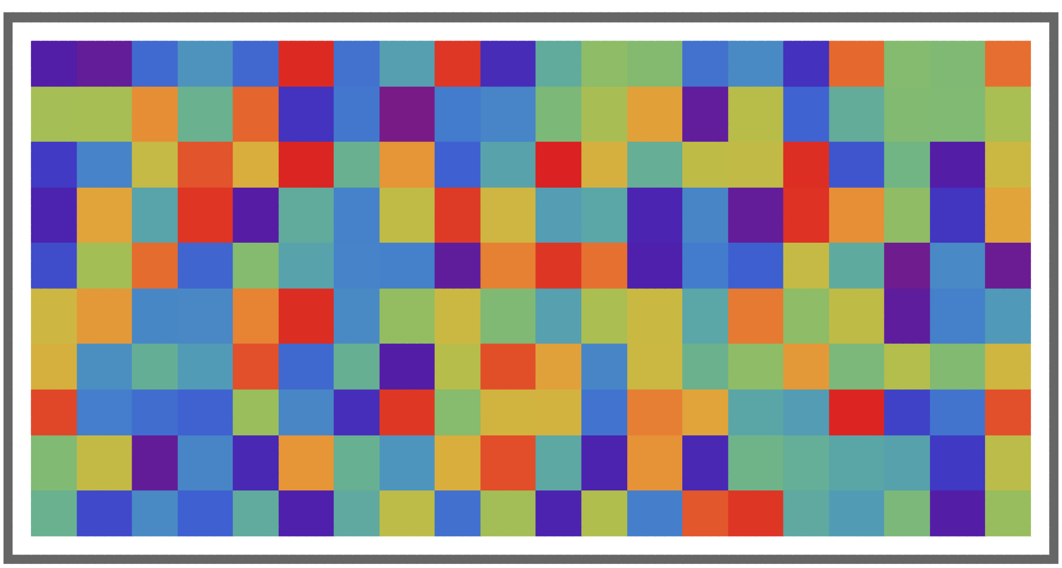
WOLFRAM U

Scientific Computing with Mathematica

PART (1): LINEAR ALGEBRA



Outline

Matrices

- Entering and Displaying Matrices
- Assembling Matrices Quickly with the **Table** Command
- Creating Special Matrices
- Concatenating Matrices
- Extracting Parts of Matrices

Operations on Matrices

- Adding and Subtracting Matrices
- Matrix Multiplication
- Matrix Power, Exponential, **Transpose**, **Trace**, Determinant and **Inverse**

Sparse Linear Algebra

- Creating Sparse Arrays
- Creating Banded Sparse Matrices

Solving Linear Systems

- The **LinearSolve** Function
- Direct and Iterative Methods
- Null Space and Rank of a Linear System

Eigensystems

Matrix Factorizations

- The QR Decomposition
- The Singular Value Decomposition
- Other Matrix Decompositions

Matrices

Entering Matrices

- To enter a matrix in Mathematica®, type a name for your matrix followed by an equal sign. Then from the menu bar select Insert ► Table/Matrix ► New.
- Select Matrix and input the matrix dimensions, then click OK.
- Fill the placeholders and use the tab key to move to the next matrix entry:

In[ ]:= $\mathcal{M} = \begin{pmatrix} 1 & 2 & 3 \\ 5 & 8 & 9 \\ 0 & 5 & 7 \end{pmatrix}$

Out[ ]= {{1, 2, 3}, {5, 8, 9}, {0, 5, 7}}

- Mathematica thinks of a matrix as a list of lists. Each row is enclosed in curly brackets with entries separated by commas, the rows are separated by commas, and the entire matrix is enclosed in curly brackets.
- You can enter a matrix in this form but it can be a little messy:

In[ ]:= $\mathcal{N}$ = {{1, 2, 3}, {5, 8, 9}, {0, 5, 7}}

Out[ ]= {{1, 2, 3}, {5, 8, 9}, {0, 5, 7}}

- A third way to enter a matrix is by using the Basic Math Assistant palette.

