Maple

Benutzeroberfläche

Kommandofenster

> Anweisung, abgeschlossen durch ; oder :

(Ergebnis wird oder wird nicht angezeigt)

mehrzeilige Anweisung mit SHIFT-ENTER

> # Kommentar

Hilfesystem

> ? Befehlsname

help - Menü

Dateiverwaltung

Drucken

Speichern im Text-, LaTeX- oder HTML-Format

Laden und Speichern von Arbeitsblättern (Dateiname.mw)

Beispiel

```
> x:=4: y:=8; z:=1.2;
```

$$y := 8$$

$$z := 1.2$$

```
> x/y; x/z;
```

$$1/2$$

$$3.333333333$$

```
> ?if
> if x>y then
> print(x)
> else
> print(y)
> end if;
```

$$8$$

```
> ?mapletour
```

Rechenoperationen

Grundoperationen

$\pm$, $*$, $/$,

Vordefinierte Größen

Pi, I

Zuweisung

$>$ Variable := Ausdruck

z.B. Zahl, arithmetische Operation, Funktion, Gleichung

Datentypen

$\{a, b, \ldots\}$ Mengen

$[a, b, \ldots]$ Listen

...

Beispiel

```
> 5*4/3-2; 9*8/7-6.0;
```

$$\frac{14}{3}$$

$$4.28571429$$

```
> {I, Pi, (1+I)^4};
```

$$\{i, \pi, -4\}$$

```
> Digits:=20: x:=1.1: y:=1/x;
```

$$y := 0.90909090909090909091$$

```
> equations:=[2*s+3*t=4,5*s+6*t=7];
```

$$equations := [2\,s + 3\,t = 4, 5\,s + 6\,t = 7]$$

```
> unknowns:={s,t};
```

$$unknowns := \{s, t\}$$

```
> equations[1]; unknowns[1];
```

$$2\,s + 3\,t = 4$$

$$s$$

Funktionen

standard Funktionen

sqrt, exp, cos, abs, arctan, ...

Zusatzpakete einbinden:

> with(Paketname)

LinearAlgebra: Lineare Algebra

DEtools: Differentialgleichungen

VectorCalculus: Vektoranalysis

Definition einer Funktion $x \mapsto f(x)$

> f := x → Ausdruck

bivariate Funktion

> f := (x,y) → Ausdruck

Aufruf mit f(Variable) bzw. f(Variable1,Variable2)

Beispiel

```
> y:=sqrt(exp(sin(Pi)));
```

$$y := 1$$

```
> with(LinearAlgebra):
> A:=Matrix([[a,b],[c,d]]);
```

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

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

$$ad - bc$$

```
> f:=x->2*x+3;
```

$$f := x \mapsto 2x + 3$$

```
> {f(1), f(f(1))};
```

$$\{5, 13\}$$

```
> g:=(x,y)->f(x)*(p+y):
> g(1,2);
```

$$5\,p + 10$$

Manipulation von Ausdrücken

Vereinfachen, Ausmultiplizieren und Faktorisieren

simplify(Ausdruck), expand(Ausdruck), factor(Ausdruck)

Ordnen nach Potenzen von x

collect(Ausdruck,x)

Typenumwandlung

convert(Ausdruck,Typ)

z.B. binary, fraction, parfrac

Feststellung des Typs mit type

Beispiel

```
> simplify((8^x+4^x)/2^(3*x));
```

$$1 + 2^{-x}$$

```
> expand(cos(2*t)^2);
```

$$4\left(\cos\left(t\right)\right)^4 - 4\left(\cos\left(t\right)\right)^2 + 1$$

```
> factor(x^4-3*x^3-x+3);
```

$$\left(x-1\right)\left(x-3\right)\left(x^2+x+1\right)$$

```
> a:=x*y-x*y*y+x*x*y*z:
> [collect(a,x),collect(a,y)];
```

$$\left[x^2yz + \left(y - y^2\right)x, -xy^2 + \left(x + x^2z\right)y\right]$$

```
> convert(10,binary);
```

$$1010$$

```
> convert(1.2,fraction);
```

$$6/5$$

```
> convert(2*x/(1-x^2),parfrac);
```

$$-(x-1)^{-1}-(x+1)^{-1}$$

Lösen von Gleichungen

Gleichungen mit einer Unbekannten x

lsg := solve(eqn,x)

numerisch: fsolve(...)

