

Scientific Computing Software. Maple

Lecture 4. Linear Algebra and ODE in Maple

Matrix operations and linear system solving. Symbolic solution of ODEs and PDEs. Plotting solutions and phase portraits

Nasedkina A. A.



Basic matrix and vector operations

- Definition of vectors and matrices
- Access to elements of vectors and matrices
- Determining the dimensions of vectors and matrices
- Operations with columns and rows of a matrix
- Addition and multiplication by scalar
- Matrix multiplication
- Matrix powers
- Reverse matrix, transpose and Hermitian conjugate
- Determinant, rank, trace

Defining Vectors: Standard Library Commands

How to define a vector?

- Using angle brackets:
 $\langle a_1, a_2, \dots, a_n \rangle$ is a column vector
 $\langle a_1 | a_2 | \dots | a_n \rangle$ is a row vector
- Using the **Vector** constructor command (great capabilities, (see Help)

Vector([a1,a2,...,an]) – column vector

Vector[row]([a1,a2,...,an]) – row vector

```
> v1 := <1, 2, 3>; whattype(v1)
```

$$v1 := \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

*Vector*_{column}

```
> w1 := <1|2|3>; whattype(w1)
```

$$w1 := \begin{bmatrix} 1 & 2 & 3 \end{bmatrix}$$

*Vector*_{row}

```
> v2 := Vector([4, 5, 6]); whattype(v2)
```

$$v2 := \begin{bmatrix} 4 \\ 5 \\ 6 \end{bmatrix}$$

*Vector*_{column}

```
> w2 := Vector[row]([4, 5, 6]); whattype(w2)
```

$$w2 := \begin{bmatrix} 4 & 5 & 6 \end{bmatrix}$$

*Vector*_{row}

Vector command and its arguments

Vector[o](n, init, ro, sym, sh, st, dt, f, a, o) – all arguments are optional

o – vector orientation: row (line) or column (column – default), or you can set the last parameter **o** in the form: **orientation=name**

n is the number of elements in the vector

init – the values of the vector elements can be specified by a function, procedure, list, array, etc.

ro – a Boolean expression in the form **readonly = true** or **false** , determines whether the elements of the vector can be changed

sym – specifies the symbolic values of vector elements in the form **symbol=name**

sh – specifies one or more indexing functions for the elements of a vector in the form **shape=name** or **shape=list**

st – specifies the storage method for elements in the form **storage=name** (ex: sparse)

dt – specifies the data type of elements in the form **datatype =name**

f – fills unspecified elements of a vector in the form **fill=value** , where is **value** – value of type **datatype**

a – specifies additional properties of the vector in the form **attributes=list**

Vector command: Examples

By default all Vector elements are zeros

> *Vector*(2)

$$\begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

> *Vector*(1..2, 1.25)

$$\begin{bmatrix} 1.25 \\ 1.25 \end{bmatrix}$$

> *Vector*(4, *symbol* = v, *orientation* = row)

$$\begin{bmatrix} v_1 & v_2 & v_3 & v_4 \end{bmatrix}$$

> *Vector*[row](3, *fill* = 1)

$$\begin{bmatrix} 1 & 1 & 1 \end{bmatrix}$$

> *v* := *Vector*(4, [5, 8]) + *Vector*(4, *fill* = 10)

$$v := \begin{bmatrix} 15 \\ 18 \\ 10 \\ 10 \end{bmatrix}$$

> *s* := { 1 = 28, 2 = -I + 1 } : *Vector*(2, *s*)

$$\begin{bmatrix} 28 \\ 1 - I \end{bmatrix}$$

Vector elements are generated using a function

> *f* := *j* → *x*^{*j*} : *Vector*(3, *f*)

$$\begin{bmatrix} x \\ x^2 \\ x^3 \end{bmatrix}$$

> *Vector*(3, *shape* = *scalar*[2, 100])

$$\begin{bmatrix} 0 \\ 100 \\ 0 \end{bmatrix}$$

Read-only Vector

> *v* := *Vector*[row]([1, 2, 3], *readonly* = true) :

> *v*[1] := 10

Error, cannot assign to a read-only Vector

Defining Matrices: Standard Library Commands

How to define a matrix?

- Using angle brackets:
 $\langle\langle a_{11}, \dots, a_{1n} \rangle | \langle a_{21}, \dots, a_{2n} \rangle | \dots | \langle a_{m1}, \dots, a_{mn} \rangle \rangle$ – matrix of size $n \times m$,
 given by columns
 $\langle\langle a_{11} \ 1 | \dots | a_{1n} \ m \rangle, \langle a_{21} \ 1 | \dots | a_{2n} \ m \rangle, \dots, \langle a_{m1} \ 1 | \dots | a_{mn} \ m \rangle \rangle$ – size matrix
 $n \times m$, specified by lines
- Matrix** constructor command (great possibilities,
 (see Help)

Matrix([[a11,a12,...,a1m],[a21,a22,...,a2m], ..., [an1,an2,..., anm]])

```
> A := <<1, 2, 3>|<4, 5, 6>|<7, 8, 10>|<11, 12, 13>>;
whattype(A)
```

$$A := \begin{bmatrix} 1 & 4 & 7 & 11 \\ 2 & 5 & 8 & 12 \\ 3 & 6 & 10 & 13 \end{bmatrix}$$

Matrix

```
> B := <<1|2|3>, <4|5|6>>; whattype(B)
```

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

Matrix

```
> M := Matrix([ [1, 2], [3, 4] ])
```

$$M := \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

Matrix command and its arguments

Matrix(r, c, init, ro, sym, sc, sh, st, o, dt, f, a) – all arguments are optional
r is the number of matrix rows

c – number of matrix columns

init – the values of the matrix elements can be specified by a function, procedure, list, matrix, vector, etc.

ro – a Boolean expression in the form **readonly = true** or **false**, determines whether the matrix elements can be changed

sym – specifies the symbolic values of the matrix elements in the form **symbol=name**

sc – determines the method of filling the matrix with elements **init** in the form **scan=name** or **scan=list** (for example, **scan=columns** – filling by columns)

sh – specifies one or more indexing functions for matrix elements in the form **shape=name** or **shape=list** (e.g. diagonal, triangular[upper], triangular[lower], symmetric, identity, etc.)

st – specifies the storage method for elements in the form **storage=name** (sparse, diagonal, etc.)

o – specifies the method for storing the matrix structure: **order= C_order** (row-based storage) or **order= Fortran_order** (column-based storage – default)

dt – specifies the data type of elements in the form **datatype =name**

f – fills unspecified matrix elements in the form **fill=value**, where **value** – value of type **datatype**

a – specifies additional properties of the matrix in the form **attributes=list**

Matrix command: Examples

Square matrix 2 by 2 with all zero elements

> *Matrix*(2)

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

Specify matrix size and elements

> *Matrix*(1..2, 1..3, 50)

$$\begin{bmatrix} 50 & 50 & 50 \\ 50 & 50 & 50 \end{bmatrix}$$

> *Matrix*(3, 2, [[1, 2], [3, 4]])

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

> *A* := *Matrix*(3, 2, [1, 2, 3, 4, 5, 6])

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

> *Matrix*(4, 3, *A*, *fill* = 87)

$$\begin{bmatrix} 1 & 2 & 87 \\ 3 & 4 & 87 \\ 5 & 6 & 87 \\ 87 & 87 & 87 \end{bmatrix}$$

> *Matrix*(2, 3, *symbol* = *m*)

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

> *s* := {(1, 1) = 55, (1, 2) = 66, (2, 1) = 77};

Matrix(2, 3, *s*)

s := {(1, 1) = 55, (1, 2) = 66, (2, 1) = 77}

$$\begin{bmatrix} 55 & 66 & 0 \\ 77 & 0 & 0 \end{bmatrix}$$

Specify matrix shape (symmetric)

> *Matrix*(3, {(1, 1) = 50, (1, 2) = 60}, *fill* = 1, *shape* = *symmetric*)

$$\begin{bmatrix} 50 & 60 & 1 \\ 60 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Generate identity matrix

> *Matrix*(3, *shape* = *identity*)

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

Upper triangular matrix with fill option

> *Matrix*(3, *fill* = 1, *shape* = *triangular*)

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

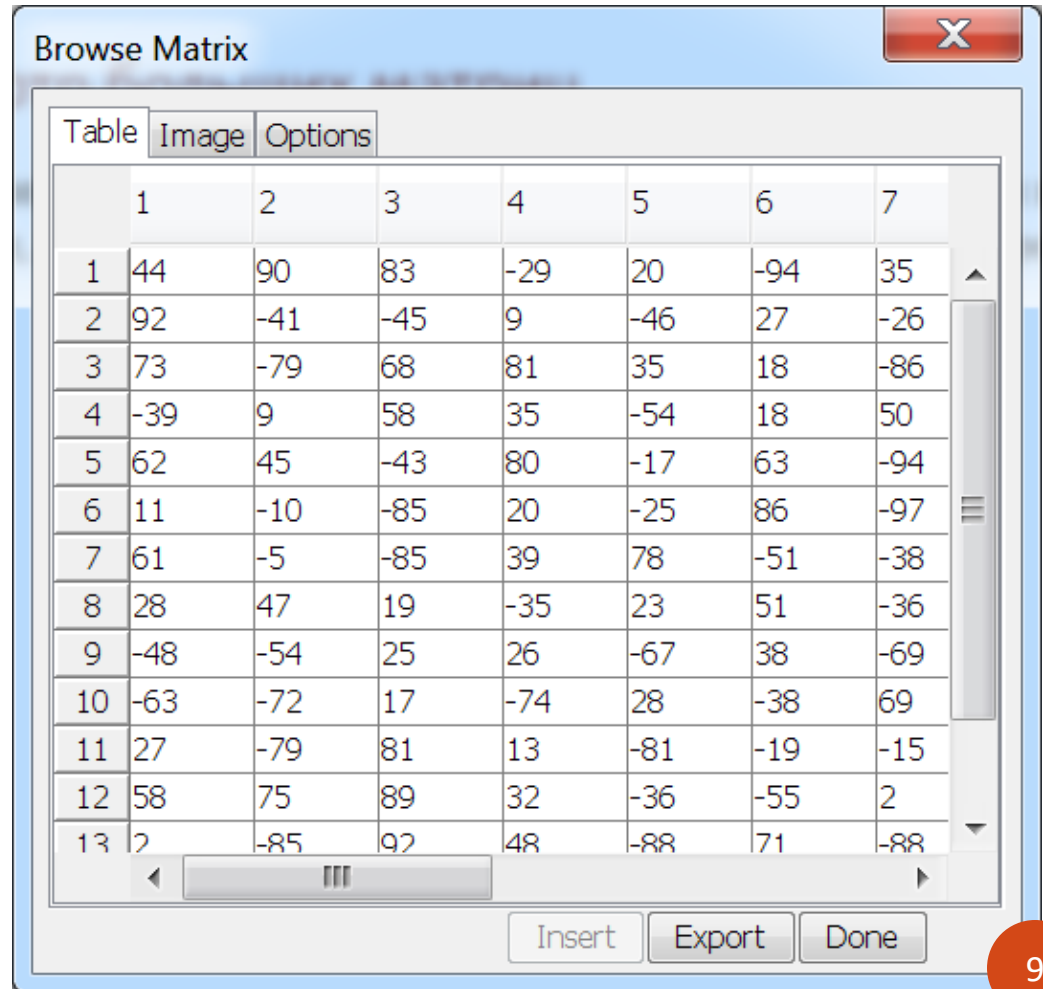
Viewing large matrices

- Matrices and vectors larger than 10 are not displayed on the screen; elements can be viewed in the browser.

```
> with(LinearAlgebra) :  
> RandomMatrix(15)
```

15 x 15 Matrix
Data Type: anything
Storage: rectangular
Order: Fortran_order

Double-click



	1	2	3	4	5	6	7
1	44	90	83	-29	20	-94	35
2	92	-41	-45	9	-46	27	-26
3	73	-79	68	81	35	18	-86
4	-39	9	58	35	-54	18	50
5	62	45	-43	80	-17	63	-94
6	11	-10	-85	20	-25	86	-97
7	61	-5	-85	39	78	-51	-38
8	28	47	19	-35	23	51	-36
9	-48	-54	25	26	-67	38	-69
10	-63	-72	17	-74	28	-38	69
11	27	-79	81	13	-81	-19	-15
12	58	75	89	32	-36	-55	2
13	2	-85	92	48	-88	71	-88

LinearAlgebra package

Use **LinearAlgebra** package to work with matrices and vectors, most of the commands for solving linear algebra problems are inside this package

Use the following syntax to use add the package:

- **LinearAlgebra[command](arguments)**
- **with(LinearAlgebra): command(arguments)**

> *with(LinearAlgebra)*

[&x, Add, Adjoint, BackwardSubstitute, BandMatrix, Basis, BezoutMatrix, BidiagonalForm, BilinearForm, CharacteristicMatrix, CharacteristicPolynomial, Column, ColumnDimension, ColumnOperation, ColumnSpace, CompanionMatrix, ConditionNumber, ConstantMatrix, ConstantVector, Copy, CreatePermutation, CrossProduct, DeleteColumn, DeleteRow, Determinant, Diagonal, DiagonalMatrix, Dimension, Dimensions, DotProduct, EigenConditionNumbers, Eigenvalues, Eigenvectors, Equal, ForwardSubstitute, FrobeniusForm, GaussianElimination, GenerateEquations, GenerateMatrix, Generic, GetResultDataType, GetResultShape, GivensRotationMatrix, GramSchmidt, HankelMatrix, HermiteForm, HermitianTranspose, HessenbergForm, HilbertMatrix, HouseholderMatrix, IdentityMatrix, IntersectionBasis, IsDefinite, IsOrthogonal, IsSimilar, IsUnitary, JordanBlockMatrix, JordanForm, LA_Main, LUdecomposition, LeastSquares, LinearSolve, Map, Map2, MatrixAdd, MatrixExponential, MatrixFunction, MatrixInverse, MatrixMatrixMultiply, MatrixNorm, MatrixPower, MatrixScalarMultiply, MatrixVectorMultiply, MinimalPolynomial, Minor, Modular, Multiply, NoUserValue, Norm, Normalize, NullSpace, OuterProductMatrix, Permanent, Pivot, PopovForm, QRdecomposition, RandomMatrix, RandomVector, Rank, RationalCanonicalForm, ReducedRowEchelonForm, Row, RowDimension, RowOperation, RowSpace, ScalarMatrix, ScalarMultiply, ScalarVector, SchurForm, SingularValues, SmithForm, StronglyConnectedBlocks, SubMatrix, SubVector, SumBasis, SylvesterMatrix, ToeplitzMatrix, Trace, Transpose, TridiagonalForm, UnitVector, VandermondeMatrix, VectorAdd, VectorAngle, VectorMatrixMultiply, VectorNorm, VectorScalarMultiply, ZeroMatrix, ZeroVector, Zip]