> *Avoid using single-letter symbols such as C, D, E, I, K and N when naming a matrix, as these letters already have a designated purpose.*
> *If you insist on using these letters for naming matrices, you can use their scripted or Greek versions.*

Matrices

Displaying Matrices

You can use the **MatrixForm** function whenever you want a nice look at your matrix:

In[ ]:=
```
A = {{0, 1}, {5, 6}};
MatrixForm[A]
```

Out[ ]//MatrixForm=

$$\begin{pmatrix} 0 & 1 \\ 5 & 6 \end{pmatrix}$$

Matrix Dimensions

The **Dimensions** command returns a list containing the number of rows and columns in the matrix, respectively:

In[ ]:=
```
Dimensions[A]
```

Out[ ]= {2, 2}

Matrices

Creating Matrices Quickly

There are several commands that produce matrices quickly:

- To get a 3×5 matrix with random integers between 0 and 10:

In[ ]:=
```
RandomInteger[10, {3, 5}] // MatrixForm
```

Out[ ]//MatrixForm=

$$\begin{pmatrix} 10 & 2 & 6 & 10 & 9 \\ 6 & 1 & 2 & 7 & 8 \\ 9 & 5 & 8 & 7 & 4 \end{pmatrix}$$

- The Table command can be used to enter matrices. The next command gives a 5×5 matrix whose i, j^{th} entry is $i + j$:

In[ ]:=
```
Table[i + j, {i, 5}, {j, 5}] // MatrixForm
```

Out[ ]//MatrixForm=

$$\begin{pmatrix} 2 & 3 & 4 & 5 & 6 \\ 3 & 4 & 5 & 6 & 7 \\ 4 & 5 & 6 & 7 & 8 \\ 5 & 6 & 7 & 8 & 9 \\ 6 & 7 & 8 & 9 & 10 \end{pmatrix}$$

- The iterators can be set to start at values other than 1:

In[ ]:=
```
Table[i + j, {i, -2, 3}, {j, 0, 2}] // MatrixForm
```

Out[ ]//MatrixForm=

$$\begin{pmatrix} -2 & -1 & 0 \\ -1 & 0 & 1 \\ 0 & 1 & 2 \\ 1 & 2 & 3 \\ 2 & 3 & 4 \\ 3 & 4 & 5 \end{pmatrix}$$

- To get a 3×4 zero matrix, you can type this:

In[ ]:=
```
Table[0, {3}, {4}] // MatrixForm
```

Out[ ]//MatrixForm=

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

- Or use the **ConstantArray** command:

In[ ]:=
```
ConstantArray[0 , {3, 4}] // MatrixForm
```

Out[ ]//MatrixForm=

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

- You can produce a 4×4 lower triangular matrix with entries on and below the diagonal equal to $i + j$:

In[ ]:=
```
Table[If[i ≥ j, i + j, 0], {i, 4}, {j, 4}] // MatrixForm
```

Out[ ]//MatrixForm=

$$\begin{pmatrix} 2 & 0 & 0 & 0 \\ 3 & 4 & 0 & 0 \\ 4 & 5 & 6 & 0 \\ 5 & 6 & 7 & 8 \end{pmatrix}$$

Matrices

Special Matrices

- The following command returns an identity matrix of the specified size:

```
In[ ]:= IdentityMatrix[5] // MatrixForm
```

Out[ ]//MatrixForm=

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

- The following command gives a diagonal matrix with the enclosed list on the diagonal:

```
In[ ]:= DiagonalMatrix[{1, 2, 3}] // MatrixForm
```

Out[ ]//MatrixForm=

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

- You can also use **DiagonalMatrix** to create a superdiagonal matrix:

```
In[ ]:= DiagonalMatrix[{1, 2, 3}, 1] // MatrixForm
```

Out[ ]//MatrixForm=

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

- The second difference matrix could be created as follows:

```
In[ ]:= n = 5;
DiagonalMatrix[Table[-2, {n}]] + DiagonalMatrix[Table[1, {n - 1}], 1] +
  DiagonalMatrix[Table[1, {n - 1}], -1]  // MatrixForm
```

Out[ ]//MatrixForm=

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

Matrices

Concatenating Matrices

A handy way to build a new matrix from existing matrices is with the command **ArrayFlatten**:

In[ ]:=
```
A = RandomInteger[5, {3, 4}];
A // MatrixForm
```

Out[ ]//MatrixForm=

$$\begin{pmatrix} 0 & 5 & 1 & 3 \\ 0 & 0 & 0 & 0 \\ 5 & 2 & 4 & 1 \end{pmatrix}$$

In[ ]:=
```
B = RandomInteger[5, {3, 4}];
B // MatrixForm
```

Out[ ]//MatrixForm=

$$\begin{pmatrix} 0 & 4 & 5 & 0 \\ 0 & 3 & 5 & 4 \\ 3 & 2 & 0 & 3 \end{pmatrix}$$

- To concatenate A and B matrices vertically:

In[ ]:=
```
ArrayFlatten[{{A}, {B}}] // MatrixForm
```

Out[ ]//MatrixForm=

$$\begin{pmatrix} 3 & 2 & 0 & 0 \\ 2 & 4 & 1 & 5 \\ 5 & 2 & 1 & 2 \\ 0 & 4 & 5 & 0 \\ 0 & 3 & 5 & 4 \\ 3 & 2 & 0 & 3 \end{pmatrix}$$

- To concatenate A and B matrices horizontally:

In[ ]:=
```
ArrayFlatten[{{A, B}}] // MatrixForm
```

Out[ ]//MatrixForm=

$$\begin{pmatrix} 3 & 2 & 0 & 0 & 0 & 4 & 5 & 0 \\ 2 & 4 & 1 & 5 & 0 & 3 & 5 & 4 \\ 5 & 2 & 1 & 2 & 3 & 2 & 0 & 3 \end{pmatrix}$$

- You can also form a block matrix:

In[ ]:=
```
ArrayFlatten[{{A , 0 , 0}, {0, B, 0}, {A, B, 0}}] // MatrixForm
```

Out[ ]//MatrixForm=

$$\begin{pmatrix} 10 & 21 & 3 & 0 & 0 & 0 & 0 & 0 \\ 4 & 5 & 6 & 0 & 0 & 0 & 0 & 0 \\ 25 & 85 & 7 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 4 & 5 & 0 & 0 \\ 0 & 0 & 0 & 0 & 3 & 5 & 4 & 0 \\ 0 & 0 & 0 & 3 & 2 & 0 & 3 & 0 \\ 10 & 21 & 3 & 0 & 4 & 5 & 0 & 0 \\ 4 & 5 & 6 & 0 & 3 & 5 & 4 & 0 \\ 25 & 85 & 7 & 3 & 2 & 0 & 3 & 0 \end{pmatrix}$$

Matrices

Extracting Parts of Matrices

Always remember that internally Mathematica thinks of a matrix as a list of lists:

In[ ]:=
```
A = {{10, 21, 3}, {4, 5, 6}, {25, 85, 7}};
A // MatrixForm
```

Out[ ]//MatrixForm=

$$\begin{pmatrix} 10 & 21 & 3 \\ 4 & 5 & 6 \\ 25 & 85 & 7 \end{pmatrix}$$

- The basic rule is that you use double square brackets to refer to individual items in a list:

In[ ]:=
```
A [[1]]
```

Out[ ]=
```
{10, 21, 3}
```

- To retrieve the entry in row 3, column 2, type:

In[ ]:=
```
A[[3, 2]]
```

Out[ ]=
```
85
```

- To extract a single column, indicate that you want **All** rows. For example, to get the second column, type:

In[ ]:=
```
A[[All, 2]]
```

Out[ ]=
```
{21, 5, 85}
```

- Use **Span**, whose infix form is **;;**, to specify a span of rows and/or columns.Here, take the first through second columns:

In[ ]:=
```
A[[All, 1 ;; 2]] // MatrixForm
```

Out[ ]//MatrixForm=

$$\begin{pmatrix} 10 & 21 \\ 4 & 5 \\ 25 & 85 \end{pmatrix}$$

Matrix Operations

Adding and Subtracting Matrices

$$A = \begin{pmatrix} 0 & 8 & 2 \\ 2 & 5 & 1 \\ 3 & 4 & 3 \end{pmatrix}; \quad B= \begin{pmatrix} 1 & 8 & 2 \\ 3 & 9 & 1 \\ 7 & 8 & 2 \end{pmatrix};$$

- The + and - operators work as expected for matrices:

In[ ]:=
```
A + B // MatrixForm
```

Out[ ]//MatrixForm=

$$\begin{pmatrix} 1 & 16 & 4 \\ 5 & 14 & 2 \\ 10 & 48 & 5 \end{pmatrix}$$

In[ ]:=
```
A - B // MatrixForm
```

Out[ ]//MatrixForm=

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

- You can also perform scalar multiplication:

In[ ]:=
```
5 * A // MatrixForm
```

Out[ ]//MatrixForm=

$$\begin{pmatrix} 0 & 40 & 10 \\ 10 & 25 & 5 \\ 15 & 20 & 15 \end{pmatrix}$$

Matrix Operations

Multiplying Matrices

$$A = \begin{pmatrix} 0 & 8 & 2 \\ 2 & 5 & 1 \\ 3 & 4 & 3 \end{pmatrix}; \quad B = \begin{pmatrix} 1 & 8 & 2 \\ 3 & 9 & 1 \\ 7 & 8 & 2 \end{pmatrix};$$

- In Mathematica, use the dot to take the product of matrices:

In[ ]:=
```
A.B // MatrixForm
```
Out[ ]//MatrixForm=

$$\begin{pmatrix} 38 & 88 & 12 \\ 24 & 69 & 11 \\ 36 & 84 & 16 \end{pmatrix}$$

*Be careful to use the dot to perform matrix multiplication. The symbol * will do elementwise multiplication.*

In[ ]:=
```
A * B // MatrixForm
```
Out[ ]//MatrixForm=

$$\begin{pmatrix} 0 & 64 & 4 \\ 6 & 45 & 1 \\ 21 & 32 & 6 \end{pmatrix}$$

Matrix Operations

Matrix Transpose

In[ ]:=
```
A = (0 8.0 2
     2 5   1
     3 4   3);
```

$$A = \begin{pmatrix} 0 & 8.0 & 2 \\ 2 & 5 & 1 \\ 3 & 4 & 3 \end{pmatrix};$$

- The Transpose command will produce the transpose of a matrix:

In[ ]:=
```
Transpose[A] // MatrixForm
```

Out[ ]//MatrixForm=

$$\begin{pmatrix} 0 & 2 & 3 \\ 8. & 5 & 4 \\ 2 & 1 & 3 \end{pmatrix}$$

Matrix Adjoint

In[ ]:=
```
ConjugateTranspose[A] // MatrixForm
```

Out[ ]//MatrixForm=

$$\begin{pmatrix} 0 & 2 & 3 \\ 8. & 5 & 4 \\ 2 & 1 & 3 \end{pmatrix}$$

Matrix Operations

Matrix Power

- To find a power of a matrix, use the command **MatrixPower**:

In[ ]:=
```
MatrixPower[A, 3] // MatrixForm
```

Out[ ]//MatrixForm=

$$\begin{pmatrix} 138. & 472. & 134. \\ 126. & 377. & 107. \\ 169. & 492. & 147. \end{pmatrix}$$

Matrix Exponential

- To find a power of a matrix, use the command **MatrixExp**:

In[ ]:=
```
MatrixExp[A] // MatrixForm
```

Out[ ]//MatrixForm=

$$\begin{pmatrix} 1424.56 & 4362.36 & 1251.52 \\ 1171.05 & 3587.79 & 1028.08 \\ 1555.42 & 4756.01 & 1370.71 \end{pmatrix}$$

Matrix Operations

Matrix Inverse

- For a nonsingular matrix M, the Inverse command will output its inverse:

In[ ]:=
```
M = RandomReal[5, {5, 5}];
M . Inverse[M];
Round[%] // MatrixForm
```

Out[ ]//MatrixForm=

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

Matrix Determinant

- To find the determinant of a matrix, use the command **Det**:

In[ ]:=
```
Det[M] // MatrixForm
```

Out[ ]//MatrixForm=

```
-133.26
```

Matrix Trace

- To find the trace of a matrix, use the command **Tr**:

In[ ]:=
```
Tr[M] // MatrixForm
```

Out[ ]//MatrixForm=

```
12.0119
```

Sparse Linear Algebra

In many applications in scientific computing (especially, numerical PDEs), arrays that arise are sparse (almost all of the entries are zero),
so it is computationally efficient to store only the nonzero entries and their positions:

```
s = SparseArray[{{1, 1} → 1, {2, 2} → 2, {3, 3} → 3, {1, 3} → 4}]
s // MatrixForm
```

Out[ ]= SparseArray[Specified elements : 4
Dimensions : {3, 3}]

Out[ ]//MatrixForm=

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

- Equivalently, you can enter a list of positions and a list of corresponding values:

In[ ]:=
```
SparseArray[{{1, 1}, {2, 2}, {3, 3}, {1, 3}} → {1, 2, 3, 4}] // MatrixForm
```

Out[ ]//MatrixForm=

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

- You can create a band diagonal matrix with the help of the **Band** command:

In[ ]:=
```
SparseArray[{Band[{1, 1}] → 1, Band[{2, 1}] → -1}, {5, 5}] // MatrixForm
```

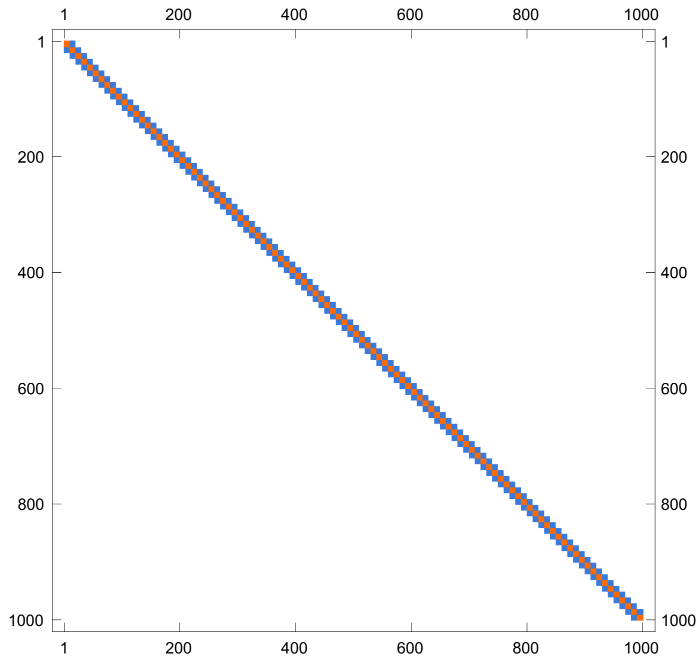
Out[ ]//MatrixForm=

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

- **MatrixPlot** can be used to visualize the sparsity pattern of large sparse matrices:

In[ ]:=
```
SparseArray[{Band[{1, 1}] → 2, Band[{2, 1}] → -1, Band[{1, 2}] → -1}, {1000, 1000}] // MatrixPlot
```

Out[ ]=



Solving Linear Systems of Equations

- The command LinearSolve provides a quick means for solving systems that have a single solution:

```
In[ ]:= A = SparseArray[{Band[{1, 1}] → 2, Band[{2, 1}] → -1, Band[{1, 2}] → -1}, {1000, 1000}];
b = ConstantArray[1.0, 1000];
x = LinearSolve[A, b];
```

- As is typical with Mathematica, the solution method is chosen automatically for you.
- With the **Method** option, you can choose different direct or iterative solution algorithms appropriate for the matrix structure (symmetric, positive definite, nonsymmetric, …)
- The Cholesky method is suitable for solving symmetric positive definite systems:

```
In[ ]:= Timing[LinearSolve[A, b, Method → "Cholesky"];]
Out[ ]= {0.006165, Null}
```

- The default method for Krylov iterative methods, BiCGSTAB, is more expensive but more generally applicable for nonsymmetric matrices:

```
In[ ]:= Timing[LinearSolve[A, b, Method → "Krylov"];]
Out[ ]= {0.000942, Null}
```

Solving Linear Systems of Equations

Null Space and Rank

Find the null space of a 3×3 matrix:

In[ ]:=
```
m = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};
NullSpace[m]
```

Out[ ]=
```
{{1, -2, 1}}
```

The action of **m** on the vector is the zero vector:

In[ ]:=
```
m . {1, -2, 1}
```

Out[ ]=
```
{0, 0, 0}
```

Find the number of linearly independent rows:

In[ ]:=
```
MatrixRank[m]
```

Out[ ]=
```
2
```

In[ ]:=
```
Length[NullSpace[m]] + MatrixRank[m] == Dimensions[m][[1]]
```

Out[ ]=
```
True
```

Eigenvalues and Eigenvectors

If you give a matrix of approximate real numbers, the Wolfram Language™ will find the approximate numerical eigenvalues and eigenvectors:

```
In[ ]:= m = {{2.3, 4.5}, {6.7, -1.2}}
Out[ ]= {{2.3, 4.5}, {6.7, -1.2}}
```

The matrix has two eigenvalues, in this case both real:

```
In[ ]:= Eigenvalues[m]
Out[ ]= {6.31303, -5.21303}
```

Here are the two eigenvectors of **m**:

```
In[ ]:= Eigenvectors[m]
Out[ ]= {{0.746335, 0.66557}, {-0.513839, 0.857886}}
```

Eigensystem computes the eigenvalues and eigenvectors at the same time. The assignment sets **vals** to the list of eigenvalues and **vecs** to the list of eigenvectors:

```
In[ ]:= {vals, vecs} = Eigensystem[m]
Out[ ]= {{6.31303, -5.21303}, {{0.746335, 0.66557}, {-0.513839, 0.857886}}}
```

Matrix Decompositions

The QR Decomposition

The QR decomposition of a matrix is a factorization of a full rank matrix into a product of an orthonormal matrix Q and a matrix R that is invertible and upper triangular.

The command **QRDecomposition** is used to compute the QR decomposition:

```
In[ ]:= {Q, R} = QRDecomposition[{{1., 2., 3.}, {4., 5., 6.}}]
Out[ ]= {{{-0.242536, -0.970143}, {-0.970143, 0.242536}},
  {{-4.12311, -5.33578, -6.54846}, {0., -0.727607, -1.45521}}}
```

```
In[ ]:= Q . R
Out[ ]= {{1., 2., 3.}, {4., 5., 6.}}
```

Matrix Decompositions

The Singular Value Decomposition SVD

A=U W Transpose[V]

- The singular value decomposition has many applications in data analysis and is a key algorithm in machine learning.
- SVD reveals a great deal of useful information about norms, rank and subspaces.

Consider a singular 3×3 matrix **m**:

```
In[ ]:= m = N[{{1, 2, 3}, {4, 5, 6}, {7, 8, 9}}];
```

Find the full singular value decomposition of **m**:

```
In[ ]:= {u, w, v} = SingularValueDecomposition [m]
Out[ ]= {{{-0.214837 , -0.887231 , 0.408248},
   {-0.520587 , -0.249644 , -0.816497}, {-0.826338 , 0.387943 , 0.408248}},
  {{16.8481 , 0. , 0.}, {0. , 1.06837 , 0.}, {0. , 0. , 0.}}, {{-0.479671 , 0.776691 , 0.408248},
   {-0.572368 , 0.0756865 , -0.816497}, {-0.665064 , -0.625318 , 0.408248}}}
```

The original matrix can be reconstructed from the singular value decomposition:

```
In[ ]:= u.w.Transpose[v]
Out[ ]= {{1. , 2. , 3.}, {4. , 5. , 6.}, {7. , 8. , 9.}}
```

Matrix Decompositions

Other Matrix Decompositions

In[ ]:= ? LUDecomposition

Symbol

Out[ ]= LUDecomposition [m] generates a representation of the LU decomposition of a square matrix m.

In[ ]:= ? SchurDecomposition

Symbol

Out[ ]= SchurDecomposition [m] yields the Schur decomposition for a numerical matrix m, given as a list $\{q, t\}$ where q is an orthonormal matrix and t is a block upper-triangular matrix.

SchurDecomposition [$\{m, a\}$] gives the generalized Schur decomposition of m with respect to a.

In[ ]:= ? JordanDecomposition

Symbol

Out[ ]= JordanDecomposition [m] yields the Jordan decomposition of a square matrix m. The result is a list $\{s, j\}$ where s is a similarity matrix and j is the Jordan canonical form of m.

References

- **Linear Algebra**
- **Sparse Linear Algebra**