Maple Arbeitsblätter

N. Geers

Rechenzentrum

Universität Karlsruhe (TH)

e-mail: geers@rz.uni-karlsruhe.de

Einige einfache Maple Beispiele

N. Geers

Rechenzentrum

Universität Karlsruhe (TH)

e-mail: geers@rz.uni-karlsruhe.de

```
> restart;
```

Durch das Kommando restart wird Maple neu initialisiert, u.a. werden alle Variablen gelöscht.

Erstes Beispiel: Polynombestimmung

Bestimme ein Polynom zweiten Grades

$p : x \mapsto a x^2 + b x + c$ **mit**

$p(1) = 1$ **,**

$p'(1) = 2$ **und**

$p''(1) = 3$

```
> p := x -> a * x^2 + b * x + c;
```

$$p := x \mapsto a x^2 + b x + c$$

```
> bedingungen := {p(1)=1, D(p)(1) = 2, (D@@2)(p)(1) = 3};
```

$$bedingungen := \{2a = 3, 2a + b = 2, a + b + c = 1\}$$

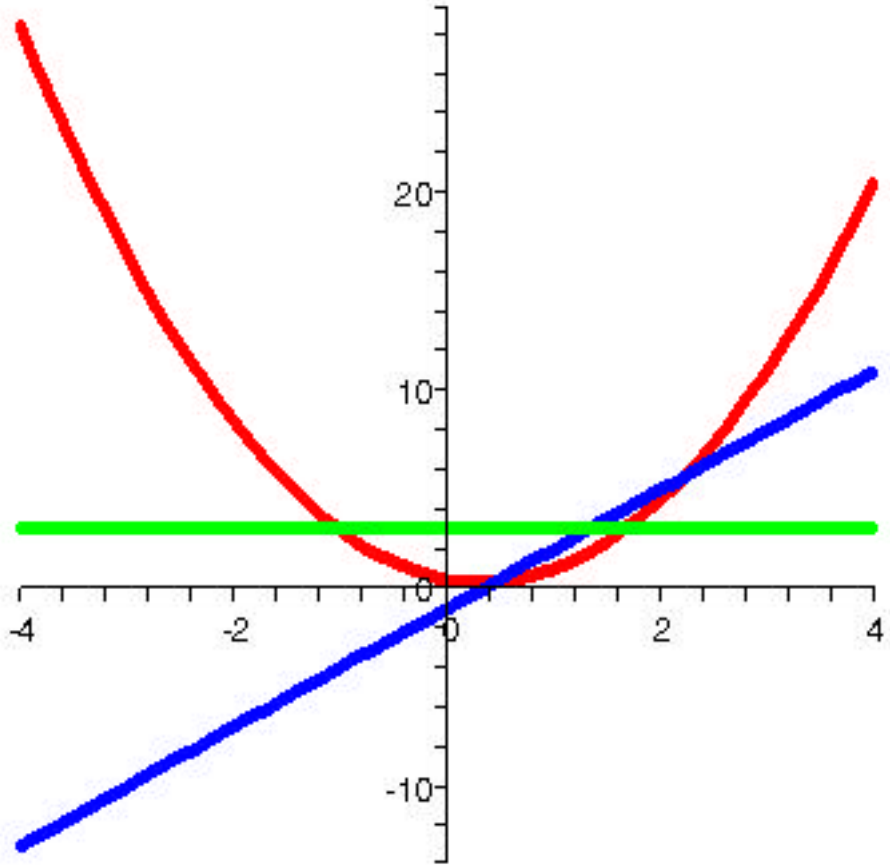
```
> loesung := solve(bedingungen);
```

$$loesung = \left\{ b = -1, c = \frac{1}{2}, a = \frac{3}{2} \right\}$$

```
> assign(%);
  p(x);
```

$$\frac{3}{2} x^2 - x + \frac{1}{2}$$

```
> plot([p,D(p),(D@@2)(p)], -4 ..4, thickness=3, color=[red, blue,
  green]);
```



Wie man an diesem einfachen Beispiel aus der Schulmathematik sieht, kann man mit einem Computeralgebrasystem wie Maple symbolische Umformungen, wie wir sie aus vielen Bereichen der Mathematik kennen, in einfacher Weise durchführen. Wir rechnen also nicht nur mit Zahlen, wie es sonst beim Einsatz des Computers zur Lösung mathematisch technischer Probleme üblich ist, sondern führen Operationen mit mathematischen Objekten:

- **Varibalen,**
- **Ausdrücken,**
- **Gleichungen,**
- **Ungleichungen,**
- **Funktionen,**
- **Matrizen,**
- **Mengen**
- **etc.**

aus.

Maple deckt dabei viele Teilgebiete der Mathematik und anderer Bereiche in Technik und Wissenschaft ab und hat sich seit mehreren Jahren als Werkzeug in Forschung und Lehre bewährt.

>

Einfache Ausdrücke und Anweisungen

> restart;

Abschluss einer Anweisung

Jede Anweisung wird durch ein Semikolon ';' oder einen Doppelpunkt ':' abgeschlossen.

Bei Abschluss der Eingabe mit ';' wird das Ergebnis angezeigt, andernfalls unterbleibt die Anzeige des Ergebnisses.

> x := sin(Pi);

$$x := 0$$

> x := sin(Pi):

Groß-/Kleinschreibung ist signifikant!

> y := sin(pi);

$$y := \sin(\mathrm{p})$$

>

Zahldarstellung

> restart;

Bei der Eingabe ist immer zu unterscheiden, ob eine Zahl mit oder ohne Dezimalpunkt eingegeben wird.

> a := 1/3;

$$a := \frac{1}{3}$$

> b := 1/3.;

$$b := 0.3333333333$$

> 3 * b; 3 * a;
4 * a;

$$0.9999999999$$

$$1$$

$$\frac{4}{3}$$

Falls eine Zahl mit Dezimalpunkt angegeben wird, wird diese Zahl intern als Dezimalzahl mit vorgegebener Stellenzahl gespeichert.

Standardmässig wird mit 10 Dezimalstellen gerechnet.

Über die Variable Digits kann die Genauigkeit der Rechnung verändert werden.

Digits := ***stellenzahl***

```
> Digits:=100;
```

$$Digits := 100$$

```
> b:=1/3.;
```

$$b := 0.33333333333333333333333333333333333333333333333333333333333333333333333333333333333333333333333333\
33333333333333333333$$

```
> 3*b - 1;
```

$$-1\ 10^{-100}$$

```
> Digits:=10;
```

$$Digits := 10$$

Zahlen, die ohne Dezimalpunkt eingegeben werden, werden nicht durch einen endlichen Dezimalbruch approximiert. So steht 1/3 für die rationale Zahl 'Ein Drittel', während 1./3. einen endlichen (Digits Stellen langen) Dezimalbruch darstellt. Die arithmetischen Operationen für rationale Zahlen werden in Maple mit 'beliebig' hoher Genauigkeit, d.h. ohne Rundungsfehler ausgeführt, während beim Rechnen mit Approximationen von reellen Zahlen Rundungsfehler auftreten, wie auch sonst in der Computerarithmetik.

Das folgende Beispiel verdeutlicht diese Problematik:

```
> x1 := 2**35 - (2**35 -1);
> x2 := 2.**35 - (2.**35-1);
```

$$x1 := 1$$

$$x2 := 0.$$

```
>
```

Operatoren

```
> restart;
```

Die wichtigsten Operatoren in Maple sind:

Operator	Funktion	Beispiele

+ Addition x + y oder z + 3
- Subtraktion x - a oder 5 - y
* Multiplikation x * y oder 32 * a
/ Division x / y oder 1 / 3
^ bzw. ** Exponentiation x ^ 5 oder a ^ b
! Fakultät n! oder 100!
= Gleichung y = 2 * x^2 - 4*x + a
|| Konkatenation x||i

Klammern werden so verwendet, wie es in der Mathematik üblich ist.
s.a. ?operator

```
> (x + 5) * (y-3);
```

$$(x+5)\,(y-3)$$

```
> (x^4 - y**4) / (x-y);
```

$$\frac{x^4 - y^4}{x - y}$$

```
> (x-2)**2 + (y/2-1)**2 = 4;
```

$$(x-2)^2 + \left(\frac{1}{2}y - 1\right)^2 = 4$$

```
> i:=5:
  x||i;
```

$$x5$$

```
>
```

Mathematische Funktionen

```
> restart;
```

Maple kennt sehr viele mathematische Funktionen, u.a. Exponentialfuntionen, trigonometrische Funktionen, Bessel-Funktionen ...
Eine Übersicht erhält man mit ?initialfunction

```
> sin(x^2)^2;
```

$$\sin(x^2)^2$$

```
> sin(alpha)^2 + cos(beta)^2;
```

$$\sin(a)^2 + \cos(b)^2$$

```
>
```

Zuweisungen

```
> restart;
```

Zuweisungen werden in Maple in einer ALGOL- bzw. Pascal-ähnlichen Syntax angegeben:

name*:= *ausdruck

Namen, die Sonderzeichen enthalten, werden in ` eingeschlossen.

```
> z:= (x^4 -y**4) / (x-y);
```

$$z := \frac{x^4 - y^4}{x - y}$$

```
> Kreis_Umfang := 2 * Pi * r;
  Kreis_Flaeche := Pi * r^2;
```

$$Kreis\_Umfang = 2\,\pi\,r$$

$$Kreis\_Flaeche = \pi\,r^2$$

```
> r := a+b;
  Kreis_Umfang;
```

$$r := a + b$$

$$2\,\pi\,(a + b)$$

```
> r := 10;
  Kreis_Umfang;
  Kreis_Flaeche;
```

$$r := 10$$

$$20\,\pi$$

$$100\,\pi$$

Verschiedene Namen sind in Maple bereits vordefiniert:

z.B. Pi bezeichnet die Konstante 3.14159265358979...

I ist die komplexe Einheit

Eine Übersicht aller anfangs definierten Namen erhält man mit ?ininames

```
>
```

Solange einem Namen noch kein Wert zugewiesen wurde, steht dieser Name für sich selbst. Durch die Zuweisung

name*:= ' *name

werden die Zuweisungen an die entsprechende Variable gelöscht.

```
> z:= 'z';
```

$$z := z$$

```
> r := 10;
```

$$r := 10$$

Die Eingabe eines Namens liefert den aktuellen Wert, der entsprechenden Variablen. Dazu wird implizit die Maple Funktion eval aufgerufen.

```
> r;
```

$$10$$

```
> Kreis_Umfang;
```

$$20\,\mathrm{p}$$

```
> r:='r';
```

$$r := r$$

```
> Kreis_Umfang;
```

$$2\,\mathrm{p}\,r$$

Einige häufige Fehler:

Falls bei einer Zuweisung das Zeichen ':' fehlt, betrachtet Maple die Zuweisung als Gleichung:

```
> a = sin(x)*cos(y)/(x*y);
```

$$a = \frac{\sin(x)\cos(y)}{x\,y}$$

Bei einer Zuweisung wird zunächst der Ausdruck auf der rechten Seite von := ausgewertet und dieser Wert wird an die Variable zugewiesen. Um diese Auswertung zu verhindern, kann man den Ausdruck in Hochkomma (') einschliessen (s. Definition der Variablen c).

```
> restart;
  a := x + y;
  x := 5; y := 6;
  b := x + y;
  c := 'x + y';
```

$$a := x + y$$

$$x := 5$$

$$y := 6$$

$$b := 11$$

$$c := x + y$$

- **Der Variablen a wird der Ausdruck x+y zugewiesen, da bei Ausführung der Anweisung a:= x+y die Variablen x und y noch undefiniert sind.**

- **Der Variablen b wird der Zahlenwert 11 zugewiesen. Zuerst wird der Ausdruck auf der rechten Seite (x+y) ausgewertet. Dies ergibt 11 und dieser Wert wird an die Variable y zugewiesen.**
- **Die Variable c hat als Wert den Ausdruck x+y, da der Ausdruck auf der rechten Seite nicht ausgewertet worden ist.**

```
> x := 20; y:= 20;
  a; b; c;
```

$$x := 20$$
$$y := 20$$
$$40$$
$$11$$
$$40$$

Bitte beachten Sie unterschiedlichen Werte von a, b und c in der letzten Ausgabe!

```
>
```

Einfache Umformungen von Ausdrücken

```
> restart;
```

Einige Maple Funktionen zum Bearbeiten von Ausdrücken:

- **Vereinfachen eines Ausdrucks: simplify oder normal**
- **Ausmultiplizieren eines Ausdrucks: expand**
- **Ausklammern von Teilausdrücken: factor**

simplify

```
> z:= (x^4 -y**4) / (x-y);
```

$$z := \frac{x^4 - y^4}{x - y}$$

```
> simplify(z);
```

$$x^3 + y x^2 + y^2 x + y^3$$

Eine vereinfachte Form des Wertes von z wird ausgegeben. Die Variable z bleibt aber unverändert, wie die Eingabe von

z;

zeigt. Um den Inhalt der Variablen z zu verändern, ist die Anweisung

z := simplify(z);

notwendig.

```
> z;
```

$$\frac{x^4 - y^4}{x - y}$$

```
> z:= simplify(z);
```

$$z := x^3 + y x^2 + y^2 x + y^3$$

simplify kennt auch Vereinfachungsregeln für Funktionen, die in Maple definiert sind.

```
> simplify(sin(alpha)^2 -1);
```

$$-\cos(a)^2$$

```
> ausdruck := exp(a+ln(b*exp(c)));
```

$$ausdruck := \mathbf{e}^{\left(a + \ln\left(b\,\mathbf{e}^{c}\right)\right)}$$

```
> simplify(ausdruck);
```

$$b\,\mathbf{e}^{(a + c)}$$

```
>
```

expand

```
> expand((x+1)*(x+2));
```

$$x^2 + 3x + 2$$

```
> expand((x^4 -y**4) / (x-y));
```

$$\frac{x^4}{x - y} - \frac{y^4}{x - y}$$

```
> expand((a+b)^2);
```

$$a^2 + 2ab + b^2$$

```
> expand(sin(x+y));
```

$$\sin(x)\cos(y) + \cos(x)\sin(y)$$

```
> expand((x+1)*(y+v), x+1);
```

$$(x+1)y + (x+1)v$$

```
>
```

factor

```
> factor(6*x^2+18*x-24);
```

$$6(x+4)(x-1)$$

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

$$(x - y)\,(x + y)^2$$

```
> factor(x^4-y^4);
```

$$(x - y)\,(x + y)\,(x^2 + y^2)$$

```
> restart;
```

assume

```
> sqrt(x^2);
```

$$\sqrt{x^2}$$

$\sqrt{x^2}$ **kann nicht vereinfacht werden, da keine zusätzlichen Informationen über x vorliegen. x könnte z.B. eine komplexe Zahl sein. Falls man z.B. weiss, dass x eine negative reele Zahl ist, kann dies mit der Funktion assume festgelegt werden. (s.a. ?assume)**

```
> assume(x<0);
  sqrt(x^2);
```

$$-x\sim$$

Die Tilde (~) deutet an, dass für die Variable x bestimmte Eigenschaften definiert sind. Die Eigenschaften können mit den Maple Funktionen is bzw. about abgefragt werden.

```
> about(x);
  is(x, real);
Originally x, renamed x~:
  is assumed to be: RealRange(-infinity,Open(0))
```

$$true$$

Eigenschaften, die mittels assume definiert wurden, werden jedoch nicht von allen Maple Funktionen berücksichtigt.

```
> restart;
```

Auswerten von Ausdrücken

- **eval wird i.a. implizit aufgerufen, wenn Ergebnisse ausgegeben werden. Ein Aufruf von eval ist i.a. nur innerhalb von Prozeduren notwendig.**
- **evalf berechnet einen numerischen Wert (f = floating point) mit Digits Stellen.**
- **evalhf berechnet einen numerischen Wert, wobei die Arithmetik des Prozessors direkt benutzt wird, d.h. evalhf ist i.a. schneller als evalf, rechnet aber immer mit ca. 15 Dezimalstellen.**

- **evalm wertet Matrix-Ausdrücke aus.**

evalf

```
> evalf(Pi);
```

$$3.141592654$$

```
> evalf(Pi,50);
```

$$3.1415926535897932384626433832795028841971693993751$$

```
> Digits:=50;
```

$$Digits := 50$$

```
> evalf(Pi);
```

$$3.1415926535897932384626433832795028841971693993751$$

```
> Digits:=10;
```

$$Digits := 10$$

evalhf

```
> evalhf(sqrt(2));
```

$$1.41421356237309515$$

```
> evalf(sqrt(2));
```

$$1.414213562$$

```
> restart;
```

Differenzieren und Integrieren

- **diff bestimmt die Ableitung eines Funktionsterms. diff hat als Argument einen Ausdruck. Wenn die Ableitungsfunktion zu einer gegebenen Funktion berechnet werden soll, verwendet man den Operator D (s.u.)**
- **Diff dient zur Beschreibung der Ableitung, wird jedoch i.a. von Maple nicht ausgewertet.**
- **int berechnet ein unbestimmtes oder bestimmtes Integral**
- **Int dient wiederum zur Beschreibung des Integrals, wird jedoch ebenfalls von Maple i.a. nicht ausgewertet.**

Differentiation

```
> diff(sin(x),x);
```

$$\cos(x)$$

```
> u := exp(x^2/s)*x/(2*Pi);
```

$$u := \frac{e^{\left(\frac{x^2}{s}\right)}}{2\,p}\,x$$

```
> diff(u,x);
```

$$\frac{x^2\,e^{\left(\frac{x^2}{s}\right)}}{s\,p} + \frac{e^{\left(\frac{x^2}{s}\right)}}{2\,p}$$

```
> Diff('u',x) = diff(u,x);
```

$$\frac{\partial}{\partial x}u = \frac{x^2\,e^{\left(\frac{x^2}{s}\right)}}{s\,p} + \frac{e^{\left(\frac{x^2}{s}\right)}}{2\,p}$$

```
> Diff(u,x) = diff(u,x);
```

$$\frac{\partial}{\partial x}\left(\frac{e^{\left(\frac{x^2}{s}\right)}}{2\,p}\,x\right) = \frac{x^2\,e^{\left(\frac{x^2}{s}\right)}}{s\,p} + \frac{e^{\left(\frac{x^2}{s}\right)}}{2\,p}$$

```
> dfy := diff(x^2*cos(y),y);
```

$$dfy := -x^2\sin(y)$$

```
> Diff(f,y,x) =  diff(x^2*cos(y),y,x);
```

$$\frac{\partial^2}{\partial y\,\partial x}f = -2\,x\sin(y)$$

```
> f := x^2*cos(y);
  Diff('f',y,x) = diff(f,y,x);
```

$$f := x^2\cos(y)$$

$$\frac{\partial^2}{\partial y\,\partial x}f = -2\,x\sin(y)$$

```
>
```

Integration

```
> restart;
```

symbolische Integration

Berechnung einer Stammfunktion

```
> int( sin(x), x );
```

$$-\cos(x)$$

```
> y:=(x^3-2*x^2-1)/(x+2)^2;
  int(y,x);
```

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

$$\frac{1}{2}x^2 - 6x + \frac{17}{x+2} + 20\ln(x+2)$$

Berechnung eines bestimmten Integrals

```
> int(x^2,x=-2..2);
```

$$\frac{16}{3}$$

Berechnung eines uneigentlichen Integrals

```
> Int(exp(-x),x=0..infinity) = int(exp(-x),x=0..infinity);
```

$$\int_0^{\infty} e^{(-x)}\,dx = 1$$

```
>
```

numerische Integration (s.a. ?int[numeric])

```
> int( exp(sin(x^2)), x =0..2);
```

$$\int_0^2 e^{\sin(x^2)}\,dx$$

Dieses Integral kann von Maple nicht symbolisch berechnet werden.
Die numerische Integration wird durch
evalf(int(...)) bzw. evalf(Int(...))
angestoßen.

```
> y:= evalf(Int(          exp(sin(x^2))      , x =0..2));
```

$$y := 3.276702922$$

```
>
```

Funktionen

```
> restart;
```

Definition von Funktionen

Eine Funktion $f : x \mapsto f(x)$ **wird in Maple folgendermaßen definiert:**

name*:= *variable* -> *funktionsterm

oder

name*:= (*variablenliste*) -> *funktionsterm

unapply erzeugt zu einem Ausdruck eine Funktion

```
> f := x -> (x^4-2*x^3+3*x)/(2*x^2-4*x);
```

$$f := x \mapsto \frac{x^4 - 2x^3 + 3x}{2x^2 - 4x}$$

```
> f(5); f(0.5);
```

$$\frac{13}{-.8750000000}$$

```
> g:= (x,y,z) -> sin(x)*cos(y)/(2*z^2);
```

$$g := (x, y, z) \mapsto \frac{\sin(x)\cos(y)}{2z^2}$$

```
> g(2,1,3);
```

$$\frac{1}{18}\sin(2)\cos(1)$$

```
> A := Pi * r^2;
  a := unapply(A,r);
  a(2);
```

$$A := \pi r^2$$

$$a := r \mapsto \pi r^2$$

$$4\pi$$

```
>
```

Ableitung einer Funktion

Die Berechnung der Ableitungsfunktion f' geschieht mit Hilfe des Operators D , jedoch

nicht mit der Funktion diff. diff hat als Argument einen Ausdruck und liefert als Ergebnis einen Ausdruck jedoch keine Funktion.

```
> fstrich := D(f);
  `f'` := D(f);
```

$$fstrich := x \mapsto \frac{4x^3 - 6x^2 + 3}{2x^2 - 4x} - \frac{(x^4 - 2x^3 + 3x)(4x - 4)}{(2x^2 - 4x)^2}$$

$$f' := x \mapsto \frac{4x^3 - 6x^2 + 3}{2x^2 - 4x} - \frac{(x^4 - 2x^3 + 3x)(4x - 4)}{(2x^2 - 4x)^2}$$

Partielle Ableitungen:

```
> g:= (x,y,z) -> sin(x)*cos(y)/(2*z^2);
  dgdy := D[2](g);
  dgdxdz := D[1,3](g);
```

$$g := (x, y, z) \mapsto \frac{\sin(x)\cos(y)}{2z^2}$$

$$dgdy := (x, y, z) \mapsto -\frac{\sin(x)\sin(y)}{2z^2}$$

$$dgdxdz := (x, y, z) \mapsto -\frac{\cos(x)\cos(y)}{z^3}$$

```
> dgdy(1,2,3);
  dgdxdz(2,5,6);
```

$$-\frac{1}{18}\sin(1)\sin(2)$$

$$-\frac{1}{216}\cos(2)\cos(5)$$

```
>
```

Lösen von Gleichungen

```
> restart;
```

- **solve löst eine oder mehrere Gleichungen**
- **dsolve löst Differentialgleichungen**
- **fsolve liefert eine numerische Lösung**

- **isolve bestimmt ganzzahlige Lösungen einer Gleichung**
- **msolve löst Matrixgleichungen**
- **rsolve löst Rekursionsgleichungen**

Symbolisches Lösen von Gleichungen und Gleichungssystemen

Gleichung mit numerischen Koeffizienten:

$$x^3 - 6x^2 + 11x - 6 = 0$$

```
> solve(x^3 - 6*x^2 + 11*x - 6=0, x);
```

$$1, 2, 3$$

Gleichung mit symbolischen Koeffizienten:

$$x^3 - a x^2 + 11 x - 6 = 0$$

```
> gleichung:= x^3 - a*x^2 + 11*x - 6=0;
```

$$gleichung = x^3 - a x^2 + 11 x - 6 = 0$$

```
> l:=solve(gleichung, x);
```

$$l :=$$

$$\frac{1}{6}\left(-396a + 648 + 8a^3 + 12\sqrt{18888 - 363a^2 - 3564a + 72a^3}\right)^{(1/3)}$$

$$- \frac{6\left(\frac{11}{3} - \frac{1}{9}a^2\right)}{\left(-396a + 648 + 8a^3 + 12\sqrt{18888 - 363a^2 - 3564a + 72a^3}\right)^{(1/3)}} + \frac{1}{3}a,$$

$$-\frac{1}{12}\left(-396a + 648 + 8a^3 + 12\sqrt{18888 - 363a^2 - 3564a + 72a^3}\right)^{(1/3)}$$

$$+ \frac{3\left(\frac{11}{3} - \frac{1}{9}a^2\right)}{\left(-396a + 648 + 8a^3 + 12\sqrt{18888 - 363a^2 - 3564a + 72a^3}\right)^{(1/3)}} + \frac{1}{3}a$$

$$+ \frac{1}{2}I\sqrt{3}\left(\frac{1}{6}\left(-396a + 648 + 8a^3 + 12\sqrt{18888 - 363a^2 - 3564a + 72a^3}\right)^{(1/3)}\right.$$

$$+\frac{6\left(\frac{11}{3}-\frac{1}{9}a^2\right)}{\left(-396a+648+8a^3+12\sqrt{18888-363a^2-3564a+72a^3}\right)^{(1/3)}}+\left(\frac{-396a+648+8a^3+12\sqrt{18888-363a^2-3564a+72a^3}}{12}\right)^{(1/3)},$$

$$+\frac{3\left(\frac{11}{3}-\frac{1}{9}a^2\right)}{\left(-396a+648+8a^3+12\sqrt{18888-363a^2-3564a+72a^3}\right)^{(1/3)}}+\frac{1}{3}a$$

$$-\frac{1}{2}I\sqrt{3}\left(\frac{1}{6}\left(-396a+648+8a^3+12\sqrt{18888-363a^2-3564a+72a^3}\right)^{(1/3)}\right.$$

$$\left.+\frac{6\left(\frac{11}{3}-\frac{1}{9}a^2\right)}{\left(-396a+648+8a^3+12\sqrt{18888-363a^2-3564a+72a^3}\right)^{(1/3)}}\right)$$