Constructor commands for defining matrices and vectors of a special type

```
> with(LinearAlgebra) :
```

Matrix with zero values

```
> ZeroMatrix(3)
```

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

Vector with zero values

```
> ZeroVector(2)
```

$$\begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

UnitVector(i, d)

Unit Vector is the vector with d entries, where all entries are zero except one with 1 at i-th position

```
> UnitVector(2, 3)
```

$$\begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$$

```
> IdentityMatrix(2)
```

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

Diagonal matrix with vector [a,b,c] on the main diagonal

```
> DiagonalMatrix([a, b, c])
```

$$\begin{bmatrix} a & 0 & 0 \\ 0 & b & 0 \\ 0 & 0 & c \end{bmatrix}$$

Random matrix with integer values

```
> RandomMatrix(3)
```

$$\begin{bmatrix} 27 & 99 & 92 \\ 8 & 29 & -31 \\ 69 & 44 & 67 \end{bmatrix}$$

Integer range: -99..99

Changing the range – option generator= a..b (not necessarily an integer) range, see Help)

Access to elements of vectors and matrices

```
> v := <23.5, -1.7, 8.2>
```

$$v := \begin{bmatrix} 23.5 \\ -1.7 \\ 8.2 \end{bmatrix}$$

Submatrix with specified rows (1st argument) and columns (2nd argument)

```
> M[1 ..2, 2 ..3]
```

$$\begin{bmatrix} 5 & 6 \\ -8 & -9 \end{bmatrix}$$

```
> v[1]
```

23.5

Subvector

```
> v[1 ..2]
```

$$\begin{bmatrix} 23.5 \\ -1.7 \end{bmatrix}$$

```
> v[-1]
```

8.2

```
> M := \begin{bmatrix} 4 & 5 & 6 \\ -7 & -8 & -9 \\ 101 & 102 & 103 \end{bmatrix} :
```

```
> M[2, 1]
```

-

Finding out the dimensions of vectors and matrices

LinearAlgebra :

- **Dimension(A)** – the dimension of a matrix or vector
- **RowDimension (A)** – the number of matrix rows
- **ColumnDimension (A)** – the number of columns of the matrix

```
> with(LinearAlgebra) :  
> M := RandomMatrix(2, 3)  

$$M := \begin{bmatrix} 33 & -77 & 27 \\ -98 & 57 & -93 \end{bmatrix}$$
  
> Dimension(M)  
2, 3  
> v := Vector([5, 6, 7]) : Dimension(v)  
3
```

```
> A := Matrix([[1, 2, 3], [4, 5, 6]])  

$$A := \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$
  
> RowDimension(A)  
2  
> ColumnDimension(A)  
3
```

Matrix Row and Column Operations: Deleting Rows and Columns

LinearAlgebra :

- **DeleteRow (A, L, outopts)** – deleting rows of matrix A
- **DeleteColumn (A, L, outopts)** – deleting columns of matrix A

L – numbers of rows (columns) to be deleted, can be in the form of an interval or a list
outopts - (optional parameter) **output** options for the resulting object, for example
outputoptions = [datatype = float, shape = ...]

```
> with(LinearAlgebra) :  
> A := <<1, 2, 3><4, 5, 6><7, 8, 9>>  
A :=  $\begin{bmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{bmatrix}$   
> DeleteRow(A, 2)  
 $\begin{bmatrix} 1 & 4 & 7 \\ 3 & 6 & 9 \end{bmatrix}$ 
```

```
> DeleteColumn(A, 2..3)  
 $\begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$   
> DeleteRow(A, [1, 3])  
 $\begin{bmatrix} 2 & 5 & 8 \end{bmatrix}$ 
```

Matrix Column and Row Operations: Extracting Rows and Columns

LinearAlgebra :

- **Row(A, L, outopts)** – extract rows of matrix A
- **Column(A, L, outopts)** – extract columns of matrix A

L – numbers of the rows (columns) to be extracted, can be in the form of an interval or a list

outopts - (optional parameter) **output** options for the resulting object, for example
outputoptions = [datatype = float, shape = ...]

```
> with(LinearAlgebra) :  
> A := RandomMatrix(4)  
  
A :=  $\begin{bmatrix} 12 & -16 & 45 & 87 \\ -2 & -9 & -81 & 33 \\ 50 & -50 & -38 & -98 \\ 10 & -22 & -18 & -77 \end{bmatrix}$ 
```

```
> r2 := Row(A, 2); whattype(r2)  
r2 :=  $\begin{bmatrix} -2 & -9 & -81 & 33 \end{bmatrix}$   
Vector_row  
  
> c3 := Column(A, 3); whattype(c3)  
c3 :=  $\begin{bmatrix} 45 \\ -81 \\ -38 \\ -18 \end{bmatrix}$   
Vector_column
```

```
> B := Row(A, 1..3);  
whattype(B[1])  
B :=  $\begin{bmatrix} 12 & -16 & 45 & 87 \\ -2 & -9 & -81 & 33 \\ 50 & -50 & -38 & -98 \end{bmatrix}$   
Vector_row
```

Operations with columns and rows of a matrix: elementary permutation of rows and columns

LinearAlgebra :

RowOperation (A, K, s, ip , outopts) – operations with rows of matrix A

ColumnOperation (A, K, s, ip , outopts) – operations with columns of matrix A

A – matrix ; K is an integer or a list of two integers ; s is an algebraic expression ; ip – an expression of the form inplace =true/false (optional), determines whether to modify the matrix A; outopts – an expression of the form outputoptions =list; inplace and outputoptions are mutually exclusive

- **RowOperation (A, [ri,rj])** – permutation of two strings **ri** And **rj** of matrix A
- **ColumnOperation (A, [ci,cj])** – permutation of two columns **ci** And **cj** of matrix A

```
> with(LinearAlgebra) :  
> A := RandomMatrix(3)  
A :=  $\begin{bmatrix} 86 & -48 & 31 \\ 20 & 77 & -50 \\ -61 & 9 & -80 \end{bmatrix}$ 
```

```
> RowOperation(A, [1, 3])  
 $\begin{bmatrix} -61 & 9 & -80 \\ 20 & 77 & -50 \\ 86 & -48 & 31 \end{bmatrix}$ 
```

```
> ColumnOperation(A, [2, 3])  
 $\begin{bmatrix} 86 & 31 & -48 \\ 20 & -50 & 77 \\ -61 & -80 & 9 \end{bmatrix}$ 
```

Operations with columns and rows of a matrix: addition and multiplication of rows and columns

- **RowOperation (A, [ri,rj], expr)** – change the line **ri** : **ri := ri+rj * expr** , where **expr** is a number or expression (analogous to **addrow***) , addition of strings
- **ColumnOperation (A, [ci,cj], expr)** – change column **ci** : **ci := ci+cj * expr** , where **expr** is a number or expression (similar to **addcol**) , addition of columns
- **RowOperation (A, r, expr)** – row multiplication **r** on the expression **expr**:
r:=r* expr
- **ColumnOperation (A, c, expr)** – multiplication of column **c** on the expression **expr**:
c:=c* expr

```
> with(LinearAlgebra) :  
> A := Matrix( [[1, 2, 3], [4, 5, 6], [7, 8, 9]])
```

$$A := \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

```
> RowOperation(A, [1, 3], 100)
```

$$\begin{bmatrix} 701 & 802 & 903 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

```
> ColumnOperation(A, [3, 2], 10)
```

$$\begin{bmatrix} 1 & 2 & 23 \\ 4 & 5 & 56 \\ 7 & 8 & 89 \end{bmatrix}$$

Addition and multiplication by a number

LinearAlgebra :

- **A+B** – matrix addition or vectors **A** And **B**
- **A*c** – multiplication of the elements of the matrix (vector) **A** by the scalar **c**
- Commands: **Add(A,B)** or **MatrixAdd (A,B)** and **Multiply(A,c)**

```
> with(LinearAlgebra) : A := Matrix([[1, 2], [3, 4]]);  
  B := Matrix(2, [10, 20, 30, 40])
```

$$A := \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

$$B := \begin{bmatrix} 10 & 20 \\ 30 & 40 \end{bmatrix}$$

```
> A + B; A · 100
```

$$\begin{bmatrix} 11 & 22 \\ 33 & 44 \end{bmatrix}$$

$$\begin{bmatrix} 100 & 200 \\ 300 & 400 \end{bmatrix}$$

```
> v := <1, 2, 3> : 10 · v
```

$$\begin{bmatrix} 10 \\ 20 \\ 30 \end{bmatrix}$$

Matrix and matrix-vector multiplication

LinearAlgebra :

- **A.B** – matrix (non-commutative) multiplication of matrices and vectors
- **MatrixVectorMultiply (A, u)** – multiplication of matrix **A** by vector **u**
- **MatrixMatrixMultiply (A, B)** – multiplication of matrix **A** by matrix **B**
- General command: **Multiply(A,B)**

```
> with(LinearAlgebra) : A := Matrix([[1, 2], [3, 4]]);  
  B := Matrix(2, [10, 20, 30, 40])
```

$$A := \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

$$B := \begin{bmatrix} 10 & 20 \\ 30 & 40 \end{bmatrix}$$

```
> A.B;
```

$$\begin{bmatrix} 70 & 100 \\ 150 & 220 \end{bmatrix}$$

```
> v := <100, 150> : A.v;
```

$$\begin{bmatrix} 400 \\ 900 \end{bmatrix}$$

```
> MatrixMatrixMultiply(A, B)
```

$$\begin{bmatrix} 70 & 100 \\ 150 & 220 \end{bmatrix}$$

```
> MatrixVectorMultiply(A, v)
```

$$\begin{bmatrix} 400 \\ 900 \end{bmatrix}$$

Matrix powers and inverse

LinearAlgebra :

- A^n – construction of a square matrix A to the power n
- A^{-1} – calculation of the inverse matrix. If inverse doesn't exist (determinant is zero), then the matrix is called *singular*
- **MatrixInverse (A).** Full syntax : **MatrixInverse (A, m, mopts , c, out, outopts)** – see Help

```
> with(LinearAlgebra) :  
> A := Matrix([ [1, 2], [3, 4] ])  

$$A := \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$
  
> MatrixInverse(A)  

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

```

```
Example of matrix with linear dependent columns  
> C := Matrix([ [3, 5], [6, 10] ], scan = columns)  

$$C := \begin{bmatrix} 3 & 6 \\ 5 & 10 \end{bmatrix}$$
  
Such matrix is singular (inverse does not exist)  
> MatrixInverse(C)  
Error, (in LinearAlgebra:-LA_Main:-  
MatrixInverse) singular matrix
```

Calculating the transpose and Hermitian conjugate transpose of a matrix

LinearAlgebra :

- $A^{\wedge}(\%T)$ – transposed matrix A^T

Transpose(A)

- $A^{\wedge}(\%H)$ is a Hermitian conjugate matrix $A^H = \overline{A^T}$ (operation of transposition and complex conjugation of elements), command:

HermitianTranspose (A)

If $A = A^T$, the matrix is called *symmetric* (*nonsymmetric* otherwise).

If $A = A^H$, the matrix is called *Hermitian*.

```
> restart; with(LinearAlgebra):
> A := Matrix([[1, 2], [3, 4]]);
      A :=  $\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$ 
> A%^T; Transpose(A)
       $\begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix}$ 
       $\begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix}$ 

> B := RandomMatrix(3, 4)
      B :=  $\begin{bmatrix} -32 & 27 & 99 & 92 \\ -74 & 8 & 29 & -31 \\ -4 & 69 & 44 & 67 \end{bmatrix}$ 
> B%^T
       $\begin{bmatrix} -32 & -74 & -4 \\ 27 & 8 & 69 \\ 99 & 29 & 44 \\ 92 & -31 & 67 \end{bmatrix}$ 

> W := Matrix([[1 + 2*I, I], [3 - 4*I, -I]])
      W :=  $\begin{bmatrix} 1 + 2I & I \\ 3 - 4I & -I \end{bmatrix}$ 
> W%^T;
       $\begin{bmatrix} 1 + 2I & 3 - 4I \\ I & -I \end{bmatrix}$ 
> W%^H; HermitianTranspose(W)
       $\begin{bmatrix} 1 - 2I & 3 + 4I \\ -I & I \end{bmatrix}$ 
       $\begin{bmatrix} 1 - 2I & 3 + 4I \\ -I & I \end{bmatrix}$ 
```

Determinant, rank and trace of a matrix

LinearAlgebra :

- **Determinant(A)** – calculation of the determinant of the matrix **A**
- **Rank(A)** – rank of the matrix
- **Trace(A)** – trace of the matrix (sum of diagonal elements)

```
> with(LinearAlgebra) : A := Matrix([ [1, 2], [3, 4] ] );
```

$$A := \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

```
> Determinant(A)
```

-2

```
> Rank(A)
```

2

```
> Trace(A)
```

5

Solving linear algebra problems

- Norms of matrices and vectors
- Checking the equality of two matrices
- Spectral analysis: eigenvalues and eigenvectors, characteristic polynomial
- Finding out the type of matrix (positive / negative definiteness ; orthogonality and unitarity)
- Solving systems of linear equations
- Matrix factorizations: QR and LU decomposition

Vector norms

LinearAlgebra :

- **Norm(A, p, c)** – p- norm of a matrix or vector
- **VectorNorm (A, p, c)** – p is the norm of the vector, c is (optional) options for the resulting object

Values of the p parameter (default = infinity)

➤ for vectors

- **2, Euclidean , Frobenius** – Euclidean norm $\|x\|_2 = \sqrt{\sum_{i=1}^n |x_i|^2} = \sqrt{x \cdot x}$
(vector length)
- **p (>0)** – p- norm

```
VectorNorm ( A,p )= add( abs(V[i])^p,  
i = 1 .. Dimension(V)) ^ (1/p)
```

- **infinity** – the maximum element by modulus

Matrix norms

LinearAlgebra :

- **Norm(A, p, c)** – p-norm of a matrix or vector
- **MatrixNorm (A, p, c)** – p-norm of the matrix, c – (optional) options for the resulting object