Probe

> subs(x=lsg,eqn)

lsg[k] bei mehreren Lösungen

Nullstelle einer Funktion f

> solve(f(x)=0,x)

Systeme von Gleichungen

solve({ eqn1, eqn2, ... }, { x1, x2, ... })

Auflösen verbleibender RootOf Ausdrücke mit allvalues

Beispiel

```
> eqn:=x^2-2*p*x+9=0:
> lsg:=solve(eqn,x);
```

$$lsg := p + sqrt\left(p^2 - 9\right), p - sqrt\left(p^2 - 9\right)$$

```
> fsolve(sin(x)=exp(x));
```

$$-12.56636713$$

```
> assume(x::real): assume(y::real):
> solve({x^2+4*y^2=1,x=y^2-1},{x,y});
```

$$\{y = 0, x = -1\}, \{y = 0, x = -1\},$$
$$\left\{y = RootOf\left(\_Z^2 + 2, label = \_L1\right), x = -3\right\}$$

```
> allvalues(%[3]);
```

$$\{x = -3, y = isqrt(2)\}, \{x = -3, y = -isqrt(2)\}$$

Grenzwerte und Reihen

Grenzwert eines Ausdrucks A für $x \to a$

limit(A, x=a, option)

option: left, right, real

Summe bzw. Reihe eines Ausdrucks A

sum(A, k=kmin..kmax)

numerische Berechnung mit evalf

Beispiel

```
> limit(n/(1+p*n),n=infinity);
```

$$p^{-1}$$

```
> f:=x->x^100*exp(1/x):
> limit(f(x),x=0);
```

undefined

```
> limit(f(x),x=0,left);
```

$$0$$

```
> s:=sum(k^(-k),k=1..infinity);
```

$$s := \sum_{k=1}^{\infty} k^{-k}$$

```
> evalf(s,30);
```

$$1.29128599706266354040728259060$$

```
> int(x^(-x),x=0..1.0);
```

$$1.291285997$$

Differentiation

Ableitung einer Funktion f bzw. eines Ausdrucks A

D(f), diff(A,x)

n-te Ableitung

(D@@n)(f), diff(A,x $ n)

partielle Ableitung $\partial_1^{k_1}\partial_2^{k_2}\cdots$

D[1 $ k1, 2 $ k2, ...](f), diff(A,x1 $ k1, x2 $ k2, ...)

Beispiel

```
> f:=x->a/x:
> df:=D(f);
```

$$df := x \mapsto -\frac{a}{x^2}$$

```
> diff(f(x),x$2);
```

$$2\,\frac{a}{x^3}$$

```
> subs(x=1,%); df(1);
```

$$2\,a$$

$$-a$$

```
> (D@@2)(sin^2);
```

$$-2\sin^2 + 2\cos^2$$

```
> g:=(x,a)->a/x:
> D[1$3,2](g);
```

$$(x, a) \mapsto -6\,x^{-4}$$

Integration

bestimmtes Integral $\int_a^b A\,dx$ eines Ausdrucks A

```
int(A,x=a..b)
```

numerische Auswertung mit evalf

Voraussetzungen an Parameter mit assume

bei Funktionen f: $A = f(x)$

unbestimmtes Integral

```
int(A,x)
```

Doppelintegral $\int_a^b \int_c^d A\,dy\,dx$ eines Ausdrucks

```
int(int(A,y=c..d),x=a..b)
```

bei Funktionen f: $A = f(x, y)$

Beispiel

```
> int(p*x,x=1..b);
```

$$1/2\,p\left(b^2-1\right)$$

```
> s:=int(cos(sin(x)),x=1..Pi);
```

$$s := \int_1^{\pi} \cos\left(\sin\left(x\right)\right) dx$$

```
> evalf(s,30);
```

$$1.53519939105743698719032369739$$

```
> f:=x->x^p:
> int(f(x),x);
```

$$\frac{x^{p+1}}{p+1}$$

```
> assume(r::real): assume(r>1):
> int(int(x*y,y=0..sqrt(r^2-x^2)),x=0..1);
```

$$-1/8+1/4\,r^2$$

Taylor Entwicklung

Entwicklung eines Ausdrucks A um $x = a$ bis zur Ordnung n

```
taylor(A,x=a,n)
```