Einsetzen $a = 6$

```
> l_6 := subs(a=6, [l]);
```

$$l\_6 := \left[\frac{1}{6}12^{(1/3)}(-12)^{(1/6)}-\frac{1}{72}12^{(2/3)}(-12)^{(5/6)}+2,\right.$$

$$-\frac{1}{12}12^{(1/3)}(-12)^{(1/6)}+\frac{1}{144}12^{(2/3)}(-12)^{(5/6)}+2$$

$$+\frac{1}{2}I\sqrt{3}\left(\frac{1}{6}12^{(1/3)}(-12)^{(1/6)}+\frac{1}{72}12^{(2/3)}(-12)^{(5/6)}\right),$$

$$-\frac{1}{12}12^{(1/3)}(-12)^{(1/6)}+\frac{1}{144}12^{(2/3)}(-12)^{(5/6)}+2$$

$$\left.-\frac{1}{2}I\sqrt{3}\left(\frac{1}{6}12^{(1/3)}(-12)^{(1/6)}+\frac{1}{72}12^{(2/3)}(-12)^{(5/6)}\right)\right]$$

Dieser Ausdruck kann mit simplify vereinfacht werden:

```
> simplify(l_6);
```

$$[3, 1, 2]$$

Oder einfacher: subs und simplify in einem Schritt:

```
> simplify(subs(a=6,[l]));
```

$$[3, 1, 2]$$

Falls eine Gleichung der Form

***ausdruck* = 0**

gelöst werden soll, ist es ausreichend, den Ausdruck als Parameter an die Funktion solve zu übergeben.

```
> solve(x^3 - 6*x^2 + 11*x - 6,x);
```

$$1, 2, 3$$

Auch Gleichungssysteme können mit solve gelöst werden. (Für die Lösung linearer Gleichungssysteme gibt es die Funktion linsolve im Package linalg.)

```
> solve({x+y=1, 2*x+y=3}, {x,y});
```

$$\{x = 2, y = -1\}$$

```
> gl_system:= {a*x^2+y^2=1, x-y-1 = 0};
  loesung:=solve( gl_system, {x,y});
```

$$gl\_system := \{a\,x^2 + y^2 = 1, x - y - 1 = 0\}$$

$$loesung = \{y = -1, x = 0\}, \left\{y = -\frac{-1 + a}{1 + a}, x = \frac{2}{1 + a}\right\}$$

Beachte: Maple schließt den Fall $a = -1$ nicht aus!!

Auch bei einem Computeralgebra-System muss das Ergebnis überprüft werden!

```
> subs(a=-1,loesung);
Error, numeric exception: division by zero
```

```
>
```

Numerisches Lösen von Gleichungen mittels fsolve

```
> restart;
  f1 := x1 + exp(x1-1) + (x2+x3^2)^2 - 27 = 0;
```

```
f2 := exp(x2-2) / x1 + x3^4 - 10 = 0;
f3 := x3^2 + sin(x2-2) + x2^2 - 7 = 0;
```

$$f1 := x1 + \mathbf{e}^{(x1 - 1)} + (x2 + x3^2)^2 - 27 = 0$$

$$f2 := \frac{\mathbf{e}^{(x2 - 2)}}{x1} + x3^4 - 10 = 0$$

$$f3 := x3^2 + \sin(x2 - 2) + x2^2 - 7 = 0$$

```
> fsolve ({f1, f2, f3}, {x1, x2, x3});
```

$$\{x1 = 1.000000000,\ x2 = 2.000000000,\ x3 = -1.732050808\}$$

Wir können die Größen f_1 , f_2 , f_3 und x_1 , x_2 , x_3 auch mit Indizes schreiben:

```
> restart;
  f[1] := x[1] + exp(x[1]-1) + (x[2]+x[3]^2)^2 - 27 = 0;
  f[2] := exp(x[2]-2) / x[1] + x[3]^4 - 10 = 0;
  f[3] := x[3]^2 + sin(x[2]-2) + x[2]^2 - 7 = 0;
```

$$f_1 := x_1 + \mathbf{e}^{(x_1 - 1)} + \left(x_2 + x_3^2\right)^2 - 27 = 0$$

$$f_2 := \frac{\mathbf{e}^{(x_2 - 2)}}{x_1} + x_3^4 - 10 = 0$$

$$f_3 := x_3^2 + \sin(x_2 - 2) + x_2^2 - 7 = 0$$

```
> fsolve({f[1], f[2], f[3]},{x[1],x[2],x[3]});
```

$$\{x_1 = 1.000000000,\ x_2 = 2.000000000,\ x_3 = -1.732050808\}$$

```
> restart;
```

Lösen einer gew. Differentialgleichung mit dsolve

1. $\left(\frac{d^2}{dx^2} y(x)\right) - y(x) = 1$

2. $\left(\frac{d}{dt} v(t)\right) + 2\,t = 0$ **mit** $v(1) = 5$

3. $\frac{d^2}{dx^2} y(x) = -y(x)$ **mit** $y(0) = 0$, $y'(0) = 1$

siehe auch: DEtools

bzw. pdsolve und PDEtools für die Lösung partieller Differentialgleichungen

```
> dgl1 := diff(y(x),x,x) -y(x) = 1;
  dgl2 := diff(v(t),t) + 2 * t = 0; anf2 := v(1) = 5;
```

$$dgl1 := \left(\frac{d^2}{dx^2} y(x) \right) - y(x) = 1$$

$$dgl2 := \left(\frac{d}{dt} v(t) \right) + 2\,t = 0$$

$$anf2 := v(1) = 5$$

```
> dsolve(dgl1, y(x));
```

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

```
> dsolve({dgl2, anf2}, v(t));
```

$$v(t) = -t^2 + 6$$

```
> dsolve({diff(y(x),x$2) = - y(x), y(0) = 0, D(y)(0) = 1}, y(x));
```

$$y(x) = \sin(x)$$

```
>
```

Einige weitere häufig verwendete Maple Elemente

```
> restart;
```

Mit dem Kammando subs werden Substitutionen durchgeführt

subs

```
> F := 2 * Pi * r;
  subs( r=5, F);
  F;
```

$$F := 2\,\pi\,r$$

$$10\,\pi$$

$$2\,\pi\,r$$

assign interpretiert Gleichungen als Zuweisungen:

```
> restart;
  x:='x'; y:='y';
```

$$x := x$$

$$y := y$$

```
> solve ({2*x + 3*y = 1,  3 * x - a*y = 0}, {x,y} );
```

$$\left\{ x = \frac{a}{2\,a + 9},\ y = \frac{3}{2\,a + 9} \right\}$$

```
> assign(%);
  x; y;
```

$$\frac{a}{2\,a + 9}$$

$$\frac{3}{2\,a + 9}$$

- **%** bezeichnet das Ergebnis der zuletzt ausgeführten Maple Funktion
- **%%** das Ergebnis des vorletzten und
- **%%%** das Ergebnis des drittletzten Funktionsaufrufs

```
>
```

Maple Libraries und Packages

```
> restart;
```

Ein Designkriterium bei der Entwicklung von Maple war, dass Maple auch auf Rechnern mit kleinem Hauptspeicher ablauffähig ist. Deshalb stehen nicht immer alle der über 2700 Funktionen direkt zur Verfügung.

Die Funktionen sind in verschiedene Libraries und Packages aufgeteilt:

- Die Standard Library

- Die Libraries für verschiedene Anwendungsgebiete (packages)

- Maple Prozeduren und Worksheets vom Maple Application Center
http://www.mapleapps.com/

Standard Library

```
>
```

Die Funktionen der Standard Library stehen jederzeit zur Verfügung.

Eine Übersicht erhält man mit

?infcn

Packages

Maple beinhaltet viele verschiedene Packages. Eine Liste aller Packages erhält man mit

?index,packages

Eine Übersicht über den Inhalt eines Packages erhält man mit

?*package-name* , z.B. ?student, ?numapprox, ?linalg oder ?combinat

Eine Beschreibung der einzelnen Funktionen erhält man mit einem Kommando der Form

?*package-name* [*functionsname*] z.B. ?student[intercept]

Die Funktionintercept zur Berechnung des Schnittpunktes von zwei Kurven in der Ebene ist Teil des Paketes student und kann deshalb noch nicht direkt aufgerufen werden.

```
> intercept(y = 2*x+1, y = x^2+1);
```

$$\mathit{intercept}(y = 2x + 1, y = x^2 + 1)$$

Beim einmaligen Aufruf der Funktion ist zusätzlich noch der Name des Packages anzugeben und zwar in der Form:

package-name [*funktionsname*] (*parameter*);

```
> student[intercept](y = 2*x+1, y = x^2+1);
```

$$\{y = 1, x = 0\}, \{x = 2, y = 5\}$$

Auch nach diesem ersten Aufruf steht diese Funktion noch nicht so zur Verfügung wie die Funktionen der Standard-Bibliothek.

```
> intercept(y = 2*x+1, y = x^2+1);
```

$$\mathit{intercept}(y = 2x + 1, y = x^2 + 1)$$

Durch das with-Kommando werden Funktionen eines Package geladen:

with (*package-name* , *funktionsname*) (Nur die angegebene Funktion wird geladen.)

with (*package-name*) (Alle Funktionen des Pakets werden geladen.)

```
> with(student,intercept);
```

```
> intercept(2*x+1-y=0, y = x^2+1);
```

$$[intercept]$$

$$\{y = 1, x = 0\}, \{x = 2, y = 5\}$$

```
>
```

Auch Computeralgebra Systeme können sich irren!

```
> restart;
```

```
> sin(arcsin(alpha));
```

$$a$$

Gilt diese Vereinfachung für alle Werte?

```
> t:=500;
```

$$t := 500$$

```
> sin(arcsin(t));
```

$$500$$

Maple liefert als Funktionswert der Sinusfunktion den Wert 500.
Ist das richtig?

Beachte: Maple nimmt an, dass t eine komplexe Variable ist.
Dann ist diese Rechnung zwar nicht falsch aber unüblich.