Values of the p parameter (default = infinity)

➤ for matrices

- **1** – maximum column norm

$$\|A\|_1 = \max_{j=1,m} \sum_{i=1}^n |a_{ij}| \quad \forall A \in C^{n \times m}$$

`MatrixNorm (A, 1) = max( seq ( VectorNorm (A[1..-1, j], 1), j = 1 .. ColumnDimension (A)))`

- **infinity** – maximum row norm

$$\|A\|_{\infty} = \max_{i=1,n} \sum_{j=1}^m |a_{ij}| \quad \forall A \in C^{n \times m}$$

`MatrixNorm(A, infinity) = max(seq(VectorNorm(A[i, 1..-1], 1), i = 1 .. RowDimension(A)))`

Matrix Norms (continued)

Values of the p parameter (default = infinity)

- **2 or Euclidean** – *spectral* norm (the root of the maximum absolute value of the matrix eigenvalues $A^H A$)

$$\|A\|_2 = \sqrt{\max_i |\lambda_i(A^H A)|} = \sqrt{\max_i |\lambda_i(AA^H)|} \quad \forall A \in \mathbb{C}^{n \times m}, \quad A^H = \overline{A^T}$$

```
MatrixNorm (A, 2) = sqrt (max ( seq (Eigenvalues (A.  
A^ (%H)) [ i ], i = 1 .. RowDimension (A))))
```

- **Frobenius** – Frobenius norm $\|A\|_F = \sqrt{\sum_{j=1}^m \sum_{i=1}^n |a_{ij}|^2} \quad \forall A \in \mathbb{C}^{n \times m}$

(not a true matrix norm, since it is not related to vector norms)

```
MatrixNorm (A, Frobenius ) = sqrt (add(add(abs (A[ i, j  
])^2, j = 1 .. ColumnDimension (A)), i = 1 ..  
RowDimension (A)))
```

Examples of calculating norms of vectors and matrices

```
> with(LinearAlgebra) :
```

```
> v := Vector([1, -2, 3])
```

$$v := \begin{bmatrix} 1 \\ -2 \\ 3 \end{bmatrix}$$

By default: infinity-norm for vectors (entry with maximum absolute value)

```
> VectorNorm(v)
```

3

1-norm (sum of absolute values)

```
> VectorNorm(v, 1)
```

6

Euclidean norm

```
> VectorNorm(v, 2)
```

$$\sqrt{14}$$

```
> VectorNorm(v, Euclidean)
```

$$\sqrt{14}$$

```
> VectorNorm(v, Frobenius)
```

$$\sqrt{14}$$

3-norm (the cube root of the sum of the moduli of the cubes of the elements)

```
> VectorNorm(v, 3)
```

$$36^{1/3}$$

infinity-norm (maximum absolute value)

```
> VectorNorm(v, infinity)
```

3

```
> A := Matrix([[10, 0, 2], [0, 9, 1], [2, 4, 1]])
```

$$A := \begin{bmatrix} 10 & 0 & 2 \\ 0 & 9 & 1 \\ 2 & 4 & 1 \end{bmatrix}$$

By default: infinity-norm for matrices (maximum absolute row sum)

```
> MatrixNorm(A)
```

12

1-norm: maximum absolute column sum

```
> MatrixNorm(A, 1)
```

13

Spectral norm (maximum absolute eigenvalue)

```
> MatrixNorm(A, 2)
```

$$\sqrt{\text{RootOf}(-196 + _Z^3 - 207 _Z^2 + 10577 _Z, \text{index} = 3)}$$

Often can be calculated as approximate value

```
> evalf(%)
```

10.73324841

Euclidean matrix norm = spectral norm

```
> MatrixNorm(A, Euclidean)
```

$$\sqrt{\text{RootOf}(-196 + _Z^3 - 207 _Z^2 + 10577 _Z, \text{index} = 3)}$$

Frobenius norm (not true matrix norm)

```
> MatrixNorm(A, Frobenius)
```

$$3\sqrt{23}$$

infinity-norm (maximum absolute row sum)

```
> MatrixNorm(A, infinity)
```

12

Checking the equality of two matrices

LinearAlgebra :

- **Equal(A,B)** – checks the logical equality of matrices A and B, the result is true/false
- **MatrixNorm (AB,1)** or **MatrixNorm (A- B, infinity)** – checking the equality of matrices by norm, for approximate values of coefficients

If $A \simeq B$, then $A-B$ is a zero matrix whose norm is close to zero.

```
> with(LinearAlgebra) :  
> A := RandomMatrix(3)  
  
A :=  $\begin{bmatrix} 27 & 99 & 92 \\ 8 & 29 & -31 \\ 69 & 44 & 67 \end{bmatrix}$   
  
Inverse matrix  
> invA := MatrixInverse(A)  
  
invA :=  $\begin{bmatrix} -\frac{3307}{327244} & \frac{2585}{327244} & \frac{5737}{327244} \\ \frac{2675}{327244} & \frac{4539}{327244} & -\frac{1573}{327244} \\ \frac{1649}{327244} & -\frac{5643}{327244} & \frac{9}{327244} \end{bmatrix}$ 
```

```
Check that  $A \cdot A^{-1} = E$  (identity matrix)  
> A.invA  
  
 $\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$   
  
> E := IdentityMatrix(3)  
  
 $E := \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$   
  
Use Equal command to check equality  $A \cdot A^{-1} = E$   
> Equal(A.invA, E)  
  
true
```

Checking the equality of two matrices: example for approximate values

```
> with(LinearAlgebra) :
> A := RandomMatrix(3)
```

$$A := \begin{bmatrix} 27 & 99 & 92 \\ 8 & 29 & -31 \\ 69 & 44 & 67 \end{bmatrix}$$

```
> E := IdentityMatrix(3)
```

$$E := \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

```
Inverse matrix with approximate values
> invA1 := evalf(invA)
```

$$\text{invA1} := \begin{bmatrix} -0.01010560927 & 0.007899304494 & 0.01753126108 \\ 0.008174328636 & 0.01387038418 & -0.004806810820 \\ 0.005039053428 & -0.01724401364 & 0.00002750241410 \end{bmatrix}$$

Check that $A \cdot A^{-1} = E$ (identity matrix)

```
> A.invA1
```

$$\begin{bmatrix} 1.00000000005000000 & 2.78000057002403268 \cdot 10^{-10} & 7.71999649536864175 \cdot 10^{-11} \\ 1.60000259569192949 \cdot 10^{-11} & 1.00000000001199996 & 2.28999896844696294 \cdot 10^{-11} \\ 3.00000597280858372 \cdot 10^{-11} & 1.26000026384742726 \cdot 10^{-10} & 1.00000000018470004 \end{bmatrix}$$

Use Equal command to check equality: $A \cdot A^{-1} = E$

```
> Equal(A.invA1, E)
```

false

Gives false because equality is approximate, use comparison by matrix norm instead

```
> MatrixNorm(A.invA1 - E, 1)
```

$4.16000039971109458 \cdot 10^{-10}$

```
> MatrixNorm(A.invA1 - E, infinity)
```

$4.05200026093108234 \cdot 10^{-10}$

Spectral Analysis: General Information on Eigenvalues and Eigenvectors

Definitions from a linear algebra course

Let A be a square matrix of size $n \times n$, in general with complex values.

- If $A\mathbf{u} = \lambda\mathbf{u}$, then the vector $\mathbf{u}$ is called an **eigenvector** of the matrix A , and the number λ is called an **eigenvalue** corresponding to the given eigenvector.
- The set of all eigenvalues of a matrix is called the **spectrum** of the matrix.
- For a matrix of size $n \times n$, the total number of eigenvalues is n .
- If the same eigenvalue occurs k times in the spectrum of a matrix, then we say that this eigenvalue is **multiple** and its **(algebraic) multiplicity** is equal to k .
- Eigenvalues of the matrix A are the roots of the **characteristic polynomial** $P_A(\lambda) = \det(A - \lambda E)$, where E is the identity matrix.
- The algebraic multiplicity of an eigenvalue λ is its multiplicity as a root of characteristic polynomial.
- Matrix $A - \lambda E$ is called the **characteristic matrix** for the matrix A

LinearAlgebra package commands :

- **Eigenvalues(A)** – returns the eigenvalues of the matrix A
- **Eigenvectors(A)** – returns sequence with two arguments: the eigenvalues of matrix A as a column vector and a matrix of eigenvectors (column-wise)
- **CharacteristicPolynomial (A, lambda)** – characteristic polynomial
- **CharacteristicMatrix (A, lambda, outopts)** – characteristic matrix

Spectral Analysis: Eigenvalues

- **Eigenvalues(A, C, imp, o, outopts)** – the remaining arguments are optional
C is the matrix for solving the generalized eigenvalue problem, i.e. the roots of the equation $\det(\lambda * C - A)$ are found
imp – the type of roots of the characteristic polynomial; if implicit = **true** or simply **implicit**, then the calculation will be implicit RootOf
o – sets the output format of the result in the form: **output = obj**, where **obj** can take the values '**Vector[row]**', '**Vector[column]**', '**list**' or a list of these values
outopts – sets **output** options constructor for the resulting object, for example
outoptions = [datatype= float, shape = ...]
- **Eigenvectors(A)** – returns the eigenvalues of matrix **A** as a column vector and a matrix of column-wise eigenvectors. The full syntax is:
- **Eigenvectors(A, C, imp, o, outopts)** – the remaining arguments are optional
C is the matrix for solving the generalized eigenvalue problem, i.e., the equations $\det(\lambda * C - A)$ are found
imp – specifies the form in which the roots of the characteristic polynomial will be calculated; if implicit = **true** or simply **implicit**, then the calculation will be implicit RootOf
o – sets the output format of the result in the form: **output = obj**, where **obj** can take the values '**values**', '**vectors**', '**list**' or a list of these values

Examples of calculating eigenvalues: Eigenvalues

```
> with(LinearAlgebra) :
> A := Matrix(3, 3, [x, 0, y, x, y, 0, y, 0, x])
```

$$A := \begin{bmatrix} x & 0 & y \\ x & y & 0 \\ y & 0 & x \end{bmatrix}$$

Matrix eigenvalues as a column vector

```
> Eigenvalues(A)
```

$$\begin{bmatrix} y \\ y + x \\ x - y \end{bmatrix}$$

Matrix eigenvalues as a list

```
> Eigenvalues(A, output = list)
```

$$[y, y + x, x - y]$$

```
> B := Matrix([ [3, -I, 0], [I, 3, 0], [0, 0, 4] ])
```

$$B := \begin{bmatrix} 3 & -I & 0 \\ I & 3 & 0 \\ 0 & 0 & 4 \end{bmatrix}$$

```
> Eigenvalues(B, output = list)
```

$$[2, 4, 4]$$

```
> M := <<1, 4, -2><-1, 0, 1><-1, 2, 1>
```

$$M := \begin{bmatrix} 1 & -1 & -1 \\ 4 & 0 & 2 \\ -2 & 1 & 1 \end{bmatrix}$$

```
> Eigenvalues(M, output = list)
```

$$[2, I, -I]$$

Symmetric random matrix with float values

```
> R := RandomMatrix(3, outputoptions = [shape = symmetric, storage
= rectangular, datatype = float])
```

$$R := \begin{bmatrix} 30. & -23. & 91. \\ -23. & 63. & -38. \\ 91. & -38. & -38. \end{bmatrix}$$

Matrix eigenvalues as vector-row

```
> Eigenvalues(R, output = Vector[row])
```

$$[-103.221603816942689, 36.3950990097986775, 121.826504807143976]$$

Examples of calculating eigenvectors

> *with(LinearAlgebra) :*

> $A := \text{Matrix}(3, 3, [x, 0, y, x, y, 0, y, 0, x])$

$$A := \begin{bmatrix} x & 0 & y \\ x & y & 0 \\ y & 0 & x \end{bmatrix}$$

Default output for Eigenvectors command: sequence of 1) eigenvalues as vector-column and 2) matrix of eigenvectors by columns

> $\text{Eigenvectors}(A)$

$$\begin{bmatrix} y+x \\ y \\ x-y \end{bmatrix}, \begin{bmatrix} 1 & 0 & -1 \\ 1 & 1 & -\frac{x}{-2y+x} \\ 1 & 0 & 1 \end{bmatrix}$$

Only matrix of eigenvectors by columns

> $\text{Eigenvectors}(A, \text{output} = \text{vectors})$

$$\begin{bmatrix} 0 & 1 & -1 \\ 1 & 1 & -\frac{x}{-2y+x} \\ 0 & 1 & 1 \end{bmatrix}$$

List of eigenvalues and eigenvectors where each eigenvalue is shown with its multiplicity and corresponding eigenvector

> $\text{Eigenvectors}(A, \text{output} = \text{list})$

$$\left[\left[y, 1, \left[\begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \right] \right], \left[y+x, 1, \left[\begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \right] \right], \left[x-y, 1, \left[\begin{bmatrix} -1 \\ -\frac{x}{-2y+x} \\ 1 \end{bmatrix} \right] \right] \right]$$

> $B := \text{Matrix}([[3, -I, 0], [I, 3, 0], [0, 0, 4]])$

$$B := \begin{bmatrix} 3 & -I & 0 \\ I & 3 & 0 \\ 0 & 0 & 4 \end{bmatrix}$$

Example of multiple eigenvalue ($\lambda=4$ repeats twice)

> $\text{Eigenvectors}(B, \text{output} = \text{list})$

$$\left[\left[2, 1, \left[\begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix} \right] \right], \left[4, 2, \left[\begin{bmatrix} -1 \\ 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \right] \right] \right]$$

Spectral analysis: characteristic polynomial and characteristic matrix

LinearAlgebra :

- **CharacteristicPolynomial (A, lambda)** – characteristic polynomial $P_A(\lambda) = \det(A - \lambda E)$, where E is the identity matrix
- **MinimalPolynomial (A, lambda)** – minimal polynomial (divisor)
- **CharacteristicMatrix (A, lambda, outopts)** – characteristic matrix in the form $\text{lambda} * E - A$

```
[> with(LinearAlgebra) :  
> M := Matrix([[1, 1, 0], [0, 1, 3], [0, 0, 2]])  
M :=  $\begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 3 \\ 0 & 0 & 2 \end{bmatrix}$   
> CharacteristicPolynomial(M, x)  
-2 + x3 - 4x2 + 5x  
> solve(%, x)  
2, 1, 1
```

```
[> Eigenvalues(M, output='list')  
[2, 1, 1]  
> CharacteristicMatrix(M, lambda)  
 $\begin{bmatrix} \lambda - 1 & -1 & 0 \\ 0 & \lambda - 1 & -3 \\ 0 & 0 & \lambda - 2 \end{bmatrix}$ 
```