Default: $a = 0$, $n = 6$

Umwandlung in polynomialen Ausdruck mit convert

Einsetzen spezieller Werte mit subs

bivariate Entwicklung

```
mtaylor(A,[x=a,y=b],n)
```

Beispiel

```
> taylor(1/x,x=a,3);
```

$$a^{-1} - \frac{x-a}{a^2} + \frac{(x-a)^2}{a^3} + O\left((x-a)^3\right)$$

```
> p:=convert(%,polynom);
```

$$p := a^{-1} - \frac{x-a}{a^2} + \frac{(x-a)^2}{a^3}$$

```
> subs(a=2,x=1,p);
```

$$\frac{7}{8}$$

```
> mtaylor(x^y,[x=a,y=1],2);
```

$$x + a\,(y-1)\ln(a)$$

Grafik

Darstellung einer bzw. mehrerer Funktionen

plot(f)

plot({ f(x), g(x) }, x=a..b)

MOUSE-CLICK $\rightarrow$ Optionen, interaktive Gestaltung

Kurven

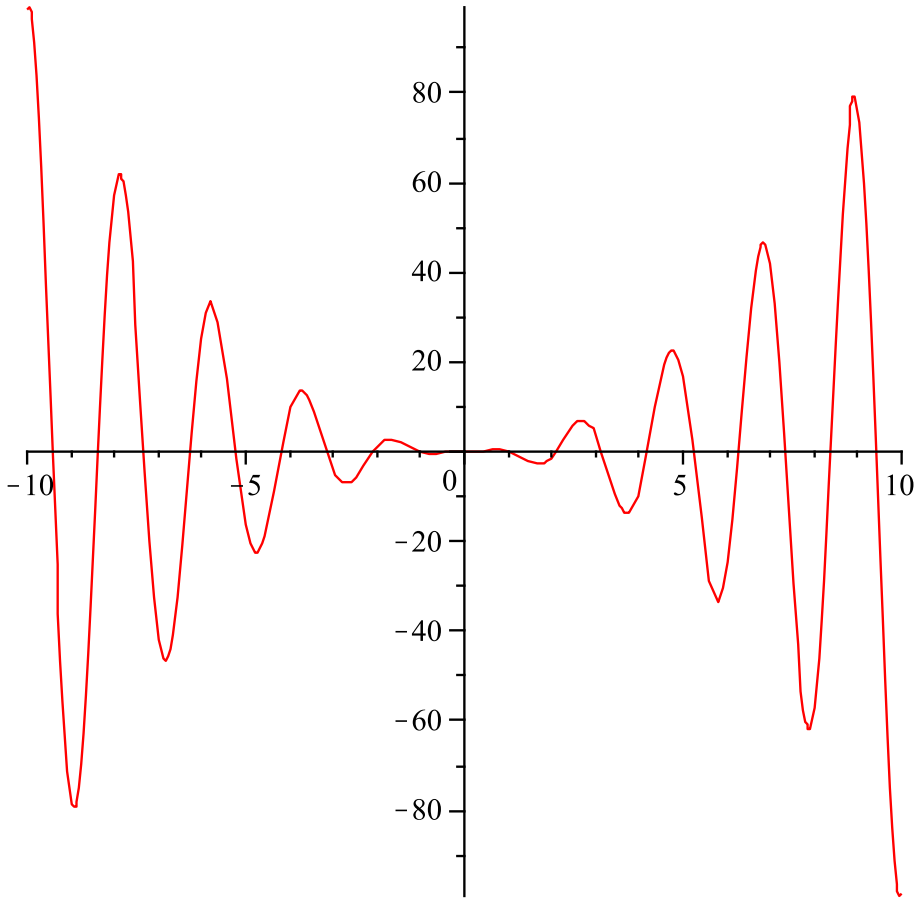
plot([x(t), y(t)], t=a..b)

