix of this polynomial.
 - (b) Compute the eigenvalues of the companion matrix and compare the vector of eigenvalues with the output of **fsolve**.
3. The Cayley-Hamilton theorem states that any matrix satisfies its own characteristic polynomial. We will verify this theorem for a random matrix.
 - (a) Generate a random 3-by-3 matrix A and use the command **CharacteristicPolynomial** to find the characteristic polynomial of A .
 - (b) To assure that we got indeed the characteristic polynomial of A , verify whether the roots of the characteristic polynomial coincide with the eigenvalues of the matrix A .
 - (c) Evaluate the characteristic polynomial in A . Make sure you obtain the zero matrix.
4. Generate a random 5-by-3 matrix, call this matrix A , and a random 5-vector, call this b .
 - (a) Compute a least squares approximation to solve $Ax = b$, using **LeastSquares**.
 - (b) Verify whether the residual vector $b - Ax$ (where x is now the solution obtained in (a)) is perpendicular to the column space of A .
5. Consider the polynomials

$$p(x) = x^4 + \alpha x^3 - \beta x^2 + 1 \quad \text{and} \quad q(x) = \alpha x^2 + \beta x + 1$$

To determine for which values of the parameters α and β the polynomials p and q have a common root, we can use a resultant, defined as the determinant of the Sylvester matrix of p and q .

Use the commands **SylvesterMatrix** and **Determinant** to find the resultant of p and q .

Compare your answer with the output of **resultant**.

References

- [1] W.C. Bauldry, B. Evans, and J. Johnson. *Linear Algebra with Maple*. John Wiley, 1995.
- [2] A. Heck. *Introduction to Maple*. Springer-Verlag, third edition, 2003.