Example conducting spectral analysis in various ways

```
> with(LinearAlgebra):
```

```
> A := Matrix([[1, 1, 0], [0, 1, 3], [0, 0, 2]])
```

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

Compute eigenvalues

```
> lambdas := Eigenvalues(A)
```

$$\text{lambdas} := \begin{bmatrix} 2 \\ 1 \\ 1 \end{bmatrix}$$

Compute eigenvectors using lis output (eigenvalues with multiplicities and corresponding eigenvector)

```
> vectors := Eigenvectors(A, output='list')
```

$$\text{vectors} := \left[\left[2, 1, \begin{bmatrix} 3 \\ 3 \\ 1 \end{bmatrix} \right], \left[1, 2, \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \right] \right]$$

Compute eigenvalues as the roots of characteristic polynomial

```
> CharacteristicPolynomial(A, λ)
```

$$-2 + \lambda^3 - 4\lambda^2 + 5\lambda$$

```
> roots := solve(%, λ)
```

$$\text{roots} := 2, 1, 1$$

Check computed eigenvalues and corresponding eigenvector by substituting them in the equation $Av = \lambda v$

The first pair of eigenvalues and eigenvector

```
> λ1 := lambdas[1]; v1 := vectors[1][3][1]
```

$$\lambda_1 := 2$$

$$v_1 := \begin{bmatrix} 3 \\ 3 \\ 1 \end{bmatrix}$$

```
> Equal(A.v1, λ1.v1)
```

true

The second pair of eigenvalues and eigenvector

```
> λ2 := lambdas[2]; v2 := vectors[2][3][1]
```

$$\lambda_2 := 1$$

$$v_2 := \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$

```
> Equal(A.v2, λ2.v2)
```

true

Determining the type of matrix: positive (negative) definiteness

LinearAlgebra :

- **IsDefinite (A, q)** – checks the positive/negative definiteness of the matrix **A**, parameter **q** has the form **query = attribute**, where attribute can have one of the values:
 - **'positive_definite'** – positive definite, i.e.:
 $x^H A x > 0 \quad \forall x \in C^n, \quad x \neq 0, \quad A \in C^{n \times n}$ for a complex matrix, where x^H is conjugate transposed vector, A – Hermitian matrix: $A = A^H$
For a real matrix, such a query will return true if $x^T A x > 0 \quad \forall x \in R^n, \quad x \neq 0, \quad A \in R^{n \times n}$
Where x^T is transposed vector.
In this case, the matrix A may be asymmetrical (i.e.) $A \neq A^T$
Equivalent definition: all eigenvalues are positive: $\lambda > 0$
 - **'positive_semidefinite'** – positive semidefinite: $(\lambda \geq 0) \quad x^H A x \geq 0$
 - **'negative_definite'** – negative definite: $(\lambda < 0) \quad x^H A x < 0$
 - **'negative_semidefinite'** – negative semidefinite: $(\lambda \leq 0) \quad x^H A x \leq 0$
 - **'indefinite'** - indefinite
- **IsDefinite (A)** – checks whether the matrix **A** is positive definite, i.e. by default **query = 'positive_definite'**

Positive/Negative Definiteness of a Matrix: Examples

```
> with(LinearAlgebra) :
> B := DiagonalMatrix([-5, 0, -1])
```

$$B := \begin{bmatrix} -5 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & -1 \end{bmatrix}$$

The matrix is not positive definite

```
> IsDefinite(B)
```

false

The matrix is not positive semidefinite

```
> IsDefinite(B, query='positive_semidefinite')
```

false

Quotation marks are not necessary

The matrix is not negative definite

```
> IsDefinite(B, query=negative_definite)
```

false

The matrix is negative semidefinite

```
> IsDefinite(B, query=negative_semidefinite)
```

true

```
> Eigenvalues(B);
```

All eigenvalues are nonnegative

$$\begin{bmatrix} 0 \\ -1 \\ -5 \end{bmatrix}$$

The matrix is symmetric

```
> Equal(B, B%T)
```

true

```
> A1 := Matrix([[3, 2], [1, 4]])
```

$$A1 := \begin{bmatrix} 3 & 2 \\ 1 & 4 \end{bmatrix}$$

```
> A2 := DiagonalMatrix([2, 2])
```

$$A2 := \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}$$

```
> A := DiagonalMatrix([A1, A2])
```

$$A := \begin{bmatrix} 3 & 2 & 0 & 0 \\ 1 & 4 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 2 \end{bmatrix}$$

The matrix is positive definite

```
> IsDefinite(A)
```

true

The matrix is not symmetric

```
> Equal(A, A%T)
```

false

All eigenvalues are positive

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

$$\begin{bmatrix} 5 \\ 2 \\ 2 \\ 2 \end{bmatrix}$$

Determining the type of a matrix: orthogonality and unitarity

LinearAlgebra :

- **IsOrthogonal (A)** – checks whether the matrix **A** is orthogonal i.e. such that $AA^T = A^T A = E$, where E is the identity matrix. Equivalently: $A^T = A^{-1}$
- **IsUnitary (A)** – tests whether a complex matrix **A** is unitary: i.e. such that $AA^H = A^H A = E$, where E is the identity matrix. Equivalently: $A^H = A^{-1}$

```
> with(LinearAlgebra) :  
> A := Matrix([ [1, 2], [3, 4] ])  
A :=  $\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$   
  
> IsOrthogonal(A)  
false  
  
> B := Matrix([ [cos(x), -sin(x)], [sin(x), cos(x)] ])  
B :=  $\begin{bmatrix} \cos(x) & -\sin(x) \\ \sin(x) & \cos(x) \end{bmatrix}$   
  
> IsOrthogonal(B)  
true
```

```
> B-1  
 $\begin{bmatrix} \frac{\cos(x)}{\cos(x)^2 + \sin(x)^2} & \frac{\sin(x)}{\cos(x)^2 + \sin(x)^2} \\ -\frac{\sin(x)}{\cos(x)^2 + \sin(x)^2} & \frac{\cos(x)}{\cos(x)^2 + \sin(x)^2} \end{bmatrix}$   
  
> B%T  
 $\begin{bmatrix} \cos(x) & \sin(x) \\ -\sin(x) & \cos(x) \end{bmatrix}$   
  
> Equal(simplify(B-1), B%T)  
true
```

Solution of systems of linear equations, matrix kernel

LinearAlgebra :

- **LinearSolve (A,B)** – solves the equation $AX=B$, where **B** is the matrix or vector on the right-hand side and **X** is the matrix or vector of unknowns. Full syntax:
LinearSolve (A, B, m, t, c, ip, outopts, methopts) – the remaining arguments are optional (for details, see Help) . The meanings of some parameters:
m is the parameter of the method to be used in the form **method = name**, where **name** can take the values 'none', 'solve', 'subs', 'Cholesky', 'LU', 'QR', 'hybrid', 'modular', 'SparseLU', 'SparseDirect' or 'SparseIterative'
- **NullSpace (A, outopts)** – search for the basis of the matrix kernel, i.e. vectors $\{x: Ax=0\}$ is equivalent to solving a homogeneous system of equations
- **GenerateEquations (A, v, B)** – generating a system of linear equations $Av=B$, where **A** is a coefficient matrix of size $m \times n$, **v** is a list of unknowns of length **n**, **B** is the vector of the right-hand side

GenerateEquations (A, [x, y, z], b)

- **GenerateMatrix (eqns, vars)** – generates a matrix of coefficients from a list (set) of equations **eqns** and a list (set) of unknown **vars**. Option **augmented=true** outputs a matrix with an added column from the right-hand side vector

GenerateMatrix ([eq1, e2, eq3], [x, y, z])

Examples of solving a system of linear equations and finding the kernel of a matrix

```

> with(LinearAlgebra) :
> sys := [3 x1 + 2 x2 + 3 x3 = 1, x1 + x2 + x3 = 3, x1
      + 2 x2 - x3 = 1] :
> vars := [x1, x2, x3] :
> A, b := GenerateMatrix(sys, vars)

      
$$A, b := \begin{bmatrix} 3 & 2 & 3 \\ 1 & 1 & 1 \\ 1 & 2 & -1 \end{bmatrix}, \begin{bmatrix} 1 \\ 3 \\ 1 \end{bmatrix}$$


> M := GenerateMatrix(sys, vars, augmented = true)

      
$$M := \begin{bmatrix} 3 & 2 & 3 & 1 \\ 1 & 1 & 1 & 3 \\ 1 & 2 & -1 & 1 \end{bmatrix}$$


> LinearSolve(A, b)

      
$$\begin{bmatrix} -10 \\ 8 \\ 5 \end{bmatrix}$$


> s := GenerateEquations(A, [x, y, z], b)
s := [3 x + 2 y + 3 z = 1, x + y + z = 3, x + 2 y - z = 1]

> solve(s)

      {x = -10, y = 8, z = 5}

```

```

> with(LinearAlgebra) :
  A := <<6,3,0>|<4,2,0>|<2,1,0>>;

      
$$A := \begin{bmatrix} 6 & 4 & 2 \\ 3 & 2 & 1 \\ 0 & 0 & 0 \end{bmatrix}$$


```

Matrix kernel: vectors that satisfy $Ax=0$

```

> kern := NullSpace(A) ;

      
$$\text{kern} := \left\{ \begin{bmatrix} -\frac{1}{3} \\ 0 \\ 1 \end{bmatrix}, \begin{bmatrix} -\frac{2}{3} \\ 1 \\ 0 \end{bmatrix} \right\}$$


```

Check that substitution into equation gives zero vector

```

> A.kern[1] ; A.kern[2] ;

      
$$\begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$


```

Matrix factorization: QR factorization

LinearAlgebra :

- **QRDecomposition (A)** – QR-decomposition of the matrix: $A=QR$, where **Q** is an orthogonal matrix, **R** is an upper triangular matrix.
- Full syntax:

QRDecomposition (A, fs, out, c, outopts , ...) – the remaining arguments are optional (see Help for details) . The meanings of some parameters are:

fs – a logical parameter in the form **fullspan = 'false'** (default, partial QR-decomposition) or **fullspan = 'true'** (or simply **fullspan** , full QR-decomposition)

out – the parameter of the output matrix in the form **output = obj** , where **obj** can take the values **'Q', 'R', 'NAG', 'rank'** or a list of these values.

outopts – sets **outputoptions options** for the resulting object

An example of calculating the QR decomposition of a matrix

> with(LinearAlgebra) :

> A := $\left\langle \begin{pmatrix} 2 \\ 3 \\ 1 \end{pmatrix}, \begin{pmatrix} 4 \\ 6 \\ -1 \end{pmatrix} \right\rangle \left\langle \begin{pmatrix} -1 \\ -\frac{3}{2} \\ -2 \end{pmatrix} \right\rangle$

$$A := \begin{bmatrix} 2 & 4 & -1 \\ 3 & 6 & -\frac{3}{2} \\ 1 & -1 & -2 \end{bmatrix}$$

> Rank(A) Since Rank(A)=2, this matrix has incomplete rank (less than number of rows)
2

Thin (reduced) QR-decomposition

> Q, R := QRDecomposition(A)

$$Q, R := \begin{bmatrix} \frac{1}{7}\sqrt{14} & \frac{1}{91}\sqrt{182} \\ \frac{3}{14}\sqrt{14} & \frac{3}{182}\sqrt{182} \\ \frac{1}{14}\sqrt{14} & -\frac{1}{14}\sqrt{182} \end{bmatrix}, \begin{bmatrix} \sqrt{14} & \frac{25}{14}\sqrt{14} & -\frac{17}{28}\sqrt{14} \\ 0 & \frac{3}{14}\sqrt{182} & \frac{3}{28}\sqrt{182} \end{bmatrix}$$

Compute sizes of Q and R

> mQ, nQ := Dimensions(Q); mR, nR := Dimensions(R);
mQ, nQ := 3, 2
mR, nR := 2, 3

Check that A=QR

> Q.R

$$\begin{bmatrix} 2 & 4 & -1 \\ 3 & 6 & -\frac{3}{2} \\ 1 & -1 & -2 \end{bmatrix}$$

> Equal(A, Q.R)

true

Full QR-decomposition (R is square upper-triangular)

> Qf, Rf := QRDecomposition(A, fullspan)

$$Qf, Rf := \begin{bmatrix} \frac{1}{7}\sqrt{14} & \frac{1}{91}\sqrt{182} & \frac{3}{13}\sqrt{13} \\ \frac{3}{14}\sqrt{14} & \frac{3}{182}\sqrt{182} & -\frac{2}{13}\sqrt{13} \\ \frac{1}{14}\sqrt{14} & -\frac{1}{14}\sqrt{182} & 0 \end{bmatrix},$$

$$\begin{bmatrix} \sqrt{14} & \frac{25}{14}\sqrt{14} & -\frac{17}{28}\sqrt{14} \\ 0 & \frac{3}{14}\sqrt{182} & \frac{3}{28}\sqrt{182} \\ 0 & 0 & 0 \end{bmatrix}$$

Compute sizes of Qf and Rf

> mQf, nQf := Dimensions(Qf); mR, nR := Dimensions(Rf);
mQf, nQf := 3, 3
mR, nR := 3, 3

Check that multiplication gives initial matrix

> Qf.Rf

$$\begin{bmatrix} 2 & 4 & -1 \\ 3 & 6 & -\frac{3}{2} \\ 1 & -1 & -2 \end{bmatrix}$$

> Equal(A, Qf.Rf)

true

Matrix factorization: LU decomposition

LinearAlgebra :

- **LUDecomposition (A)** – LU- decomposition of a matrix such that: $A=PLU$, where L is a lower triangular matrix, U is an upper triangular matrix, P is a permutation matrix (the identity matrix with its rows rearranged).
- Full syntax:

LUDecomposition (A, m, out, c, ip , outopts , ...) – the remaining arguments are optional (for details, see Help) . The meanings of some parameters:

m is the parameter of the method used in the form **method = '**

GaussianElimination ' (normal Gaussian method, default), **method = '**

FractionFree ' (Gaussian method without division), **method = 'RREF'**, **method = 'Cholesky** ' or **method = 'none'**

out – the parameter of the output matrix in the form **output = obj** , where **obj** can take the values 'P', 'L', 'U'; 'U1', 'R ' (for $A=PLU1R$); 'NAG', 'determinant', 'rank' or a list of these values.