parametrisierte Flächen

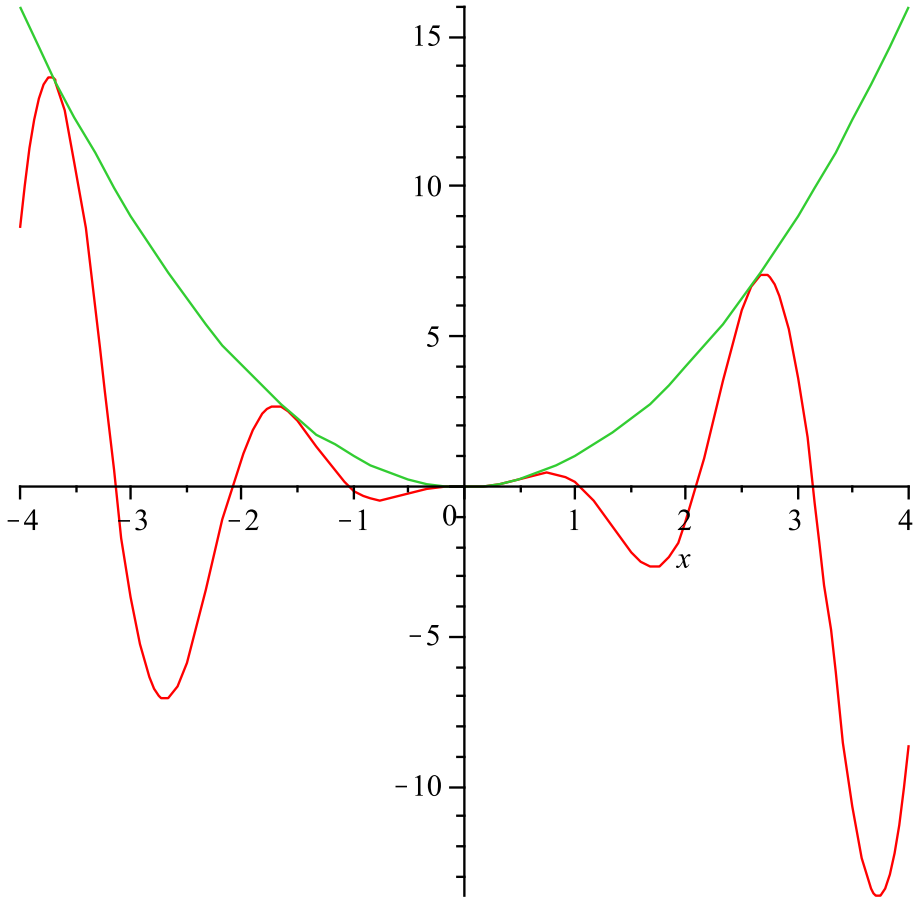
plot3D([x(u,v), y(u,v), z(u,v)], u=a..b, v=c..d)

Beispiel

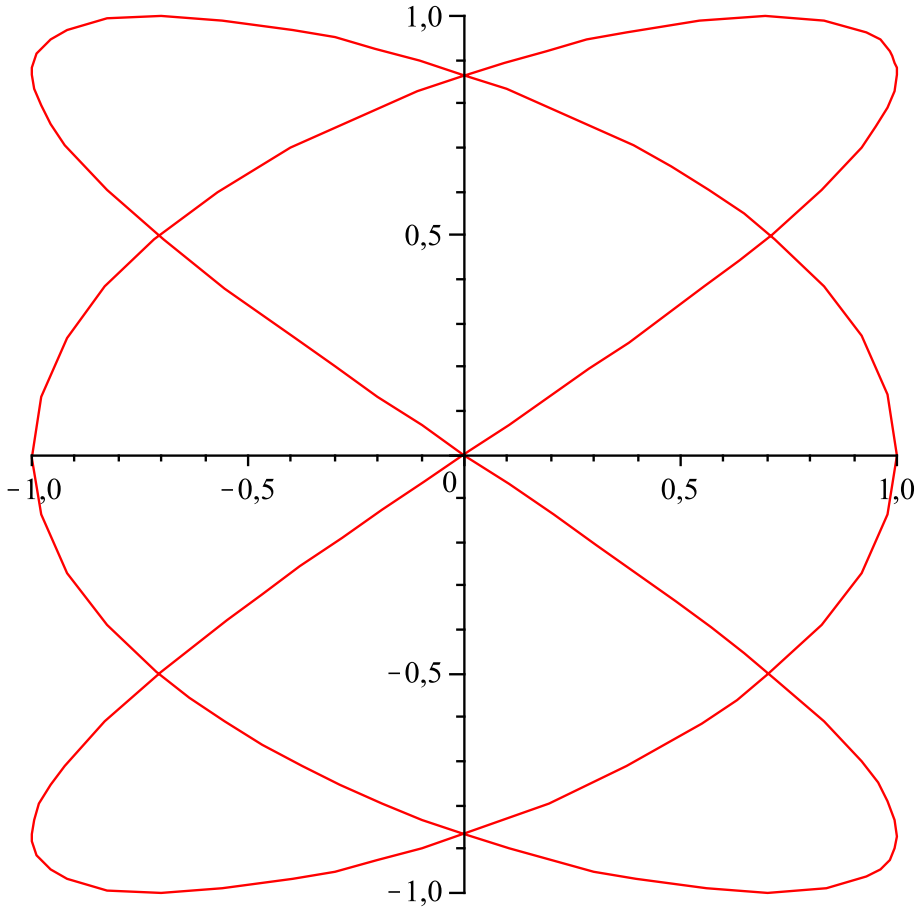
```
> f:=x->x^2*sin(3*x):
> plot(f);
```