In neueren Maple Versionen gibt es das Paket RealDomain, das alle Operationen in der Menge der reellen Zahlen durchführt.

```
> assume(alpha>0);
  about(alpha);
  sin(arcsin(alpha));
```

```
Originally alpha, renamed alpha~:
  is assumed to be: RealRange(Open(0),infinity)
```

$$a\sim$$

Die obige Rechnung ist falsch, da wir jetzt festgelegt haben, dass a **eine positive reele Zahl ist.**

```
> arcsin(sin(alpha));
```

$$\arcsin(\sin(a\sim))$$

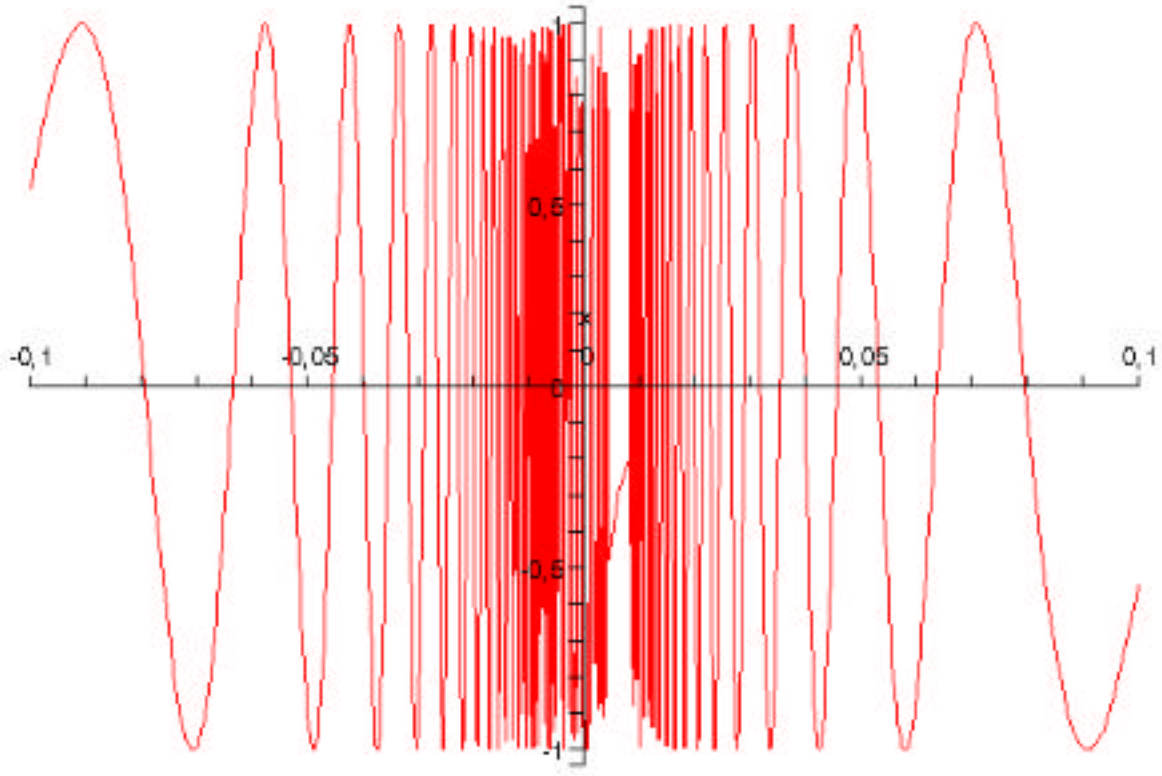
```
> y := sin(t);
  z := arcsin(y);
  evalf(z);
```

$$y := \sin(500)$$

$$z := -500 + 159\,\pi$$

$$-.4867680$$

```
> plot(sin(1/x),x=-0.10..0.10);
```



Diese Grafik enthält offensichtlich einige Fehler!

```
> gl_system:= {a*x^2+y^2=1, x-y-1 = 0};
  loesung:=solve( gl_system, {x,y});
```

$$gl\_system := \{x - \sin(500) - 1 = 0,\ a\,x^2 + \sin(500)^2 = 1\}$$

$$loesung := \{x = 0,\ \sin(500) = -1\},\ \left\{\sin(500) = -\frac{-1+a}{a+1},\ x = \frac{2}{a+1}\right\}$$

Beachte: Maple schließt den Fall $a = -1$ nicht aus!!

```
> subs(a=-1,loesung);
Error, numeric exception: division by zero
```

```
>
```

Auch die Ergebnisse,
die ein Computeralgebra System liefert,
müssen überprüft werden!

Weiter

Datenstrukturen in Maple

N. Geers

Rechenzentrum

Universität Karlsruhe (TH)

e-mail: geers@rz.uni-karlsruhe.de

```
> restart;
```

Die wichtigsten Datenstrukturen in Maple sind

- **Folgen,**
- **Listen,**
- **Mengen,**
- **Tabellen,**
- **Felder,**
- **Matrizen**

Folgen (Sequences)

Eine Folge (sequence oder expression sequence) besteht aus mehreren Ausdrücken, die jeweils durch ein Komma voneinander getrennt sind.

Einzelne Elemente einer Folge werden in der Form *name*[*index*] angesprochen.

Für *index* kann angegeben werden:

- **[*i*] , *i* ist ein Integer-Ausdruck: Auswahl des i-ten Elements**
- **[*i..j*] , Auswahl einer Teilfolge mit den Elemente i, i+1, ... j**
- **[ ] , Auswahl aller Elemente**

```
> s := 1, 4, 9, 16, 25;
```

$$s := 1, 4, 9, 16, 25$$

```
> s[1];
  s[3];
  s[2..4];
```

$$1$$

$$9$$

$$4, 9, 16$$

Mit whattype kann der Type eines Objektes festgestellt werden.

```
> whattype(s);
```

$$exprseq$$

```
> t := sin, cos, tan;
```

$$t := \sin, \cos, \tan$$

Folgen können konkateniert werden:

```
> u := s,t;
```

$$u := 1, 4, 9, 16, 25, \sin, \cos, \tan$$

```
> v := s,s;
```

$$v := 1, 4, 9, 16, 25, 1, 4, 9, 16, 25$$

Manche Maple Prozeduren haben Folgen als Parameter oder liefern Folgen als Ergenbis.
Folgen als Argumente einer Funktion:

```
> max(v);
```

$$25$$

```
> min(s,0,s);
```

$$0$$

Folgen als Ergebnis eines Funktionsaufrufs:

```
> restart;
  coeffs(2*a*x**3 + b*x**2 + d, x, t);t;
```

$$d, b, 2\,a$$

$$1, x^2, x^3$$

Mit der Funktion seq werden Folgen erzeugt:

```
> seq(i^2, i=2..6);
```

$$4, 9, 16, 25, 36$$

```
> a := seq(1/i, i = 1..6);
```

$$a := 1, \frac{1}{2}, \frac{1}{3}, \frac{1}{4}, \frac{1}{5}, \frac{1}{6}$$

Aufgabe: Bestimme die Koeffizienten eines Polynoms:

```
> a:= 3*x^3 + y*x - 11;
```

$$a := 3\,x^3 + y\,x - 11$$

```
> seq( coeff(a,x,i), i = 0.. degree(a,x) );
```

$$-11, y, 0, 3$$

>

Listen

Listen (Lists) und Mengen (Sets) werden aus Folgen konstruiert.

Eine Liste ist eine in [ ] eingeschlossene Folge.

Eine leere Liste wird mit [ ] bezeichnet.

```
> l := [x, 1, 1 -z, 1,x];
```

$$l := [x, 1, 1 - z, 1, x]$$

```
> whattype(l);
```

$$list$$

Der wesentliche Unterschied zwischen Listen und Folgen ist, dass bei einer Liste von Listen die Struktur erhalten bleibt, während eine Folge von Folgen zu einer einzigen Folge zusammengefaßt wird.

Die Elemente einer Liste können deshalb selbst wieder Listen sein:

```
> Punkte := [ [0,0], [1.5, 0.5], [1.0, 2.0], [-0.5, 1.5], [0, 0] ];
```

$$Punkte := [[0, 0], [1.5, 0.5], [1.0, 2.0], [-.5, 1.5], [0, 0]]$$

Einzelne oder mehrere Elemente einer Liste können in der Form *name*[*index*] ausgewählt werden.

Als Indexausdruck kann angegeben werden:

- **[*i*] , *i* ist ein Integer-Ausdruck: Auswahl des i-ten Elementes**
- **[*i..j*] , der Elemente i, i+1, ... j**
- **[ ] , Auswahl aller Elemente**

Das Ergebnis einer solchen Auswahl ist eine Folge.

Um zwei Listen zu einer zusammenzufassen, kann deshalb folgendermaßen vorgegangen werden:

```
> hochpunkte := [ [-1, 0], [2, 3], [5, 4] ];
  tiefpunkte := [ [-6, -2], [0, -1], [3, 2] ];
  extrempunkte := [ hochpunkte[], tiefpunkte[] ];
```

$$hochpunkte := [[-1, 0], [2, 3], [5, 4]]$$

$$tiefpunkte := [[-6, -2], [0, -1], [3, 2]]$$

$$extrempunkte := [[-1, 0], [2, 3], [5, 4], [-6, -2], [0, -1], [3, 2]]$$

>

Beispiele zu Listen:

Erstelle eine Liste, die die Elemente x, x^2, x^3, ... bis x^10 enthält.

```
> l := [seq(x^i, i = 1..10)];
```

$$l := [x, x^2, x^3, x^4, x^5, x^6, x^7, x^8, x^9, x^{10}]$$

Erstelle eine Liste der x- bzw. y-Koordinaten zu einer gegebenen Liste von Punkten.
(Die Funktion nops liefert die Anzahl der Elemente einer Liste.)

```
> l:= [ [0,0],[1,0.8],[2,1.9],[3,2.6] ];
```

$$l := [[0, 0], [1, 0.8], [2, 1.9], [3, 2.6]]$$

```
> xl:= [ seq(l[i][1],i=1..nops(l)) ];
```

$$xl := [0, 1, 2, 3]$$

```
> yl := [ seq(l[i][2],i=1..nops(l)) ];
```

$$yl := [0, 0.8, 1.9, 2.6]$$

Erstelle eine Wertetabelle einer Funktion $f : x \mapsto f(x) = \frac{x^4 - 2x^3 + 3x}{2x^2 + 1}$ **im Intervall [-5, 5]**

```
> f := x -> (x^4-2*x^3+3*x)/(2*x^2+1);
  x1 := -5; x2:= 5; n:= 20; dx:= (x2-x1)/n;
  wertetabelle:= [  seq( [ x1+i*dx, f(x1+i*dx) ], i=0..n) ];
```

$$f := x \mapsto \frac{x^4 - 2x^3 + 3x}{2x^2 + 1}$$

$$x1 := -5$$

$$x2 := 5$$

$$n := 20$$

$$dx := \frac{1}{2}$$

$$wertetabelle = \left[\left[-5, \frac{860}{51} \right], \left[\frac{-9}{2}, \frac{9261}{664} \right], \left[-4, \frac{124}{11} \right], \left[\frac{-7}{2}, \frac{3605}{408} \right], \left[-3, \frac{126}{19} \right], \left[\frac{-5}{2}, \frac{335}{72} \right], \left[-2, \frac{26}{9} \right], \right.$$

$$\left[\frac{-3}{2}, \frac{117}{88} \right], [-1, 0], \left[\frac{-1}{2}, \frac{-19}{24} \right], [0, 0], \left[\frac{1}{2}, \frac{7}{8} \right], \left[1, \frac{2}{3} \right], \left[\frac{3}{2}, \frac{45}{88} \right], \left[2, \frac{2}{3} \right], \left[\frac{5}{2}, \frac{245}{216} \right], \left[3, \frac{36}{19} \right],$$

$$\left. \left[\frac{7}{2}, \frac{399}{136} \right], \left[4, \frac{140}{33} \right], \left[\frac{9}{2}, \frac{3861}{664} \right], \right.$$

$$\left[5, \frac{130}{17}\right]$$

Operationen für Listen sind:

op, nops, member , convert, subsop

>

Mengen (Sets)

Mengen (Sets) sind dadurch charakterisiert, dass die Elemente nicht in einer festen Reihenfolge vorliegen und dass jedes Element nur einmal in der Menge vorhanden ist.

> s:= {x, 1, 1 - z, 1 , x};

$$s := \{1, x, 1 - z\}$$

> whattype(s);

$$set$$

nops liefert die Anzahl der Elemente einer Liste bzw. Menge.

> nops(l);

$$4$$

> nops(s);

$$3$$

> s[2..3];

$$\{x, 1 - z\}$$

Mengen können Argument oder Funktionswerte von Maple Funktionen sein:

> solve({x^2+y^2=1, y=2*x-1});

$$\{x = 0, y = -1\}, \left\{x = \frac{4}{5}, y = \frac{3}{5}\right\}$$

Operationen für Mengen sind:

union, intersect , minus, member , convert, op, nops

>

Tabellen

Tabellen werden implizit erzeugt, wenn ein Wert an eine indizierte Variable zugewiesen wird oder durch den Aufruf der Funktion table.

> farben:= table([rot=[1,0,0], grün=[0,1,0], blau=[0,0,1], weiss=[1,1,1]]);

$$farben := TABLE([blau = [0, 0, 1], weiss = [1, 1, 1], rot = [1, 0, 0], grün = [0, 1, 0]])$$

> farben[rot];

[1, 0, 0]

```
> farben[schwarz] := [0,0,0];
```

$$farben_{schwarz} := [0, 0, 0]$$

```
> farben[weiss]; farben[schwarz];
```

[1, 1, 1]

[0, 0, 0]

```
>
```

Felder (Arrays)

Felder werden mit dem Kommando array erzeugt. Sie sind eine Spezialisierung des Datentyps table. Die Indizes eines Arrays sind immer ganzzahlig.