outopts – sets **outputoptions options** for the resulting object

An example of calculating the LU decomposition of a matrix

```
> with(LinearAlgebra):  
> A := <<0,-2, 0, 3>|<1, 3, 0, 1>|<1, 1, 0, 0>|<-3, 4, 1, 0>>
```

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

PLU-factorization: P is permutation matrix, L is lower-triangular, U is upper-triangular

```
> P, L, U := LUDecomposition(A)
```

$$P, L, U := \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}, \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ -\frac{3}{2} & \frac{11}{2} & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \begin{bmatrix} -2 & 3 & 1 & 4 \\ 0 & 1 & 1 & -3 \\ 0 & 0 & -4 & \frac{45}{2} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Check that A=PLU

```
> P.L.U
```

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

```
> Equal(A, P.L.U)
```

true

Display only upper-triangular matrix

```
> LUDecomposition(A, output='U')
```

$$\begin{bmatrix} -2 & 3 & 1 & 4 \\ 0 & 1 & 1 & -3 \\ 0 & 0 & -4 & \frac{45}{2} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

```
> GaussianElimination(A)
```

$$\begin{bmatrix} -2 & 3 & 1 & 4 \\ 0 & 1 & 1 & -3 \\ 0 & 0 & -4 & \frac{45}{2} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Solving ordinary differential equations in Maple: analytical solution

- **The dsolve and odetest**
- **Analytical solutions of ODEs**
 - General solution
 - Fundamental system of solutions
 - Solution of the Cauchy problem or boundary value problem
 - Graphical representation of the solution
 - Solution of systems of differential equations

The dsolve and odetest

Solution of ODE (ordinary differential equations)

- **dsolve (ODE)**
- **dsolve (ODE, y(x), options)**
- **dsolve ({ODE, ICs}, y(x), options)**

ODE – an ODE or a system of ODEs in the form of a set or list

y(x) is an indefinite function of one variable either a set or list of such functions (unknown equations)

ICs – initial conditions in the form $y(x_0)=a$, $D(y)(x_0)=b$, ..., $(D@@n)(y)(x_0)=c$

options – additional options, for example :

- **type = exact** (analytical solution is the default option),
- **type = series** (approximate solution in the form of a power series) ,
- **type = numeric** (numerical solution),
- **output = basis** (fundamental system of solutions)
- **method= laplace** (solution using integral transformations)

Checking the found solution

- **odetest (sol,ode)**

Analytical solution of ODE

- **dsolve (ode , var , options)**
- An analytical solution is sought by default (**type=exact**).
- If the desired function cannot be identified, the solution is output implicitly. To output the solution explicitly, specify the **type= explicit option** or set **_EnvExplicit := true** before executing the **dsolve** command.
- The general solution of a differential equation depends on arbitrary constants, the number of these constants is equal to the order of the differential equation. In *Maple* , such constants are typically denoted as **_C1** , **_C2** , and so on.
- In the output line, the solution of a non-homogeneous linear ODE always consists of terms that contain arbitrary constants (this is *the general solution* of the corresponding **homogeneous** ODE), and terms without arbitrary constants (this is *a particular solution* of the same **non-homogeneous** ODE).
- **dsolve** command produces a solution to a differential equation in the format $y(x)=\dots$. In order to work with the solution further (for example, to plot a solution graph), the right-hand side of the resulting solution should be separated with the command **rhs (%)** .

Analytical solution of one ODE: examples

The first order differential equation

> *restart,*

```
de := diff(y(x), x) + y(x) * cos(x) = sin(x) * cos(x);  
dsolve(de, y(x));
```

$$de := \frac{d}{dx} y(x) + y(x) \cos(x) = \sin(x) \cos(x)$$

$$y(x) = \sin(x) - 1 + e^{-\sin(x)} _C1$$

Using prime notation to denote derivatives

> *dsolve*($y' + y \cdot \cos(x) = \sin(x) \cdot \cos(x)$);

$$y(x) = \sin(x) - 1 + e^{-\sin(x)} _C1$$

The second order differential equation

> *restart,* $deq := \frac{d^2}{dx^2} y(x) - 2 \left(\frac{d}{dx} y(x) \right) + y(x) = \sin(x) + e^{-x}$;

> *dsolve*($deq, y(x)$);

$$y(x) = e^x _C2 + e^x x _C1 + \frac{1}{4} e^{-x} (2 \cos(x) e^x + 1)$$

Using prime notation to denote derivatives

> *dsolve*($y'' - 2 \cdot y' + y = \sin(x) + e^{-x}$);

$$y(x) = e^x _C2 + e^x x _C1 + \frac{1}{4} e^{-x} (2 \cos(x) e^x + 1)$$

Solution of the Cauchy problem or boundary value problem

- **dsolve** command can find a solution to a Cauchy problem or a boundary value problem if, in addition to the ODE, initial or boundary conditions are specified for the unknown function.

- **dsolve ({ODE, ICs}, y(x), options)**

Set initial or boundary conditions

- using a differential operator $y(a)=y_a$, $D(y)(b)=y_b$, $(D@@2)(y)(c)=y_c$, $(D@@3)(y)(d)=y_d, \dots, (D@@n)(y)(e)=y_e$
- using prime notation (2 D-Math): $y(a)=y_a, y'(b)=y_b, y''(c)=y_c, y'''(d)=y_d, \dots, y^{(n)}(e)=y_e$

Use D-operator to set initial conditions

> $de := \text{diff}(y(x), x\$4) + \text{diff}(y(x), x\$2) = 2 * \cos(x);$

$$de := \frac{d^4}{dx^4} y(x) + \frac{d^2}{dx^2} y(x) = 2 \cos(x)$$

> $cond := y(0) = -2, D(y)(0) = 1, (D@@2)(y)(0) = 0, (D@@3)(y)(0) = 0;$

$$cond := y(0) = -2, D(y)(0) = 1, D^{(2)}(y)(0) = 0, D^{(3)}(y)(0) = 0$$

> $dsolve(\{de, cond\}, y(x));$

$$y(x) = -2 \cos(x) - \sin(x) x + x$$

Use prime notation to set initial conditions

> $dsys := \{y'''' + y'' = 2 \cdot \cos(x), y(0) = -2, y'(0) = 1, y''(0) = 0, y'''(0) = 0\};$

> $dsolve(dsys, y(x));$

$$y(x) = -2 \cos(x) - \sin(x) x + x$$

Solution of a boundary value problem : the exact solution and its graph

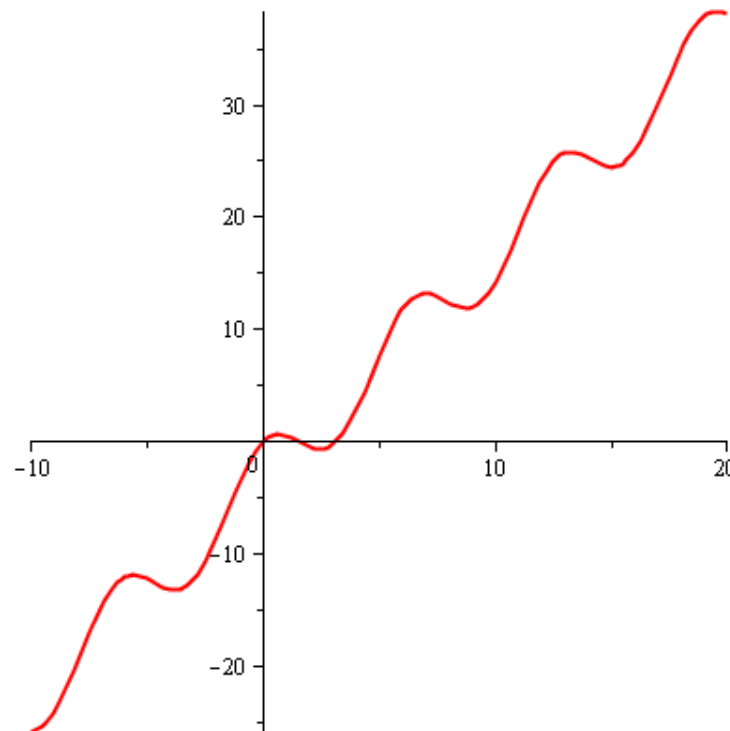
```
> restart;  
> de := diff(y(x), x$2) + y(x) = 2*x - pi; cond := y(0) = 0, y(pi/2) = 0;  
dsolve({de, cond}, y(x));
```

$$de := \frac{d^2}{dx^2} y(x) + y(x) = 2x - \pi$$

$$cond := y(0) = 0, y\left(\frac{1}{2}\pi\right) = 0$$

$$y(x) = \cos(x) \pi + 2x - \pi$$

```
> y1 := rhs(%): plot(y1, x=-10..20, thickness=2);
```



Solving ODE systems

- **dsolve ({ sys }, {x(t), y(t),...}, options)** is the solution of the ODE system
- **dsolve ({ sys , cond }, {x(t),y(t),...} , options)** is the solution of Cauchy problem or boundary value problem with a system of ODEs ; **x(t), y(t),...** is a set of unknown functions

$$\begin{aligned} &> \text{restart, sys} := \frac{d}{dt} x(t) = -4x(t) - 2y(t) + \frac{2}{e^t - 1}, \frac{d}{dt} y(t) = 6x(t) + 3y(t) - \frac{3}{e^t - 1} \\ &\quad \text{sys} := \frac{d}{dt} x(t) = -4x(t) - 2y(t) + \frac{2}{e^t - 1}, \frac{d}{dt} y(t) = 6x(t) + 3y(t) - \frac{3}{e^t - 1} \end{aligned}$$

Compare list and set when specifying the sytem of equations and the unknown functions

$$\begin{aligned} &> \text{sol1} := \text{dsolve}([\text{sys}], [x(t), y(t)]) \\ &\quad \text{sol1} := \left\{ x(t) = -\frac{1}{2} _C2 + \frac{2}{3} e^{-t} _C1 + 2 e^{-t} \ln(e^t - 1) + 2 e^{-t}, y(t) = \frac{-_C1 - 3 \ln(e^t - 1) - 3}{e^t} + _C2 \right\} \end{aligned}$$

$$\begin{aligned} &> \text{sol2} := \text{dsolve}(\{ \text{sys} \}, \{x(t), y(t)\}) \\ &\quad \text{sol2} := \left\{ x(t) = \frac{2 + 2 \ln(e^t - 1) - _C1}{e^t} + _C2, y(t) = -2 _C2 - 3 e^{-t} - 3 e^{-t} \ln(e^t - 1) + \frac{3}{2} e^{-t} _C1 \right\} \end{aligned}$$

$$\begin{aligned} &> \text{simplify}(\text{rhs}(\text{sol1}[1])); \text{simplify}(\text{rhs}(\text{sol2}[1])) \\ &\quad -\frac{1}{2} _C2 + \frac{2}{3} e^{-t} _C1 + 2 e^{-t} \ln(e^t - 1) + 2 e^{-t} \\ &\quad 2 e^{-t} + 2 e^{-t} \ln(e^t - 1) - e^{-t} _C1 + _C2 \end{aligned}$$

$$\begin{aligned} &> \text{simplify}(\text{rhs}(\text{sol1}[2])); \text{simplify}(\text{rhs}(\text{sol2}[2])) \\ &\quad -e^{-t} _C1 - 3 e^{-t} \ln(e^t - 1) - 3 e^{-t} + _C2 \\ &\quad -2 _C2 - 3 e^{-t} - 3 e^{-t} \ln(e^t - 1) + \frac{3}{2} e^{-t} _C1 \end{aligned}$$

Check that solution satisfies the given system of equations

$$\begin{aligned} &> \text{odetest}(\text{sol1}, [\text{sys}]) \\ &\quad [0, 0] \end{aligned}$$

$$\begin{aligned} &> \text{odetest}(\text{sol2}, \{ \text{sys} \}) \\ &\quad \{0\} \end{aligned}$$

Solution of a system of ODEs with initial conditions (Cauchy problem): examples

> *restart*;

> $\text{sys} := \text{diff}(x(t), t, t) + 5 \cdot \text{diff}(x(t), t) + 2 \cdot \text{diff}(y(t), t) + y(t) = 0,$
 $3 \cdot \text{diff}(x(t), t, t) + 5 \cdot x(t) + \text{diff}(y(t), t) + 3 \cdot y(t) = 0, x(0)$
 $= 1, (D(x))(0) = 0, y(0) = 1;$

$$\text{sys} := \frac{d^2}{dt^2} x(t) + 5 \left(\frac{d}{dt} x(t) \right) + 2 \left(\frac{d}{dt} y(t) \right) + y(t) = 0,$$

$$3 \left(\frac{d^2}{dt^2} x(t) \right) + 5 x(t) + \frac{d}{dt} y(t) + 3 y(t) = 0, x(0) = 1,$$

$$D(x)(0) = 0, y(0) = 1$$

> $\text{sol} := \text{dsolve}(\{\text{sys}\}, \{x(t), y(t)\})$

$$\text{sol} := \left\{ x(t) = -\frac{1}{2} e^{-t} + \frac{3}{2} e^t - 2 e^t t, y(t) = 2 e^{-t} - e^t + 4 e^t t \right\}$$

Check that solution satisfies the given system of equations

> $\text{odetest}(\text{sol}, \{\text{sys}\});$

$\{0\}$

Solving ordinary differential equations in Maple: approximate and numerical solutions

- Approximate solutions (power series)
- Numerical solutions
- **odeplot** command

Approximate solutions of ODEs

- For many types of ODEs, an exact analytical solution cannot be found. In these cases, the ODE can be solved using approximate methods, in particular, by expanding the unknown function into a power series.
- **dsolve (ODE, y(x), 'type=series')**
dsolve ({ODE, ICs}, y(x), 'type=series')
dsolve ({ sysODE , ICs }, { funcs }, 'type=series')
- Instead of **type=series** You can simply specify **series**
- The order of the series expansion (default = 6) can be changed using an environment variable **Order**. To specify the order of decomposition **n**, precede the **dsolve** command with **Order:=n**.
- If a general solution of an ODE is sought in the form of an expansion in a power series, then the coefficients at the powers of x of the found expansion will contain unknown values of the function at zero **y (0)** and its derivatives **D(y)(0)**, **(D@@2)(y)(0)**, etc.
- To isolate *a particular solution*, one must specify initial conditions, the number of which must coincide with the order of the corresponding ODE.
- The power series expansion has the type **series**. In order to work with the solution further (for example, to plot the solution), it should be converted to a polynomial using the **convert (% , polynom)**, and then select the right side of the resulting expression using the **rhs (%)** command.