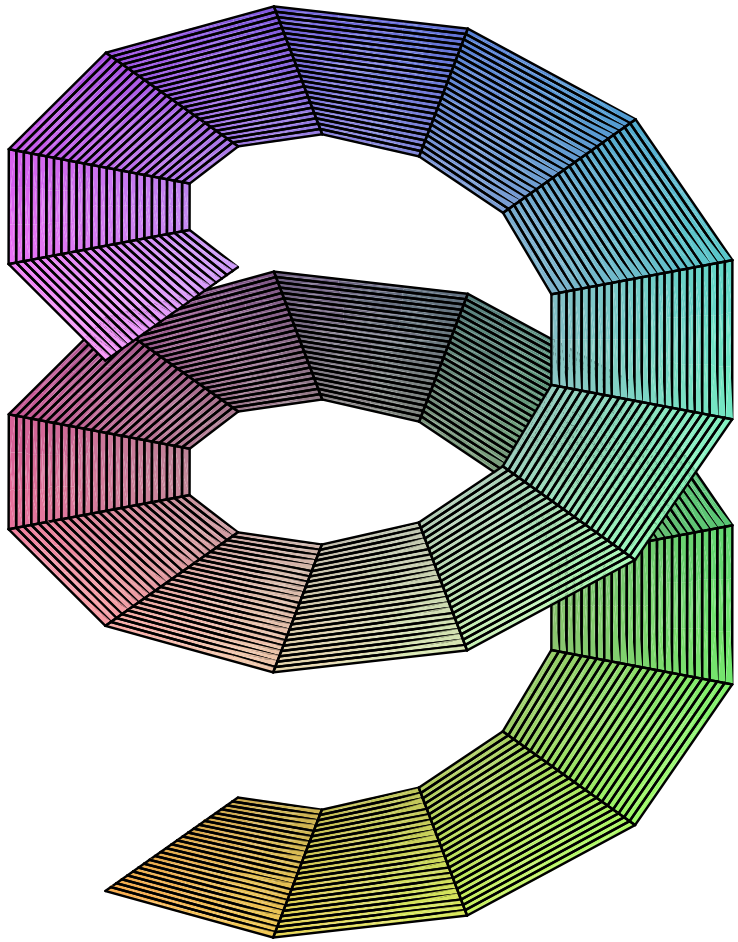
```
> plot({f(x),x^2},x=-4..4);
```



```
> plot([cos(3*t),sin(2*t),t=0..2*Pi]);
```



```
> x:=(r,t)->r*cos(t):
> y:=(r,t)->r*sin(t):
> z:=(r,t)->t:
> plot3d([x(r,t),y(r,t),z(r,t)],r=1..2,t=0..4*Pi);
```