1- bzw. 2-dimensionale Arrays mit der unteren Indexgrenze 1 werden von Maple i.a. wie Vektoren bzw. Matrizen behandelt (s.a. matrix).

```
> restart;
> v:= array(1..5);
  A := array(-2..4,0..1);
```

$$v := array(1 .. 5, [])$$

$$A := array(-2 .. 4, 0 .. 1, [])$$

```
> A[0,0] := sin(x);
```

$$A_{0,\,0} := \sin(x)$$

```
> w := array(1..8, [seq(i^3,i=1..8)]);
```

$$w := [1, 8, 27, 64, 125, 216, 343, 512]$$

```
> x:= array(-1..3,[a,b,c,d,e]);
```

$$x := ARRAY([-1 .. 3], [(-1) = a, (0) = b, (1) = c, (2) = d, (3) = e])$$

```
> z := array(1..2, 1..2, [ [11, 12], [21, 22] ]);
```

$$z := \begin{bmatrix} 11 & 12 \\ 21 & 22 \end{bmatrix}$$

Auswertung und Ausgabe von Feldern und Feldelementen

Aus Effizienzgründen werden Ausdrücke, die Arrays enthalten i.a. nicht elementweise berechnet.

Bei Matrizen (1- bzw. 2-dim Arrays mit unterer Indexgrenze 1) erfolgt die Berechnung der einzelnen Elemente bei Aufruf der Funktion evalm.

Bei anderen Arrays ist ein Aufruf der Form
map (eval, *array*);
erforderlich.

```
> w;
```

```
x;
z;
```

$$w$$

$$x$$

$$z$$

```
> evalm(w);
  evalm(x);
  evalm(z);
```

$$[1, 8, 27, 64, 125, 216, 343, 512]$$

$$x$$

$$\begin{bmatrix} 11 & 12 \\ 21 & 22 \end{bmatrix}$$

```
> map(eval,w);
  map(eval,x);
  map(eval,z);
```

$$[1, 8, 27, 64, 125, 216, 343, 512]$$

$$ARRAY([-1 .. 3], [(-1) = a, (0) = b, (1) = c, (2) = d, (3) = e])$$

$$\begin{bmatrix} 11 & 12 \\ 21 & 22 \end{bmatrix}$$

Eindimensionale Felder (Vektoren) werden i.a. als Zeilenvektor ausgegeben. Um eine bessere Lesbarkeit zu erreichen, ist die Konvertierung in eine Matrix mittels convert sinnvoll. Ein Vektor wird dann spaltenweise ausgegeben:

```
> evalm(convert(w,matrix));
```

$$\begin{bmatrix} 1 \\ 8 \\ 27 \\ 64 \\ 125 \\ 216 \\ 343 \\ 512 \end{bmatrix}$$

Bei der Definition von Feldern (arrays) können verschiedene

Attribute festgelegt werden (s. ?indexfcn):

symmetric, antisymmetric, sparse, diagonal, identity

Beispiele zu Operationen mit ganzen Arrays, z.B. Berechnung von Matrix-Vektor-Ausdrücken, Lösen linearer Gleichungssysteme etc. finden Sie im Arbeitsblatt linalg.mws

>

Kontrollstrukturen in Maple

N. Geers

Rechenzentrum

Universität Karlsruhe (TH)

e-mail: geers@rz.uni-karlsruhe.de

```
> restart;
```

Vergleichsausdrücke, logische Ausdrücke

```
>
```

Als Operatoren in Vergleichsausdrücken werden verwendet:

= Abfrage auf Gleichheit

< Abfrage auf 'kleiner als'

> Abfrage auf 'grösser als'

<= Abfrage auf 'kleiner oder gleich'

>= Abfrage auf 'grösser oder gleich'

<> Abfrage auf Ungleichheit

Logische Ausdrücke werden verknüpft mit den logischen Operationen and, or und not und können mit der Funktion evalb (b für boolean) ausgewertet werden.

Bedingte Anweisungen

```
>
```

Die Syntax der if Anweisung lautet:

if *bedingung* **then**
anweisungsfolge
[elif *bedingung* **then**
anweisungsfolge **]**
. . .

[else
anweisungsfolge **]**
fi

• Der elif - und der else -Block sind optional.

- **Der elif -Block kann mehrfach in einer if -Anweisung auftreten.**

Beispiele zur if -Anweisung:

```
> x:= -1/2; y:= 0;
```

$$x := \frac{-1}{2}$$

$$y := 0$$

```
> if x < 1 then 0 fi;
```

$$0$$

```
> if y > 1 then
    z := 0
  elif y = 1 then
    z := 1
  else
    z := -1
  fi;
```

$$z := -1$$

```
> if x<0 then
    lprint(`x = `, x );
    x:= abs(x)
  fi;
```

```
x = , (-1)/2
```

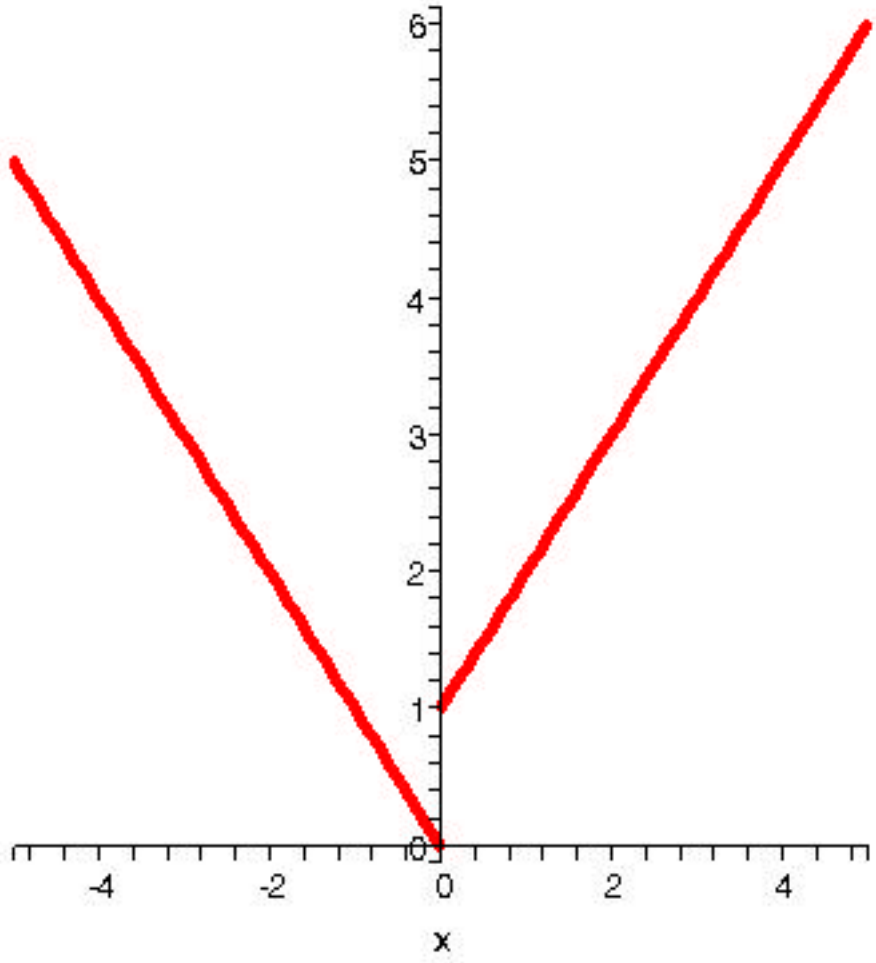
$$x := \frac{1}{2}$$

Abschnittsweise Definition einer Funktion:

```
> x:='x':
  f:= x -> if x < 0 then
        -x
      elif x = 1/2 then
        0
      else
        1+x
      fi;
```

$$f := x \mapsto \textbf{if } x < 0 \textbf{ then } -x \textbf{ elif } x = \frac{1}{2} \textbf{ then } 0 \textbf{ else } 1 + x \textbf{ end if}$$

```
> plot('f(x)', x=-5..5, discont=true, color=red, thickness=3);
```



Man beachte in obigem Beispiel, dass das erste Argument der Funktion plot in der Form 'f(x)' und nicht nur als f(x) übergeben wird. Dies ist immer dann notwendig, wenn der Funktionsausdruck nicht als ein Funktionsterm angegeben werden kann.

Die Option discont=true bewirkt, dass Maple die Funktion zuerst auf Unstetigkeitsstellen untersucht.

>

Wiederholungsanweisungen

>

Die Syntax der for/while Anweisung lautet:

for ***name*** **[from** ***ausdruck*** **] [by** ***ausdruck*** **] [to** ***ausdruck*** **]**
[while ***ausdruck*** **]**
do
anweisungsfolge
od

oder

while *ausdruck*
do
anweisungsfolge
od

oder

for *name* **[in** *ausdruck* **] [while** *ausdruck* **]**
do
anweisungsfolge
od

Die in [ ] eingeschlossenen Teile der Anweisung sind jeweils optional.

Beispiele zur Wiederholungsanweisung:

```
> for i from 0 by 2 to 6
  do
    print (`i = ` || i)
  od;
```

i = 0
i = 2
i = 4
i = 6

```
> for i from 0 by 2 while i <=6
  do
    print (`i = ` || i)
  od;
```

i = 0
i = 2
i = 4
i = 6

Wenn keine Schrittweite angegeben wird, ist die Standardschrittweite 1.

```
> for i from 0 to 4
  do
    print (`i = ` || i)
  od;
```

i = 0
i = 1
i = 2
i = 3
i = 4

Wenn kein Anfangswert festgelegt wird, ist der Anfangswert 1.

```
> for i to 4
  do
    print (`i = ` || i)
  od;
```

i = 1

i = 2

i = 3

i = 4

Die while-Schleife:

```
> i := 6:
  while i > 2
  do
    print (`i = ` || i);
    i := i - 1
  od:
```

i = 6

i = 5

i = 4

i = 3

Für Mengen oder Listen kann die Form for *name* in *name* verwendet werden.

```
> Menge := { a, 2, b, c, 5, cos(x), x^2 }:
  S := {}:
  for e in Menge
  do
    if type(e, integer) then
      S := S union {e};
    fi;
  od;
  S;
```

$\{2, 5\}$

Die nachfolgende Schleife wird für alle Elemente der Liste ausgeführt, bis das erste Element gefunden wird, für das die Bedingung, dass es vom Typ integer ist, nicht erfüllt ist.

```
> Liste := [ 3, 6, 20, x, u, w, l, 17 ]:
  for e in Liste while type(e,integer)
  do
    print (e);
  od;
```

3

6

20

Bestimme die Summe alle ungeraden, zweistelligen Zahlen:

```
> summe := 0:
  for i from 11 by 2 while i < 100
  do
    summe := summe + i;
  od:
  summe;
```

2475

Falls die for -Anweisung hinter od mit einem ; abgeschlossen würde, wird das Ergebnis jeder Anweisung im Schleifenrumpf angezeigt.

Bestimme die Summe aller Elemente einer Liste:

```
>  liste :=[ 3, 5, 2, 7, 8, 9, 25 ];
```

$$liste := [3, 5, 2, 7, 8, 9, 25]$$

```
>  lsum := 0:
   for z in liste do
     lsum := lsum + z
   od :
   lprint (`Die Summe der Listenelemente ist : `, lsum);
```

```
Die Summe der Listenelemente ist : , 59
```

Suche die nächste Primzahl, die grösser als n ist:

```
>  n := 1000000;
```

$$n := 1000000$$

```
>  if type(n,even) then
     N:= n+1
   else
     N = n
   fi:
   while not isprime(N)
   do
     N:= N+2
   od:
   lprint (`Die kleinste Primzahl groesser als ` || n ||` ist ` || N);
```

```
Die kleinste Primzahl groesser als 1000000 ist 1000003
```

Beenden einer Wiederholung mittels break

Aufgabe: Suche die erste Zahl der Form $2^i - 1, i = 3, 5, 7$**, ... die keine Primzahl ist.**

```
> for i from 3 by 2 do
    if isprime (2^i - 1) then
      print (2^i - 1, `ist eine Primzahl`)
    else
      print(2^i - 1, `ist keine Primzahl`);
      break
    fi
  od;
```

7, *ist eine Primzahl*

31, *ist eine Primzahl*

127, *ist eine Primzahl*

511, *ist keine Primzahl*

```
>
```

Schleifen und seq Funktion

Folgen oder Listen, die nach einem bestimmten Muster aufgebaut sind, können sowohl mit Hilfe einer for -Schleife als auch mit der Maple Funktion seq erstellt werden. In der Regel ist die Funktion seq vorzuziehen, da sie insbesondere für Folgen mit vielen Elementen eine wesentlich kürzere Rechenzeit benötigt.