Approximate solutions of ODEs: examples

Solution of the Cauchy problem as a power series expansion up to the 5-th order

> restart, Order := 5

> dsolve($\left\{ \frac{d}{dx} y(x) = y(x) + x e^{y(x)}, y(0) = 0 \right\}, y(x), type = series$)
Order := 5

$$y(x) = \frac{1}{2} x^2 + \frac{1}{6} x^3 + \frac{1}{6} x^4 + O(x^5)$$

Obtain general solution as a power series expansion up to the 4-th order

> restart; Order := 4; de := diff(y(x), x, x) - y(x)^3 = exp(-x)*cos(x);
Order := 4

$$de := \frac{d^2}{dx^2} y(x) - y(x)^3 = e^{-x} \cos(x)$$

When the initial conditions are not given, the answer is expressed in terms of the indeterminate function and its derivatives evaluated at the origin

> f:=dsolve(de, y(x), series);

$$f := y(x) = y(0) + D(y)(0) x + \left(\frac{1}{2} y(0)^3 + \frac{1}{2} \right) x^2 + \left(\frac{1}{2} y(0)^2 D(y)(0) - \frac{1}{6} \right) x^3 + O(x^4)$$

Set initial conditions

> subs(y(0) = 1, D(y)(0) = 0, f)

>

$$y(x) = 1 + x^2 - \frac{1}{6} x^3 + O(x^4)$$

Approximate solutions of ODEs: an example of comparing exact and approximate solutions

> *restart, dsys := y''' - y' = 3 * (2 - x^2) * sin(x), y(0) = 1, y'(0) = 1, y''(0) = 1;*

$$dsys := \frac{d^3}{dx^3} y(x) - \left(\frac{d}{dx} y(x) \right) = 3 (2 - x^2) \sin(x), y(0) = 1, D(y)(0) = 1, D^{(2)}(y)(0) = 1$$

Exact solution

> *dsolve({dsys}, y(x))*

$$y(x) = \frac{3}{4} e^{-x} + \frac{7}{4} e^x + \frac{21}{2} \cos(x) + 6 \sin(x) x - \frac{3}{2} \cos(x) x^2 - 12$$

Take right-hand side for the plot

> *y1 := rhs(%)*

$$y1 := \frac{3}{4} e^{-x} + \frac{7}{4} e^x + \frac{21}{2} \cos(x) + 6 \sin(x) x - \frac{3}{2} \cos(x) x^2 - 12$$

Approximate solution as series expansion

> *Order := 6; dsolve({dsys}, y(x), series)*

Order := 6

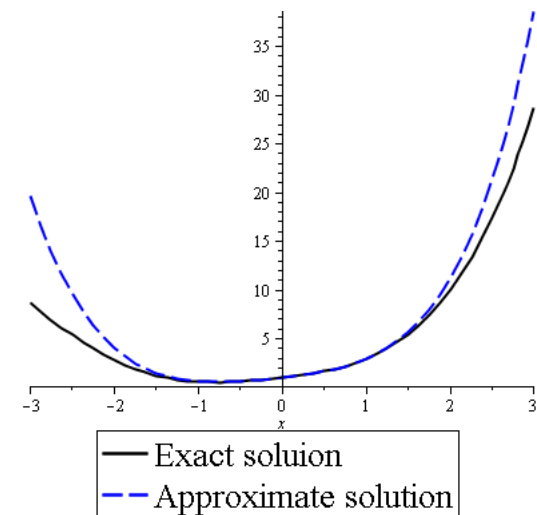
$$y(x) = 1 + x + \frac{1}{2} x^2 + \frac{1}{6} x^3 + \frac{7}{24} x^4 + \frac{1}{120} x^5 + O(x^6)$$

Convert to polynomial (neglect the residual) and take right-hand side for the plot

> *y2 := convert(rhs(%), polynomial)*

$$y2 := 1 + x + \frac{1}{2} x^2 + \frac{1}{6} x^3 + \frac{7}{24} x^4 + \frac{1}{120} x^5$$

> *plot([y1, y2], x=-3..3, color=[black, blue], thickness=2, linestyle=[solid, dash], legend
= ["Exact solution", "Approximate solution"], legendstyle=[font=["TIMES", 20]])*



Numerical solution of ODE and its graph : odeplot

- **dsolve (ODE, y(x), 'type=numeric')**
dsolve ({ODE, ICs}, y(x), 'type=numeric')
dsolve ({ sysODE , ICs }, { funcs }, 'type=numeric')
- Instead of **type=numeric** one can simply specify **numeric**
- In additional options, one can specify the method of numerical integration of the differential equation in the form **method = numeric_method**, where as **numeric_method** can be one of the following: **rkf45**, **rosenbrock**, **bvp**, **rkf45_dae**, **rosenbrock_dae**, **dverk78**, **lsode**, **gear**, **taylorseries**, **mebdfi**, **classical**.
- For example, **method = rkf45** – Runge-Kutta-Felberg method of 4-5th order (set by default); **method = dverk78** – Runge-Kutta method of 7-8th order; **method = classical** – classical Runge-Kutta method of 3rd order; **method = gear** - Gere's method.
- **dsolve** command with the parameter **type=numeric** creates a procedure that can be accessed to calculate the solution for a given value of the variable and to plot a graph over a given interval.
- **plots[odeplot] (sol_num , [x, y(x)], x=x1..x2 ,options)** is the graph of the numerical solution of the ODE, where **sol_num** is the numerical solution obtained using the **dsolve** command with the **numeric** option

Numerical solutions of ODEs: examples

```
> restart, eq :=  $\frac{d^2}{dx^2} y(x) - x \sin(y(x)) = \sin(2x)$ 
```

$$eq := \frac{d^2}{dx^2} y(x) - x \sin(y(x)) = \sin(2x)$$

```
> cond := y(0) = 0, D(y)(0) = 1
```

$$cond := y(0) = 0, D(y)(0) = 1$$

```
> sol_num := dsolve({eq, cond}, y(x), numeric)
```

```
sol_num := proc(x_rkf45) ... end proc
```

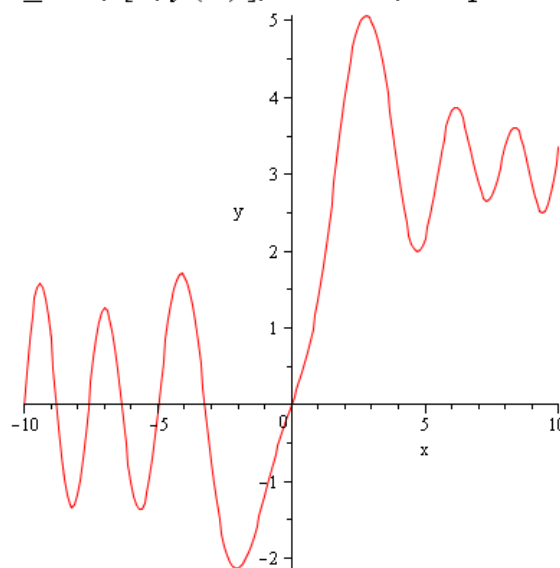
Solution for $x=0.5$ (function value and the values of the derivative at this point)

```
> sol_num(0.5)
```

$$\left[x = 0.5, y(x) = 0.544926322815993624, \frac{d}{dx} y(x) = 1.27250331302082364 \right]$$

Graph of numerical solution for x from $(-10,10)$:

```
> with(plots) : odeplot(sol_num, [x, y(x)], -10..10, numpoints = 200)
```



Graphical representation of solutions of ordinary differential equations

- **odeplot** command and graphical commands of the **DEtools** package
- 2 D Solution Plots, **DEplot** command
- Construction of a phase portrait and a direction field
 - **DEplot**
 - **phaseportrait**
 - **dfieldplot**

Graphical commands for representing solutions of ODEs and systems of ODEs

Plots package graphical commands for ODEs

- **odeplot (dsn , vars , range, options)** – plot of numerical solution of ODE or ODE systems

DEtools

- **DEplot** – a graph of the numerical solution of an ODE or a system of ODEs
- **DEplot3d** – 3D graph of the numerical solution of an ODE or a system of ODES
- **phaseportrait** – phase portrait for a system of two first-order ODEs or one ODE of any order; for an *autonomous* system of first-order ODEs the direction field is also drawn
- **dfieldplot** – plot of a two-dimensional direction field (vector field) for a system of two *autonomous* first-order ODEs or one first-order ODE

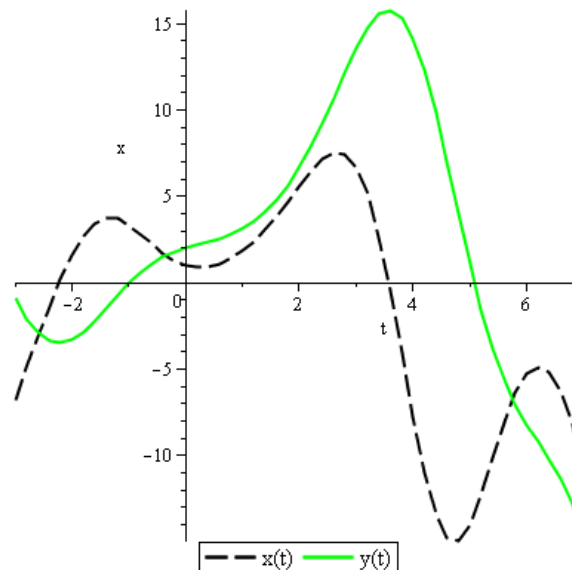
Plotting the numerical solution of the Cauchy problem for a first-order ODE system using the odeplot command

```
> restart, cond := x(0) = 1, y(0) = 2
                                cond := x(0) = 1, y(0) = 2

> sys :=  $\frac{d}{dt} x(t) = 2 y(t) \sin(t) - x(t) - t$ ,  $\frac{d}{dt} y(t) = x(t)$ 
                                sys :=  $\frac{d}{dt} x(t) = 2 y(t) \sin(t) - x(t) - t$ ,  $\frac{d}{dt} y(t) = x(t)$ 

> F := dsolve( {sys, cond}, [x(t), y(t)], numeric)
                                F := proc(x_rkf45) ... end proc

> with(plots) :
> p1 := odeplot(F, [t, x(t)], -3 ..7, color = black, thickness = 2, linestyle = 3, legend
    = "x(t)") :
> p2 := odeplot(F, [t, y(t)], -3 ..7, color = green, thickness = 2, legend = "y(t)") :
> display(p1, p2)
```



DEplot

Two-dimensional graphs of solutions of ODEs or systems of ODEs

- **DEplot** command from the **DEtools** package plots (two-dimensional) graphs of ODE solutions, as well as phase portraits and the direction field. This command is similar to the **odeplot** command, but is more powerful.
- For differential equations of the order higher than first, the **DEplot** command plots only the solution curves of the ODE, while for the first-order ODE systems, phase portraits with a direction field can be plotted.
- Unlike **odeplot**, the **DEplot** command itself finds a numerical solution to the differential equation.
- **DEplot (deqns , vars , trange , options)**
- **DEplot (deqns , vars , trange , inits , options)**
- **DEplot (deqns , vars , trange , xrange , yrange , options)**
- **DEplot (deqns , vars , trange , inits , xrange , yrange , options)**
- The main parameters of **DEplot** are similar to those of **odeplot** :
 - **deqns** – an equation or list of equations
 - **vars** – list of unknown functions;
 - **trange** – range of change of the independent variable;
 - **inits** – a list of lists of initial conditions for each integral curve;
 - **xrange** and **yrange** – ranges for the first and second dependent variables (functions);
 - **options** – additional parameters.

DEplot command (continued)

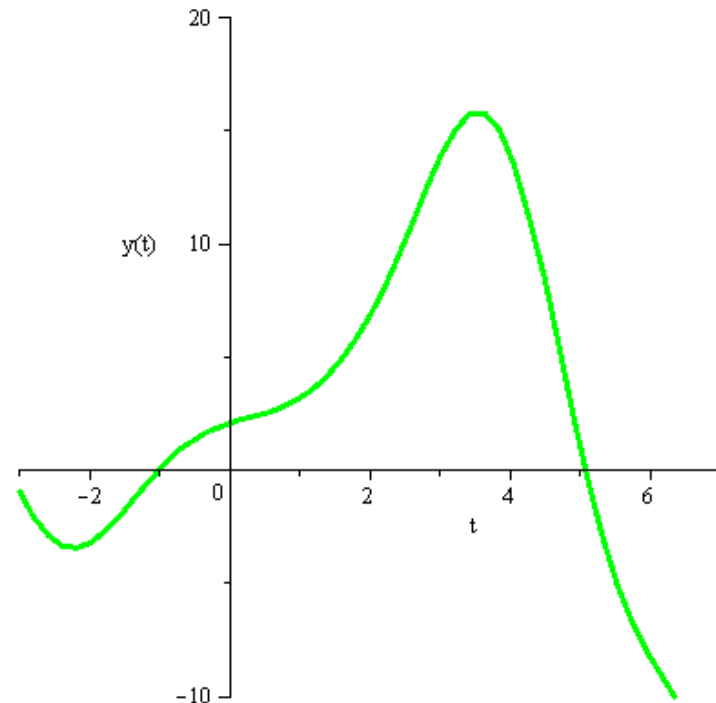
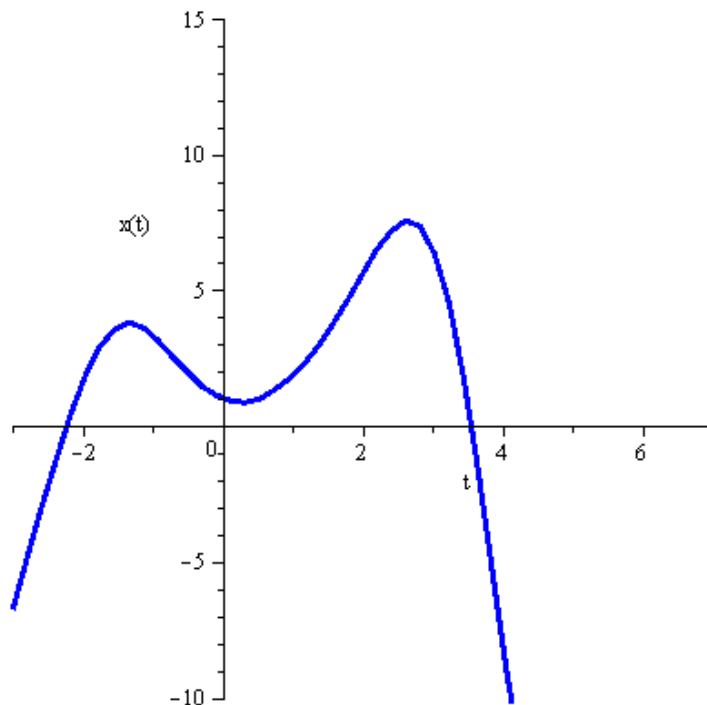
- To solve a differential equation of the n -th order, the initial conditions can be specified in a compact form: $[x_0, y_0, y'_0, y''_0, \dots]$, where x_0 is the point at which the initial conditions are specified, y_0 is the value of the desired function at the point x_0 ; $y'_0, y''_0, \dots$ are the values of first, second, etc. respective derivatives, up to the $(n - 1)$ -th order.