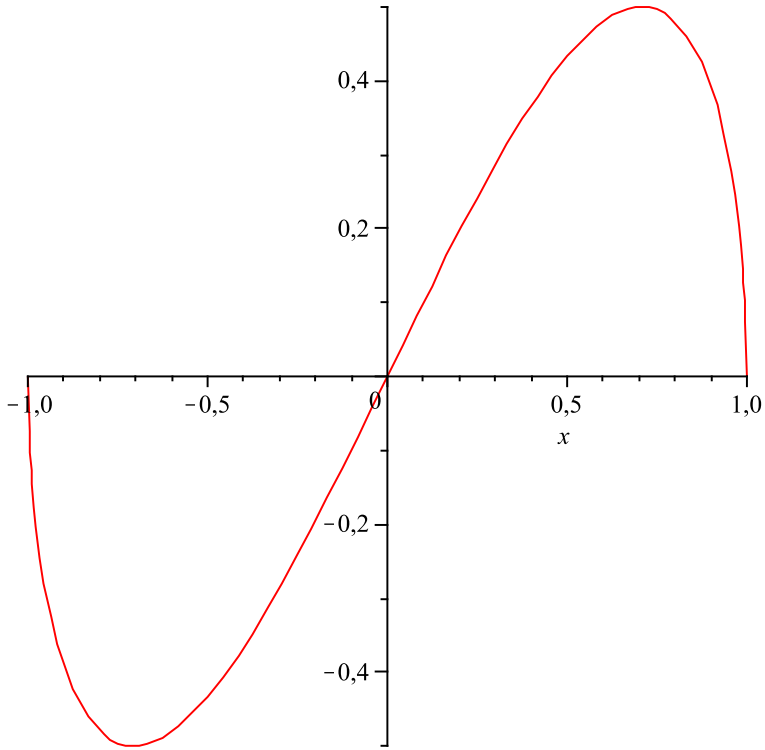


Beispiel

```
> f:=x->x*sqrt(1-x^2);
```

$$f := x \mapsto x\sqrt{1-x^2}$$

```
> plot(f(x),x=-1..1);
```



```
> df:=D(f):
> xe:=solve(df(x)=0,x);
```

$$xe := -1/2\,sqrt(2),\ 1/2\,sqrt(2)$$

```
> ddf:=D(df):
> xw:=solve(ddf(x)=0,x);
```

$$xw := 0,\ 1/2\,sqrt(6),\ -1/2\,sqrt(6)$$

```
> evalf(xw[2]);
```

$$1.224744872$$

Vektoren und Matrizen

Paket: LinearAlgebra

Definition von Vektoren

Vector[row]([x1, x2, ...])

<a,b,c> Spaltenvektor

<a|b|c> Zeilenvektor

Definition von $m \times n$-Matrizen

Matrix([[a11, ..., a1n], ..., [am1, ..., amn]])

Matrix(m, n, (j,k) $\rightarrow$ f(j,k))

<a,b,c;d,e,f> Matrix aus 2 Zeilen

<a,b,c|d,e,f> Matrix aus 2 Spalten

auch als Vektor von Vektoren möglich

einige Optionen

Nullvektor: Vector(n)

Matrix mit symbolischen Einträgen: Matrix(m, n, symbol=a)

Zugriff auf Elemente

x[k] bzw. A[j,k]

Zugriff auf Blöcke

A[j1..j2,k1..k2]

Beispiel

```
> with(LinearAlgebra):
> u:=<1,2,Pi,sqrt(3)>;
```

$$u := \begin{bmatrix} 1 \\ 2 \\ \pi \\ \sqrt{3} \end{bmatrix}$$

```
> v:=Vector[row](3,k->x^k);
```

$$v := \begin{bmatrix} x & x^2 & x^3 \end{bmatrix}$$

```
> {u[2],v[3]};
```

$$\{2, x^3\}$$

Beispiel

```
> A:=Matrix([[11, 12, 13],[21, 22, 23]]);
```

$$A := \begin{bmatrix} 11 & 12 & 13 \\ 21 & 22 & 23 \end{bmatrix}$$

```
> {A[1,2],A[2,3]};
```

$$\{12, 23\}$$

```
> M:=Matrix(2,3,symbol=m);
```

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

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

$$m_{2,1}$$

Rechnen mit Matrizen

Paket: LinearAlgebra

elementare Operationen

$\pm$, MatrixMatrixMultiply, MatrixVectorMultiply, VectorMatrixMultiply

einige Funktionen

- transponierte Matrix: transpose
- inverse Matrix: MatrixInverse
- Eigenwerte: Eigenvalues
- Jordan-Normalform: JordanForm
- Singulärwerte: SingularValues

Beispiel

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

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

```
> b:=Vector([x,y]):
> MatrixMatrixMultiply(A,A);
```

$$\begin{bmatrix} 2 & 3 \\ 6 & 11 \end{bmatrix}$$

```
> MatrixVectorMultiply(A,b);
```

$$\begin{bmatrix} y \\ 2x + 3y \end{bmatrix}$$

Beispiel

```
> MatrixInverse(A);
```

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

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

$$\begin{bmatrix} 3/2 + 1/2\sqrt{17} \\ 3/2 - 1/2\sqrt{17} \end{bmatrix}$$

```
> JordanForm(A);
```

$$\begin{bmatrix} 3/2 + 1/2\sqrt{17} & 0 \\ 0 & 3/2 - 1/2\sqrt{17} \end{bmatrix}$$

```
> SingularValues(A);
```

$$\begin{bmatrix} 3/2\sqrt{2} + 1/2\sqrt{10} \\ 3/2\sqrt{2} - 1/2\sqrt{10} \end{bmatrix}$$