Beispiel: Es soll eine Folge der ersten 3000 Quadratzahlen erstellt werden.

```
> restart;
  N := 3000;
  t1 := time():
  s1 := seq(i^2, i=1..N):
  zeit = (time()-t1) * Sekunden;
```

$N := 3000$

$zeit = 0.030$ *Sekunden*

Eine zweite Möglichkeit, diese Folge zu erzeugen ist:

```
> t1 := time():
  s2 := NULL:
  for i from 1 to N do
    s2 := s2,i^2
  od:
  zeit = (time()-t1) * Sekunden;
```

$zeit = 0.862$ *Sekunden*

```
>
```

Graphik mit Maple

N. Geers

Rechenzentrum

Universität Karlsruhe (TH)

e-mail: geers@rz.uni-karlsruhe.de

```
> restart;
```

Die nachfolgenden Anweisungen sind für Grossbildprojektionen notwendig, um die Strichstärke bei 2-D Grafiken zu setzen.

```
> plotsetup(default);
> plots[setoptions](thickness=3);
```

Zur grafischen Darstellung von Funktionen etc. gibt es in Maple die Kommandos plot und plot 3d.

Weitere Grafikfunktionen sind im plots Package enthalten bzw. in den Funktionen DEtools[DEplot] und stats[statplot] .

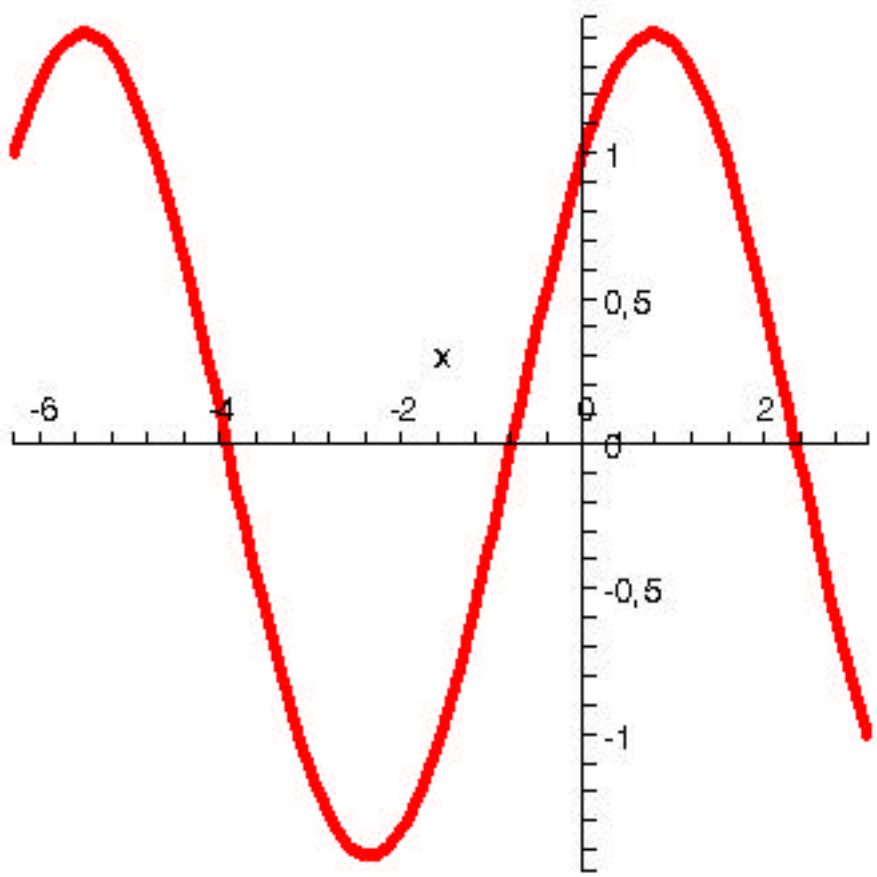
Nachdem eine Grafik erstellt wurde, kann sie i.a. über ein Funktionsmenü (Grafik anklicken und dann rechte Maustaste betätigen) modifiziert werden.

```
>
```

Einige einfache Beispiele

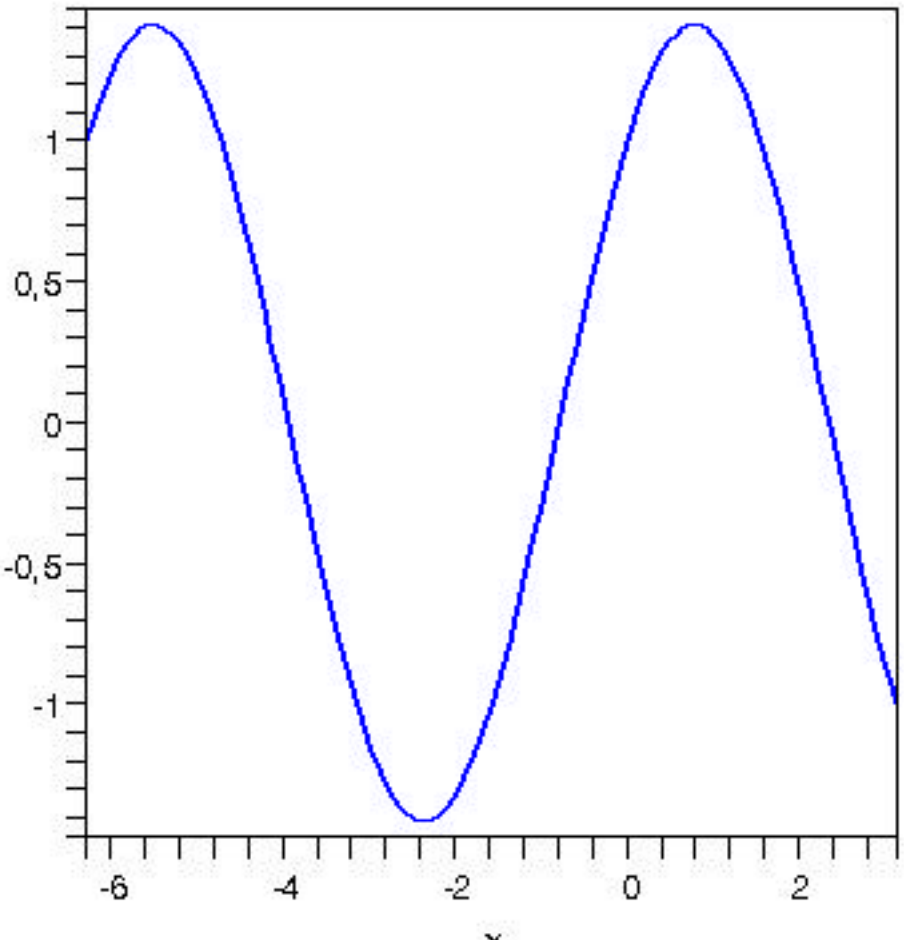
Ein einfaches Beispiel zur grafischen Darstellung einer Funktion ist:

```
> plot(cos(x) + sin(x), x=-2*Pi..Pi);
```



Nach dem Anklicken mit der linken Maustaste kann die Grafik über die Kontextleiste (am oberen Bildschirmrand) oder über ein Funktionsmenü (rechte Maustaste) modifiziert werden. Ausserdem hat die Prozedur plot zahlreiche Optionen, über die Eigenschaften der Grafik definiert werden können.

```
> plot(cos(x) + sin(x), x=-2*Pi..Pi, style=line, linestyle=1,
  color=blue,     thickness=2, axes=boxed);
```



Zur grafischen Darstellung mehrerer Funktionen in einem Schaubild wird

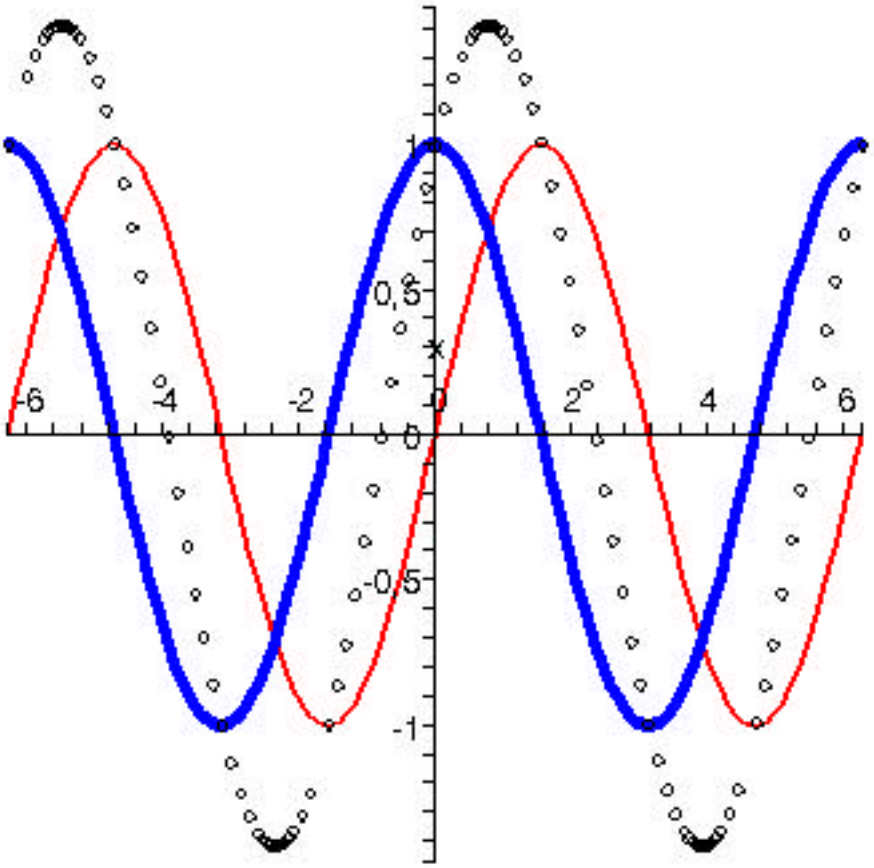
▪ eine Menge von Funktionen (in { } eingeschlossen) oder

▪ eine Liste von Funktionen (in [ ] eingeschlossen)

als erster Parameter an die Prozedur plot übergeben.

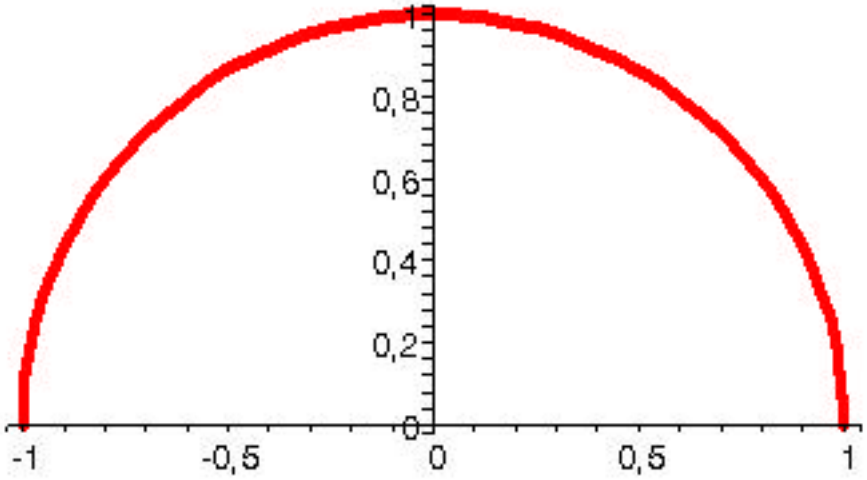
Wenn Attribute (z.B. style, color etc.) für die einzelnen Funktionsgraphen festgelegt werden sollen, ist die Angabe als Liste von Funktionen empfehlenswert, da bei einer Menge die Reihenfolge nicht definiert ist.

```
> plot([ sin(x),cos(x), sin(x)+cos(x)],x=-2*Pi..2*Pi, style=[ line,
  line, point], linestyle=[1,3], color=[red, blue, black],
  thickness=[2,3]);
```