The most commonly used graphics options for the DEplot command

- **linecolor** – line color
- **color** – color of the arrows
- **arrows** = **SMALL**, **MEDIUM**, **LARGE**, **LINE** or **NONE** – arrow type (value **NONE** is set by default)
- **scene** – determines which variables to display on the graph: **scene** = $[x, y]$ – two-dimensional phase portrait, **scene** = $[t, x]$ or **scene** = $[t, y]$ – integral curves
- **iterations** – the number of iterations required to improve the accuracy of calculations (by default this number is 1)
- **stepsize** – a number equal to the distance between the points on the graph, by default for the interval $[a, b]$ it is equal to $\text{abs}((b-a))/48$, used to output a smoother curve
- **obsrange** = **true/false** – option to interrupt calculations if the solution graph goes beyond the interval set for drawing

Plot of the numerical solution of the Cauchy problem for a system of first-order ODEs using the DEplot command

```
> sys := diff(x(t), t) = 2·y(t)·sin(t) - x(t) - t, diff(y(t), t) = x(t);  
      sys :=  $\frac{d}{dt} x(t) = 2 y(t) \sin(t) - x(t) - t, \frac{d}{dt} y(t) = x(t)$   
=> cond := x(0) = 1, y(0) = 2 :  
=> with(DEtools) :  
=> DEplot([sys], [x(t), y(t)], t = -3 .. 7, x = -10 .. 15, [[cond]], scene = [t, x(t)], linecolor = blue);  
      DEplot([sys], [x(t), y(t)], t = -3 .. 7, x = -20 .. 15, y = -10 .. 20, [[cond]], scene = [t, y(t)],  
      linecolor = green)
```



Plotting the solution of the Cauchy problem (integral curve) for a third-order ODE using the DEplot command

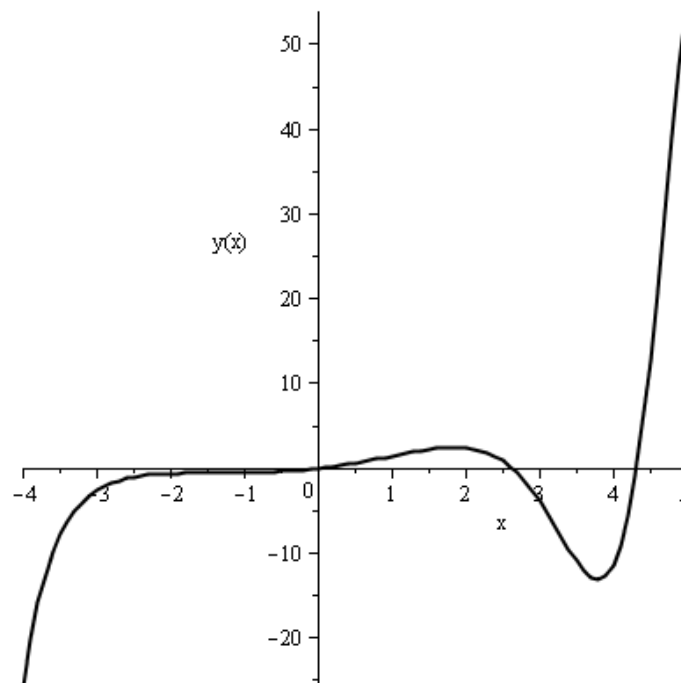
> *restart, with(DEtools) :*

The 3rd order differential equation

> $de3 := \frac{d^3}{dx^3} y(x) + x \sqrt{\left| \frac{d}{dx} y(x) \right|} + x^2 y(x) = 0 :$

Define initial conditions using D-operator and plot an integral curve for them

> $DEplot(de3, \{y(x)\}, x = -4..5, [[y(0) = 0, D(y)(0) = 1, D^{(2)}(y)(0) = 1]], stepsize = 0.1, linecolor = black, thickness = 2)$



Plotting the solution of the Cauchy problem (integral curves and direction field) for a first-order ODE using the DEplot command

```
> restart, with( DEtools ) :
```

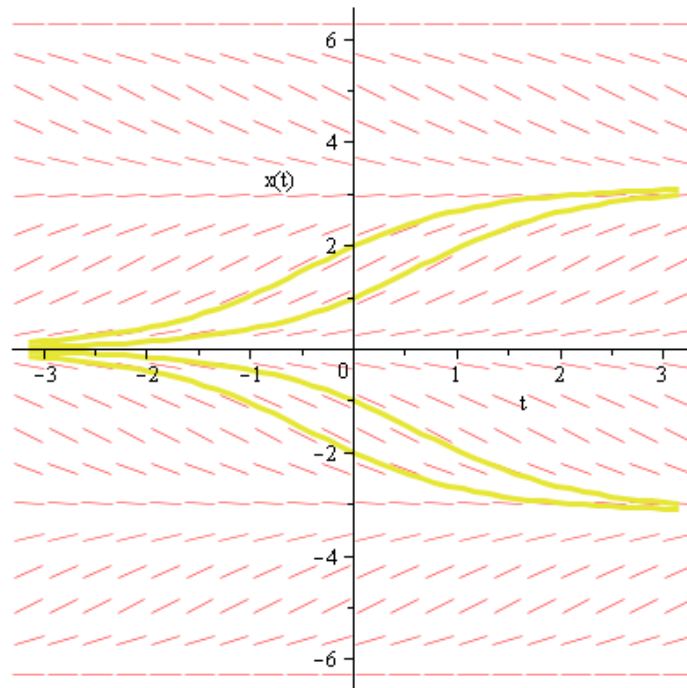
The 1st order differential equation

```
> de1 :=  $\frac{d}{dt} x(t) = \sin(x(t))$  :
```

Plot integral curves for several initial conditions in the plane (t,x(t))

Use compact form [x0,y0,y'0,...] for initial conditions

```
> DEplot( de1, x(t), t = - $\pi$  ..  $\pi$ , [ [ 0, 1 ], [ 0, -1 ], [ 0, 2 ], [ 0, -2 ] ], x = -2  $\pi$  .. 2  $\pi$ , arrows = LINE )
```



Phase portraits and direction fields

- In order to draw the entire phase portrait (the set of different phase trajectories), it is necessary to indicate the initial conditions for each phase trajectory .
- The initial conditions can be specified in a compact form: $[t_0, x_0, y_0]$, where t_0 is the point at which the initial conditions are set, x_0 and y_0 are the values of the functions x and y at the point t_0 .
- For *autonomous* systems of differential equations (i.e., not explicitly dependent on t) the direction field in the form of arrows will be constructed.

DEplot , phaseportrait commands and dfieldplot

- Using the **DEplot** command, one can construct a phase portrait on the (x, y) plane for a system of two first-order ODEs $x'(t)=f(x,y,t)$, $y'(t)=g(x,y,t)$ if **scene= $[x,y]$** is specified in the parameters of this command.
- **phaseportrait(deqns, vars, trange, inits, options)** - phase portrait for a system of two first-order ODEs or one ODE of any order, for an autonomous system of first-order ODEs the direction field is also drawn
- **dfieldplot (deqns , vars , trange , xrange , yrange , options)** – direction field for a system of two autonomous first-order ODEs or one first-order ODE. The **dfieldplot** command does not require initial conditions.

Direction field of a first-order nonlinear ODE using the DEplot command

- First order ODE of the form

$$\frac{dy(x)}{dx} = \frac{Q(x, y(x))}{P(x, y(x))} \quad (1)$$

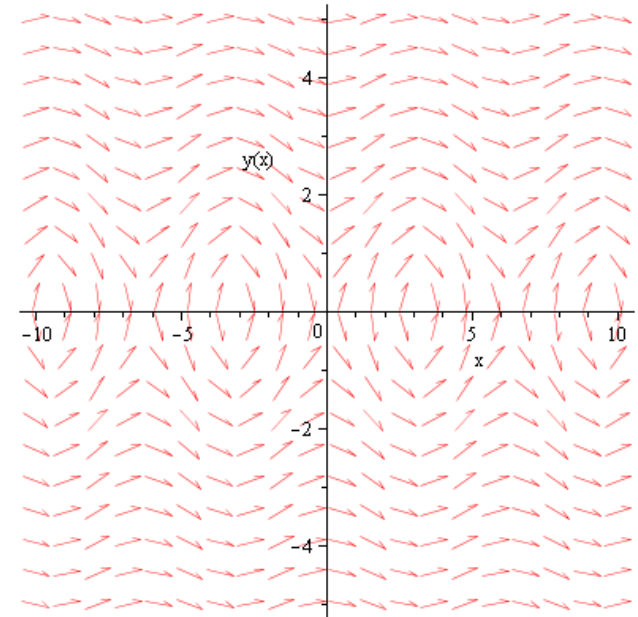
is equivalent to a system of two first-order ODEs of the form

$$\begin{cases} \frac{dx(t)}{dt} = P(x(t), y(t)) \\ \frac{dy(t)}{dt} = Q(x(t), y(t)) \end{cases} \quad (2)$$

- An equilibrium point of equation (1) or system (2) is a point (x_0, y_0) which is a solution of the system

$$\begin{cases} P(x, y) = 0, \\ Q(x, y) = 0. \end{cases}$$

```
> restart, with( DEtools ) : nonlin_ode :=  $\frac{d}{dx} y(x) = \frac{\sin(x)}{y(x)}$  :
> DEplot( nonlin_ode, y(x), x = -10 .. 10, y = -5 .. 5, stepsize = 0.05 )
```



```
> Q := numer( rhs( nonlin_ode ) )
```

$Q := \sin(x)$

```
> P := denom( rhs( nonlin_ode ) )
```

$P := y(x)$

Find the equilibrium points

```
> _EnvAllSolutions := true : solve( [ P = 0, Q = 0 ], [ x, y(x) ] )
[ [ x =  $\pi$ , y(x) = 0 ] ]
```

Phase trajectories with direction field for a nonlinear first-order ODE system with given initial conditions (using the DEplot command)

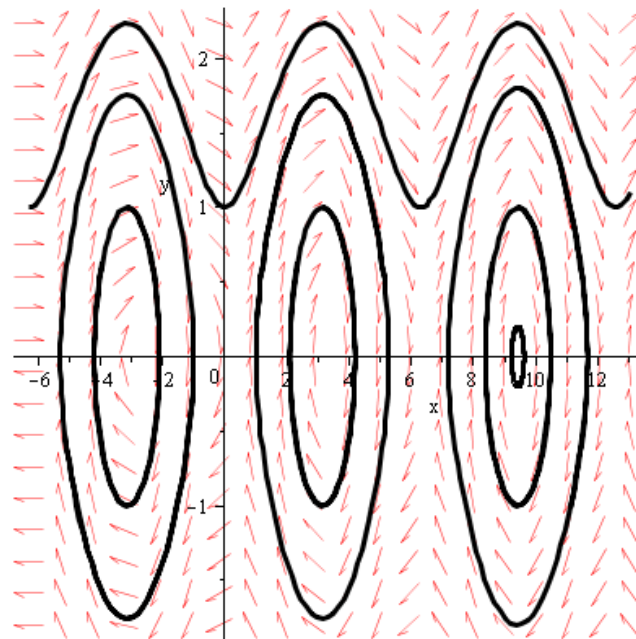
> *restart, with(DEtools) :*

Equivalent system of two first-order ODEs

> $sys := \frac{d}{dt} x(t) = y(t), \frac{d}{dt} y(t) = \sin(x(t))$

> $DEplot(\{sys\}, [x(t), y(t)], t = 0..4\pi, [[0, 1, 0], [0, -1, 0], [0, \pi, 1], [0, -\pi, 1], [0, 3\pi, 0.2], [0, 3\pi, 1], [0, 3\pi, 1.8], [0, -2\pi, 1]], stepsize = 0.1, linecolor = black)$

$sys := \frac{d}{dt} x(t) = y(t), \frac{d}{dt} y(t) = \sin(x(t))$



Phase trajectories without direction field for a nonlinear first-order ODE system with given initial conditions (using the DEplot command)

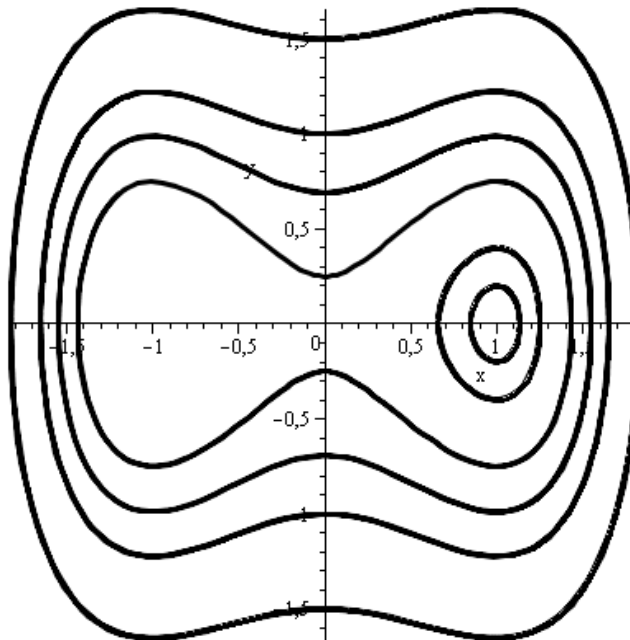
> *restart, with(DEtools) :*

Nonlinear autonomous system of two first-order ODEs

> $\text{sys} := \frac{d}{dt} x(t) = y(t), \frac{d}{dt} y(t) = x(t) - x(t)^3$

$$\text{sys} := \frac{d}{dt} x(t) = y(t), \frac{d}{dt} y(t) = x(t) - x(t)^3$$

> *DEplot({sys}, [x(t), y(t)], t = 0..20, [[0, 1, 0.2], [0, 0, 1], [0, 1, 0.4], [0, 1, 0.75], [0, 0, 1.5], [0, -0.1, 0.7]], stepsize = 0.1, arrows = none, linecolor = black)*