Lineare Gleichungssysteme

quadratische Systeme

- LinearSolve(A,b): löst $Ax = b$
- GaussianElimination(W): Transformation auf obere Dreiecksform
- BackwardSubstitute($<$ A $|$ b $>$): löst System in oberer Dreiecksform

allgemeine Systeme

- ReducedRowEchelonForm(A): Echelonform von A
- NullSpace(A): Basis für $\ker A$

Beispiel

```
> with(LinearAlgebra):
> A:=<<1|2>,<3|r>>; b:=Vector(2,symbol=s);
```

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

$$b := \begin{bmatrix} s_1 \\ s_2 \end{bmatrix}$$

```
> LinearSolve(A,b);
```

$$\begin{bmatrix} \frac{-2\,s_2+s_1\,r}{-6+r} \\ -\frac{3\,s_1-s_2}{-6+r} \end{bmatrix}$$

```
> GaussianElimination(<A|b>);
```

$$\begin{bmatrix} 1 & 2 & s_1 \\ 0 & -6+r & s_2-3\,s_1 \end{bmatrix}$$

Beispiel

```
> A:=Matrix(3,5,(j,k)->j+k);
```

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

```
> E:=ReducedRowEchelonForm(A);
```

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

```
> BackwardSubstitute(E,free=t);
```

$$\begin{bmatrix} -3 + t_2 + 2\,t_1 \\ 4 - 2\,t_2 - 3\,t_1 \\ t_2 \\ t_1 \end{bmatrix}$$

Kontrollanweisungen

Schleifen

```
for k from kmin to kmax do
    Befehle
end do

while logischer Ausdruck do
    Befehle
end do
```

Verzweigungen

```
if logischer Ausdruck then
    Befehle
elif logischer Ausdruck then
    Befehle
else
    Befehle
end if
```

Beispiel

```
> # loops
> squares:={}:
> for k from 1 to 9 do
> squares:=squares union {k^2}
> od:
> squares;
```

$$\{1, 4, 9, 16, 25, 36, 49, 64, 81\}$$

```
> fibonacci:=[1,1]: n:=2:
> while fibonacci[2]<100 do
> fibonacci:=[fibonacci[2],fibonacci[1]+fibonacci[2]]
> od:
> fibonacci;
```

$$[89, 144]$$

Beispiel

```
> # branching
> s:=rand(-1..1): s:=s();
```

$$s := -1$$

```
> if s>0 then
> print(positive)
> elif s<0 then
> print(negative)
> else
> print(zero)
> fi;
```

negative

Prozeduren

Syntax

```
name := proc(Parameterfolge)
local Variablenfolge;
global Variablenfolge;
    Befehle
return Ergebnis;
end proc;
```

Speichern/Laden mit save name "Dateiname" bzw. read "Dateiname"

Beispiel

```
newton := proc (f, x0, s)
  local x;
  x := x0;
  # Schleife bis Funktionswert klein genug
  while 10^(-s) < abs(f(x)) do
    x :=x-f(x)/D(f)(x); # Newton-Schritt
    print(evalf(x,s));  # Ausgabe der neuen Näherung
  end do;
end proc;
```

```
> read "newton":
> f := x -> sin(x)-exp(-x):
> newton(f, 5.0, 6);
```

8.32528

10.2881

9.11858

9.43464

9.42470

Beispiel

```
> # define function
> bisect:=proc(f,interval)
>    local a,b,c,fa,fb,fc:
>    global TOL:
>    a:=interval[1]: b:=interval[2]:
>    fa:=f(a): fb:=f(b):
>    while TOL<(b-a) do
>       c:=(a+b)/2: fc:=f(c):
>       if 0<fa*fc then
>          a:=c: fa:=fc:
>       else
>          b:=c: fb:=fc:
>       end if:
>    end do:
>    return c:
> end proc:
```

Beispiel

```
> # define test data
> TOL:=0.0001:
> f:=x->x^2-1;
```

$$f := x \mapsto x^2 - 1$$

```
> interval:=[0,3]:
> x:=bisect(f,interval);
```

$$x := \frac{32769}{32768}$$

```
> interval:=[0,3.0]:
> x:=bisect(f,interval);
```

$$x := 1.000030517$$

```
> save bisect, "procedures":
```