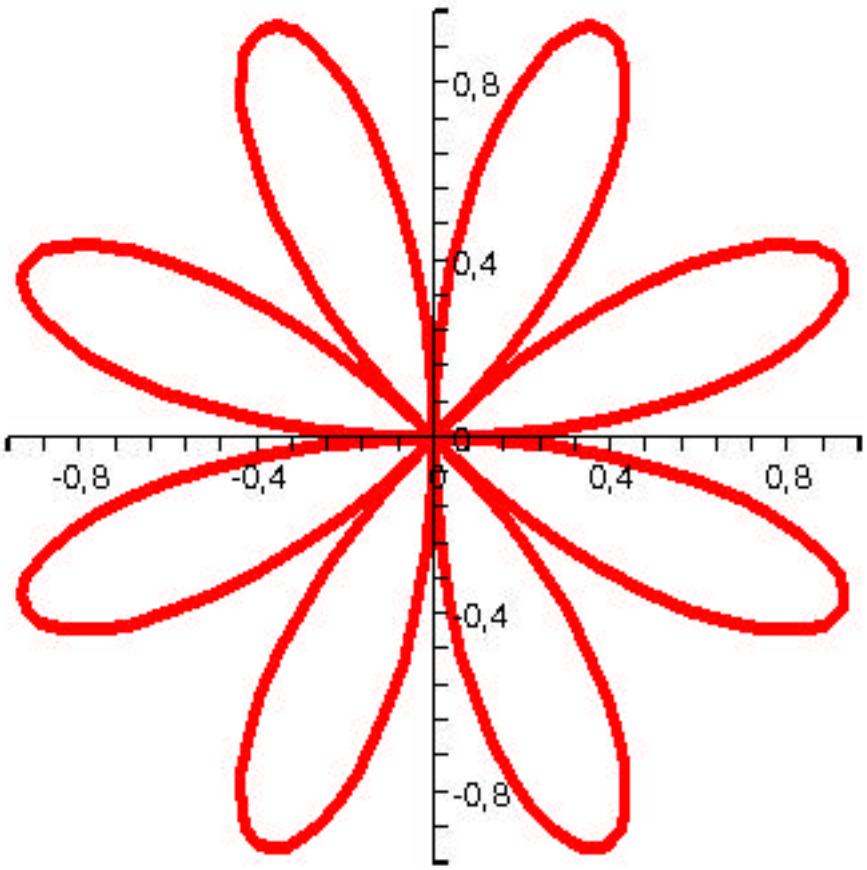
Grafische Darstellung einer Funktion, die in Parameterform gegeben ist:

```
> plot([sin(t),cos(t),t=-Pi/2..Pi/2], scaling=constrained);
```



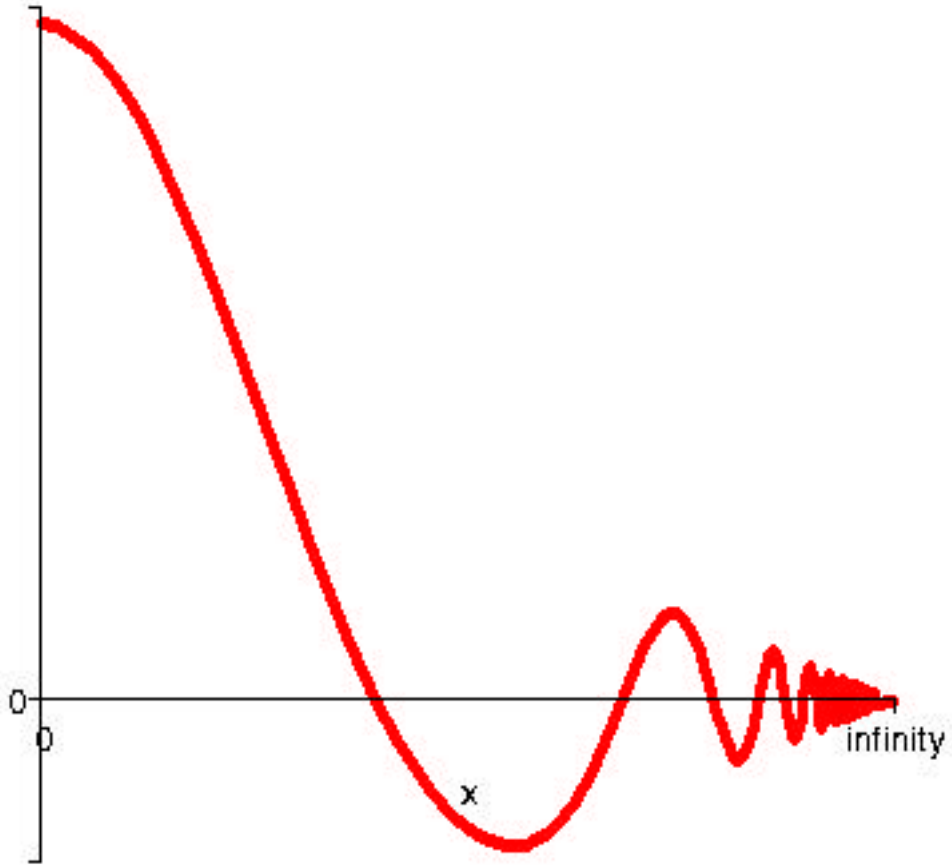
Darstellung mittels Polarkoordinaten:

```
> plot([sin(4*x),x,x=0..2*Pi],coords=polar);
```



Wenn als Grenze des Bereichs, über dem die Funktion dargestellt werden soll, infinity angegeben wird, erhält man eine Darstellung des Funktionsverlaufs für *x* **® ¥.**

> plot(1/x*sin(x),x=0..infinity);



Auch Punktmengen können grafisch dargestellt werden. Im nachfolgenden Beispiel sind folgende Punkte gegeben:

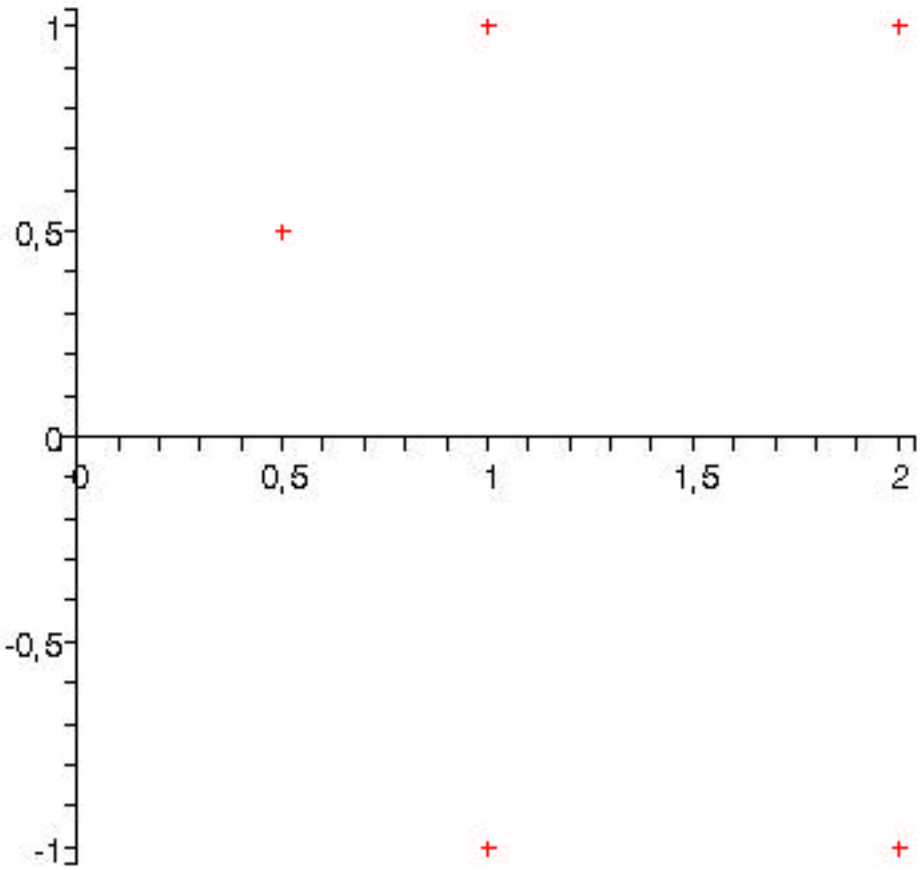
(0.5 , 0.5), (1 , 1), (2 , 1), (2 , -1), (1 , -1), (0 , 0)

Die Koordinaten der Punkte werden als Liste an das plot-Kommando übergeben.

```
> l := [[0.5,0.5],[1,1],[2,1],[2,-1],[1,-1],[0,0]];
```

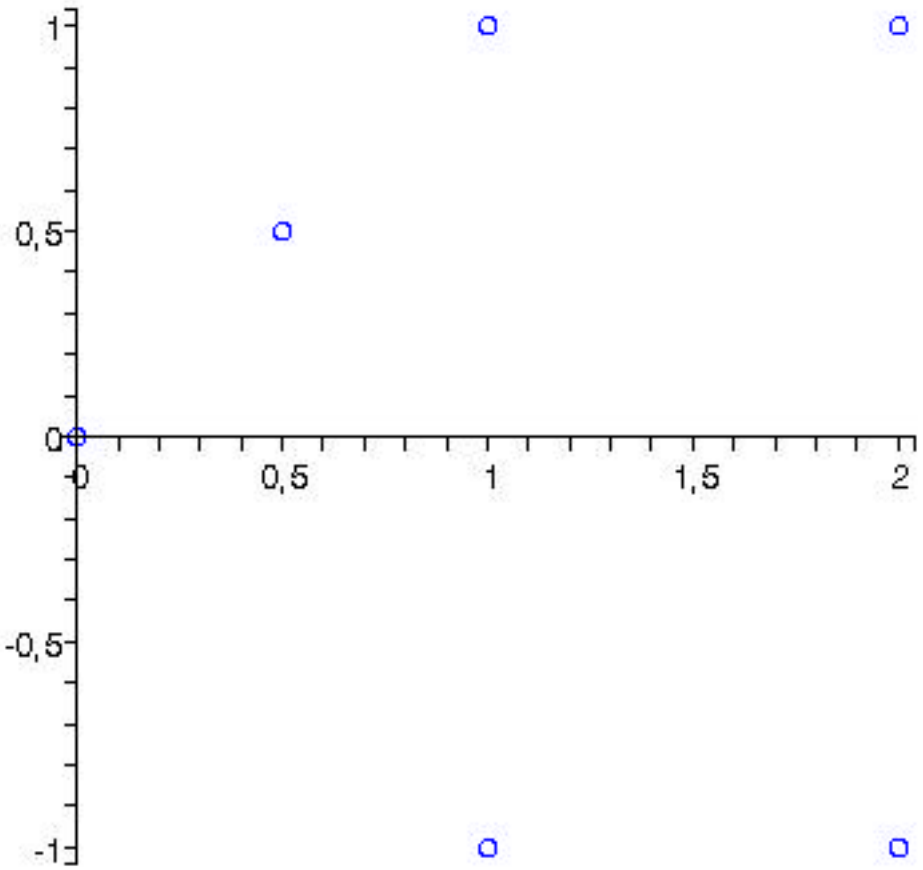
$$l := [[0.5, 0.5], [1, 1], [2, 1], [2, -1], [1, -1], [0, 0]]$$

```
> plot(l, style=point, symbol=cross, symbolsize=10);
```



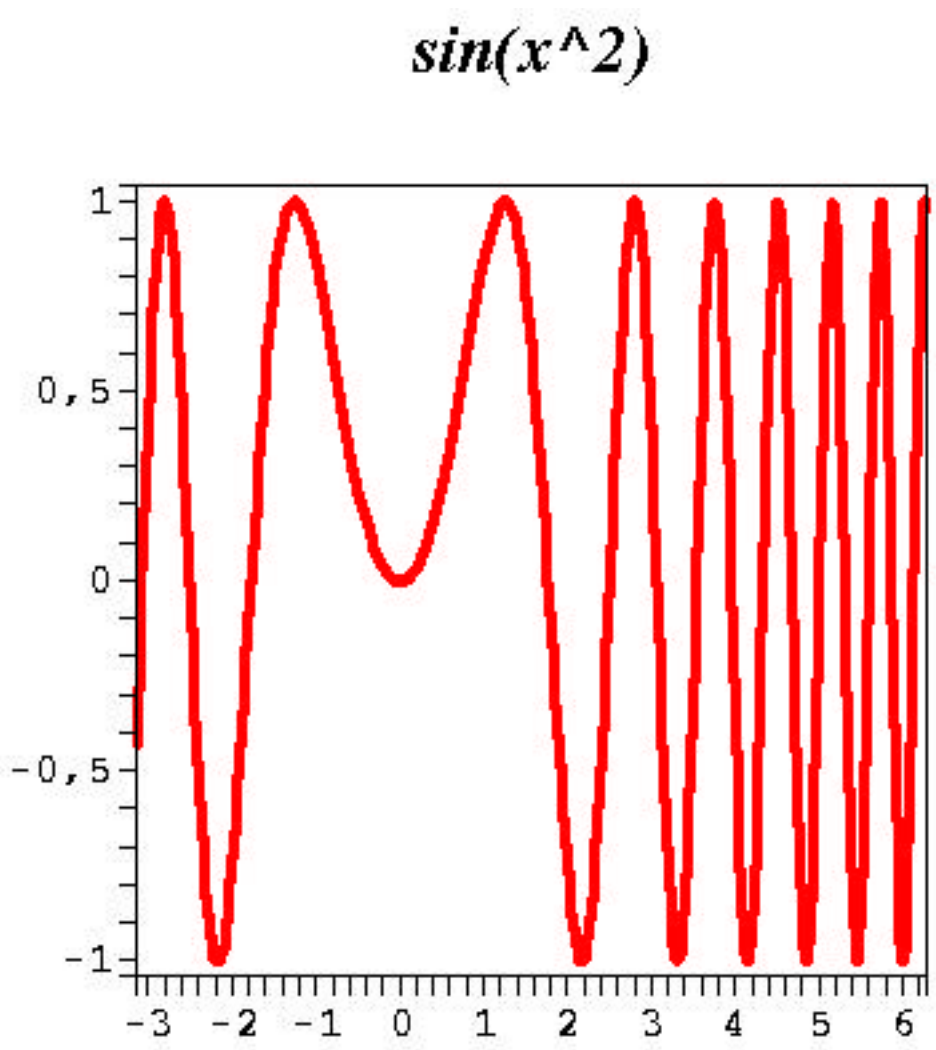
Wenn sehr viele Punkte dargestellt werden sollen, ist die Funktion pointplot aus dem plots Package i.a. deutlich schneller als die Funktion plot .

```
> plots[pointplot](l, color=blue, symbol=circle);
```



Verschiedene Optionen können beim Aufruf des plot-Kommandos gesetzt werden.
Eine Liste der Optionen erhält man mit ?plot[options]

```
> plot(sin(x^2),x=-Pi..2*Pi,
  title=`sin(x^2)`, titlefont=[TIMES,BOLDITALIC,20],
  labels=[`x Achse`, `y Achse`], labelfont=[TIMES,BOLD,15],
  axes=boxed,xtickmarks=10,ytickmarks=5, font=[COURIER,BOLD,12]);
```



>

Mehrere grafische Darstellungen in einem Bild

Wenn die Ausgabe des plot-Kommandos an eine Variable zugewiesen wird, kann anschliessend, z.B. im display Kommando auf die Grafik zugegriffen werden.