Phase portraits of linear autonomous systems for the first-order differential equations

Consider a linear dynamic system or a single equation
$$\begin{cases} \frac{dx}{dt} = ax + by, \\ \frac{dy}{dt} = cx + dy \end{cases} \quad (1) \quad \frac{dy}{dx} = \frac{cx + dy}{ax + by} \quad (2)$$

An **equilibrium point** of equation (2) or system (1) is a solution of the linear system
$$\begin{cases} ax + by = 0, \\ cx + dy = 0 \end{cases}$$

It is obvious that for a homogeneous linear system the solution depends on the determinant of the system matrix

$$A = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$$

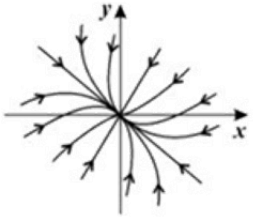
- If **det(A) ≠ 0**, i.e. the matrix is non-singular, then the solution of the linear system will be the **point {x=0, y=0}**.
- If **det(A)=0**, i.e. the matrix is singular, then the solution of the linear system will be all points of the line $ax+by=0$ (or $y=-a*x/b$), since in this case the coefficients will be proportional: $a/b=c/d$. In this case, instead of an equilibrium point we have an equilibrium **line $y=-a*x/b$**

The type of phase portrait of a linear autonomous first-order differential equation system is determined by the eigenvalues λ_1, λ_2 of the matrix A , i.e., the roots of the characteristic polynomial $P_A(\lambda) = \det(A - \lambda E)$, where E is the identity matrix:

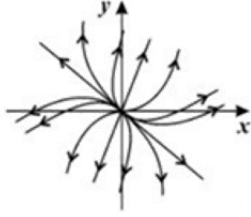
$$\begin{vmatrix} a - \lambda & b \\ c & d - \lambda \end{vmatrix} = 0$$

Types of phase portraits of linear autonomous systems for the first-order differential equations

Nodes



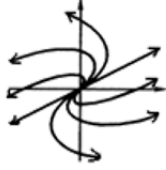
Stable Node (Sink)
 λ_1, λ_2 are real and negative



Unstable Node (Source)
 λ_1, λ_2 are real and positive



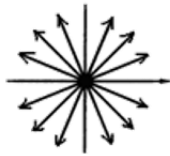
Stable Degenerate Node
 $\lambda_1 = \lambda_2$ is real and negative



Unstable Degenerate Node
 $\lambda_1 = \lambda_2$ is real and positive

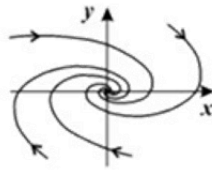


Special case:
 Stable Degenerate
 Star Node
 $\lambda_1 = \lambda_2$ is real and negative
 (many eigenvectors)

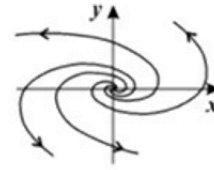


Special case:
 Unstable Degenerate
 Star Node
 $\lambda_1 = \lambda_2$ is real and positive
 (many eigenvectors)

Foci (Spirals)

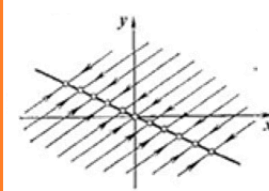


Stable Focus
 (Spiral Sink)
 λ_1, λ_2 are complex
 and $\text{Re}\lambda_1, \text{Re}\lambda_2$ are negative)

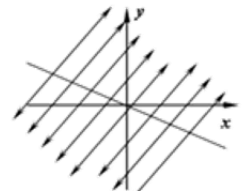


Unstable Focus
 (Spiral Source)
 λ_1, λ_2 are complex
 and $\text{Re}\lambda_1, \text{Re}\lambda_2$ are positive)

Lines of Equilibria

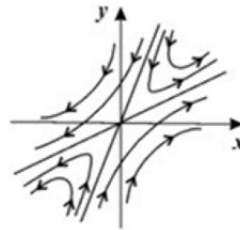


Stable Line of
 Equilibria
 $\lambda_1 = 0, \lambda_2 < 0$



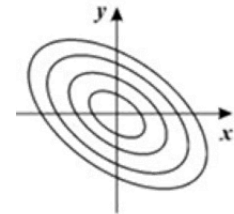
Unstable Line of
 Equilibria
 $\lambda_1 = 0, \lambda_2 > 0$

Saddle Point



Saddle Point (always unstable)
 λ_1, λ_2 are real and
 have opposite signs

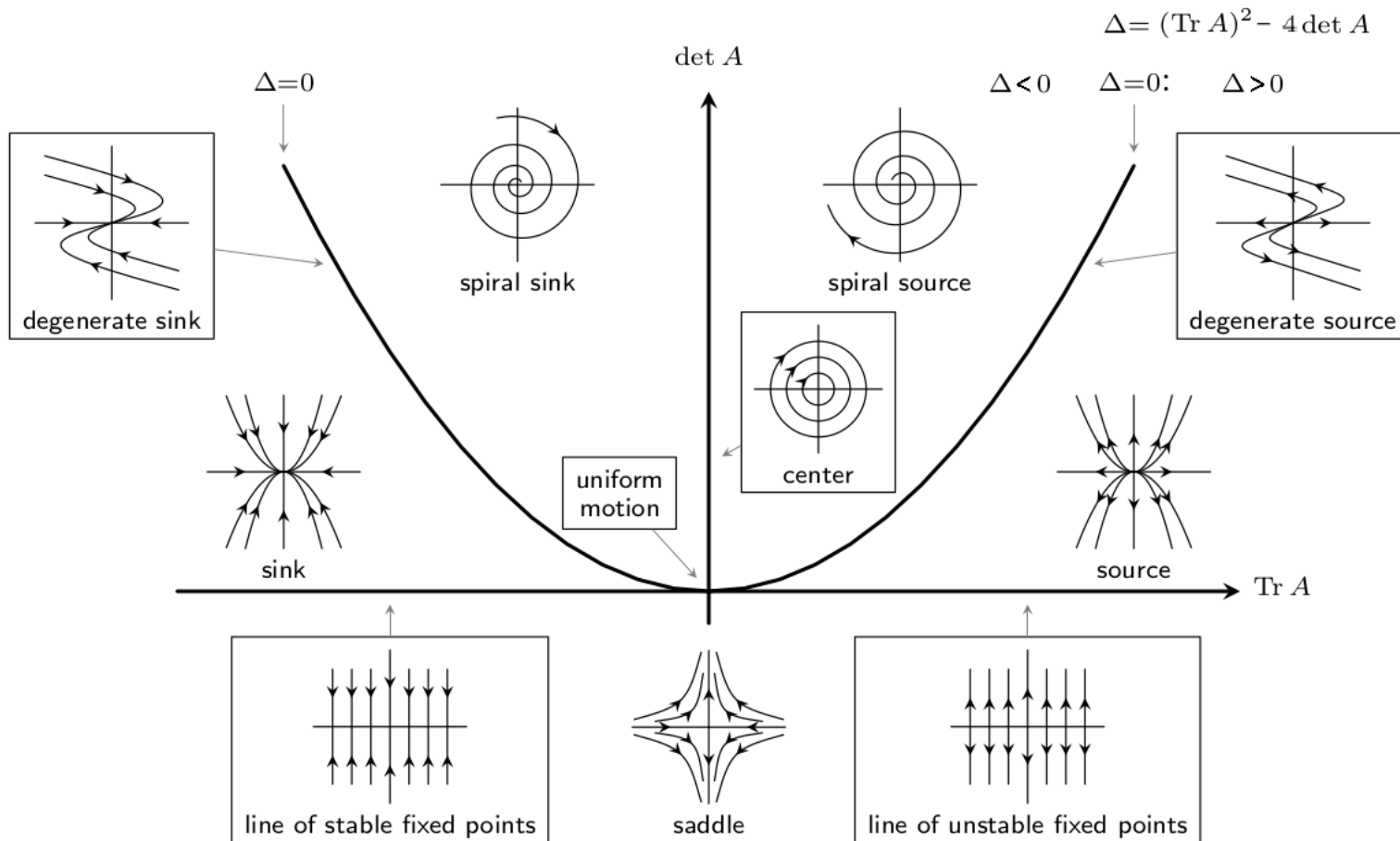
Center



Center (always stable)
 λ_1, λ_2 are complex and
 purely imaginary

Stability diagram of linear autonomous system $dx/dt = Ax$: phase portraits on $(\det A, \text{tr} A)$ plane

Poincaré Diagram: Classification of Phase Portraits in the $(\det A, \text{Tr } A)$ -plane

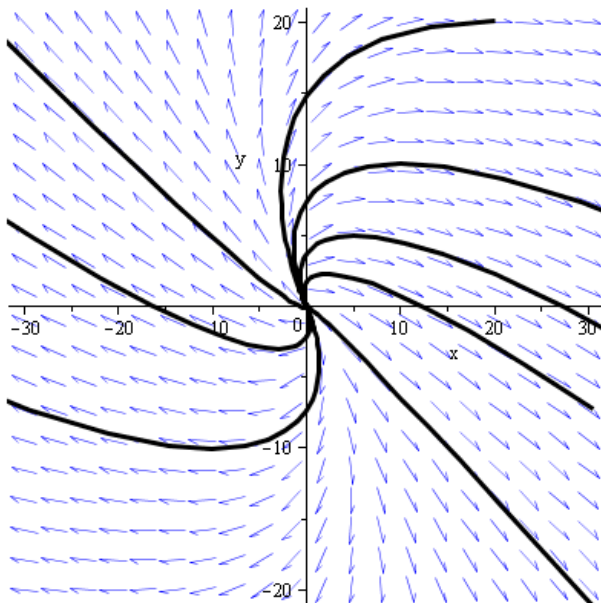


Phase trajectories and direction field for a linear autonomous first-order ODE system with given initial conditions (using the phaseportrait command)

```

> restart, with( DEtools ) :
> P := 3 x(t) + y(t) : Q := -x(t) + y(t) :
> sys :=  $\frac{d}{dt} x(t) = P, \frac{d}{dt} y(t) = Q$ 
sys :=  $\frac{d}{dt} x(t) = 3 x(t) + y(t), \frac{d}{dt} y(t) = -x(t) + y(t)$ 
> phaseportrait( [sys], [x(t), y(t)], t = -10..10, [[0, 1, -2],
[0, -3, -3], [0, -2, 4], [0, 5, 5], [0, 5, -3], [0, -5, 2],
[0, 5, 2], [0, -1, 2]], x = -30..30, y = -20..20, stepsize
= 0.1, colour = blue, linecolor = black)

```



```

> with( LinearAlgebra ) :
> A, b := GenerateMatrix( [P=0, Q=0], [x(t), y(t)] )
A, b :=  $\begin{bmatrix} 3 & 1 \\ -1 & 1 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \end{bmatrix}$ 
> Eigenvalues(A)
 $\begin{bmatrix} 2 \\ 2 \end{bmatrix}$ 

```

This is unstable degenerate node, as eigenvalues are equal and positive

Solution of partial differential equations

- pdsolve
- PDEtools package
- PDEplot command

pdsolve command

Solution of an equation or system of partial differential equations (PDEs)

- **pdsolve (PDE, f, other_options)**
- **pdsolve (PDE_system , funcs , other_options)**
- **pdsolve (PDE_or_PDE_system , conds , type=numeric, other_options)**

Checking the found solution

- **pdetest (sol,pde)**

```
> PDE := x*diff(f(x,y),y)-y*diff(f(x,y),x) = 0;
```

$$PDE := x \left(\frac{\partial}{\partial y} f(x, y) \right) - y \left(\frac{\partial}{\partial x} f(x, y) \right) = 0$$

```
> sol:=pdsolve(PDE);
```

$$sol := f(x, y) = _F1(x^2 + y^2)$$

_F1 - произвольная функция

Проверка решения

```
> pdetest(sol,PDE);
```

pdsolve command: using options

Solve thermal conductivity problem

$$> \text{heat} := \frac{\partial}{\partial t} u(x, t) = \frac{\partial^2}{\partial x^2} u(x, t) :$$

Search for solution as a product

$$> \text{pdsolve}(\text{heat}, \text{HINT} = '*')$$

$$(u(x, t) = _F1(x) _F2(t)) \text{ \&where } \left[\left\{ \frac{d^2}{dx^2} _F1(x) = _c1 _F1(x), \frac{d}{dt} _F2(t) = _c1 _F2(t) \right\} \right]$$

Build solution in explicit form

$$> \text{sol} := \text{pdsolve}(\text{heat}, \text{HINT} = X(x) T(t), \text{build})$$

$$\text{sol} := u(x, t) = e^{\sqrt{-c_1} x} _C1 e^{-c_1 t} _C2 + \frac{_C1 e^{-c_1 t} _C3}{e^{\sqrt{-c_1} x}}$$

Find particular solution for specific values of parameters

$$> \text{const} := \left\{ _c1 = -4, _C1 = \frac{I}{2}, _C2 = -1, _C3 = 1 \right\}$$

$$\text{const} := \left\{ _C1 = \frac{1}{2} I, _C2 = -1, _C3 = 1, _c1 = -4 \right\}$$

$$> \text{simplify}(\text{eval}(\text{rhs}(\text{sol}), \text{const}))$$

$$e^{-4 t} \sin(2 x)$$

PDEtools package: PDEplot command

- **PDEtools** – a package for working with partial differential equations (PDEs), contains commands for studying and finding analytical solutions to PDEs, visualizing PDE solutions, etc.
- **PDEplot (PDE, inits , srange , options)** – solution and plotting of first-order PDEs (linear or nonlinear)

Solve nonlinear wave equation

```
> pde :=  $\frac{\partial}{\partial t} u(x, t) + u(x, t) \cdot \frac{\partial}{\partial x} u(x, t)$  : ini := [s, 0, 0.5 + 0.5 · sin(s)] :
```

```
> with(PDEtools) :
```

```
> PDEplot(pde, x = 0 .. 4 · π, t = 0 .. 3, ini, s = 0 .. 4 · π, style = patch, orientation = [-70, 15], axes = normal)
```