Im folgenden Beispiel sind 4 Datenpunkte vorgegeben. Zu diesen Punkten wird eine Ausgleichsgerade berechnet. Danach können die Datenpunkte und die Ausgleichsgerade in einem Bild angezeigt werden.

Dazu werden zwei Graphiken mit unterschiedlichen Attributen erstellt. Die erste Graphik hat das Attribut style=point und enthält nur die Datenpunkte.

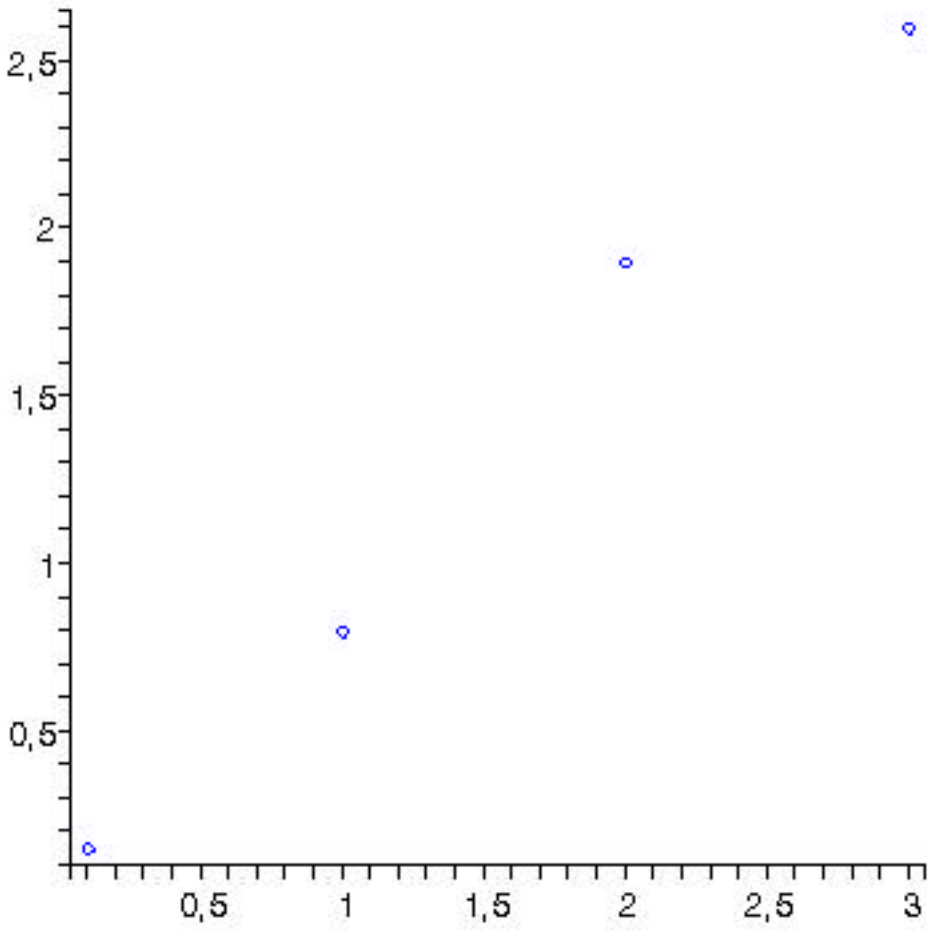
Nachdem mit der Funktion leastsquare aus dem Package stats[fit] die Ausgleichsgerade berechnet wurde, kann diese Gerade mit dem Attribut style=line dargestellt werden.

Beide Graphiken werden mit dem Kommando disply aus dem plots-Package in einem Bild dargestellt.

```
> l:= [[0.1,0.15],[1,0.8],[2,1.9],[3,2.6]];
```

$$l := [[0.1, 0.15], [1, 0.8], [2, 1.9], [3, 2.6]]$$

```
> plot(l,style=point,color=blue,thickness=3,symbol=diamond);
```




```
> plt1:=plot(l,style=point,color=blue,thickness=3,symbol=diamond):
```

Wenn das Ergebnis eines plot Aufrufs an eine Variable zugeweisen werden soll, sollte die Anweisung immer mit ':' und nicht mit ';' abgeschlossen werden, da sonst die ganze plot-Datenstruktur in das Arbeitsblatt geschrieben wird.

Jetzt wird die Ausgleichsgerade berechnet.

```
> with(stats):with(fit):
```

```
> xl:= [seq(l[i][1],i=1..nops(l))];
  yl := [seq(l[i][2],i=1..nops(l))];
```

$$xl := [0.1, 1, 2, 3]$$

$$yl := [0.15, 0.8, 1.9, 2.6]$$

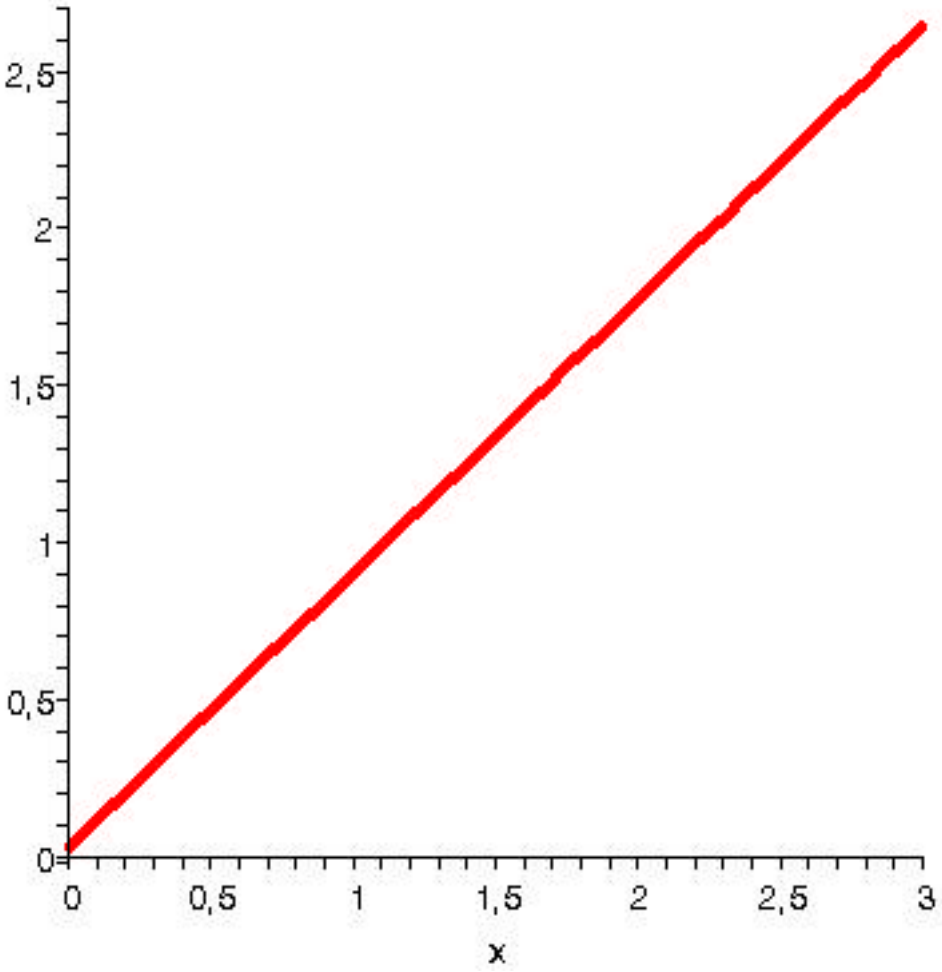
```
> x:='x'; y:= 'y';
  leastsquare[[x,y],y=a*x+b,{a,b}]([xl,yl]);
```

$$x := x$$

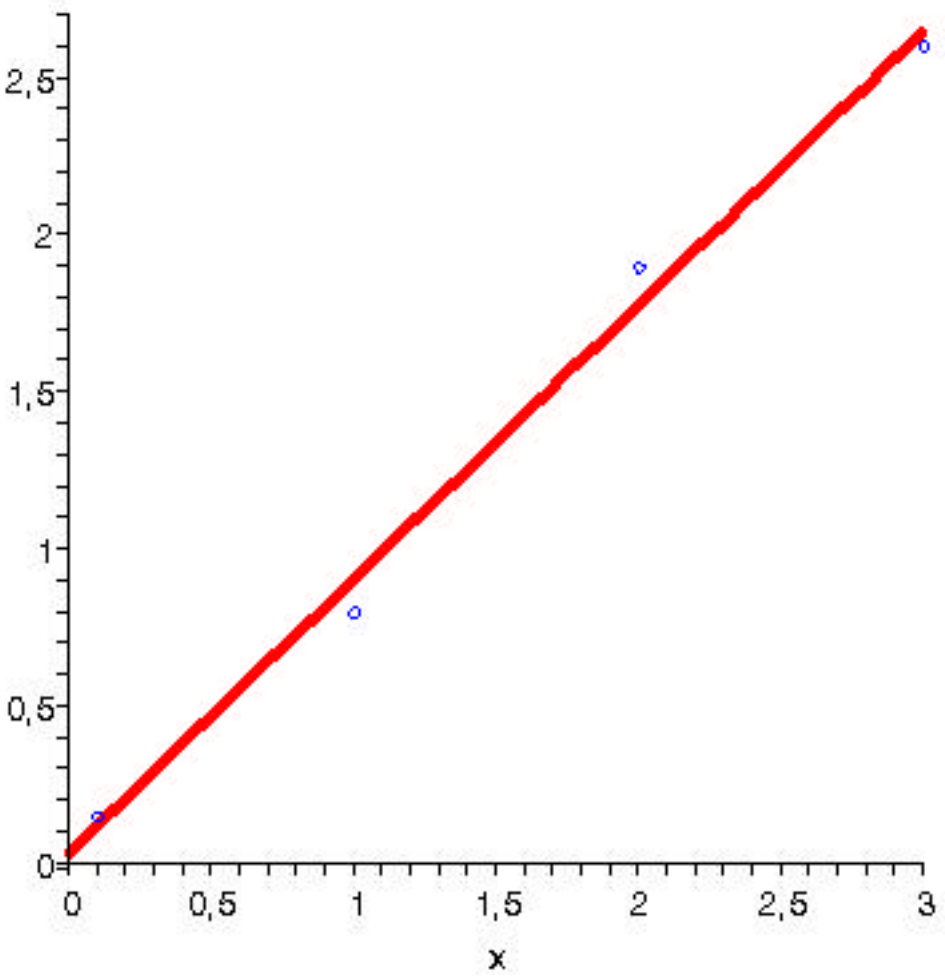
$$y := y$$

$$y = 0.8717472119\,x + 0.03308550186$$

```
> assign(%);
> plot(y,x=0..3);
```




```
> plt2:=plot(y,x=0..3):
> plots[display]({plt1,plt2});
```



>

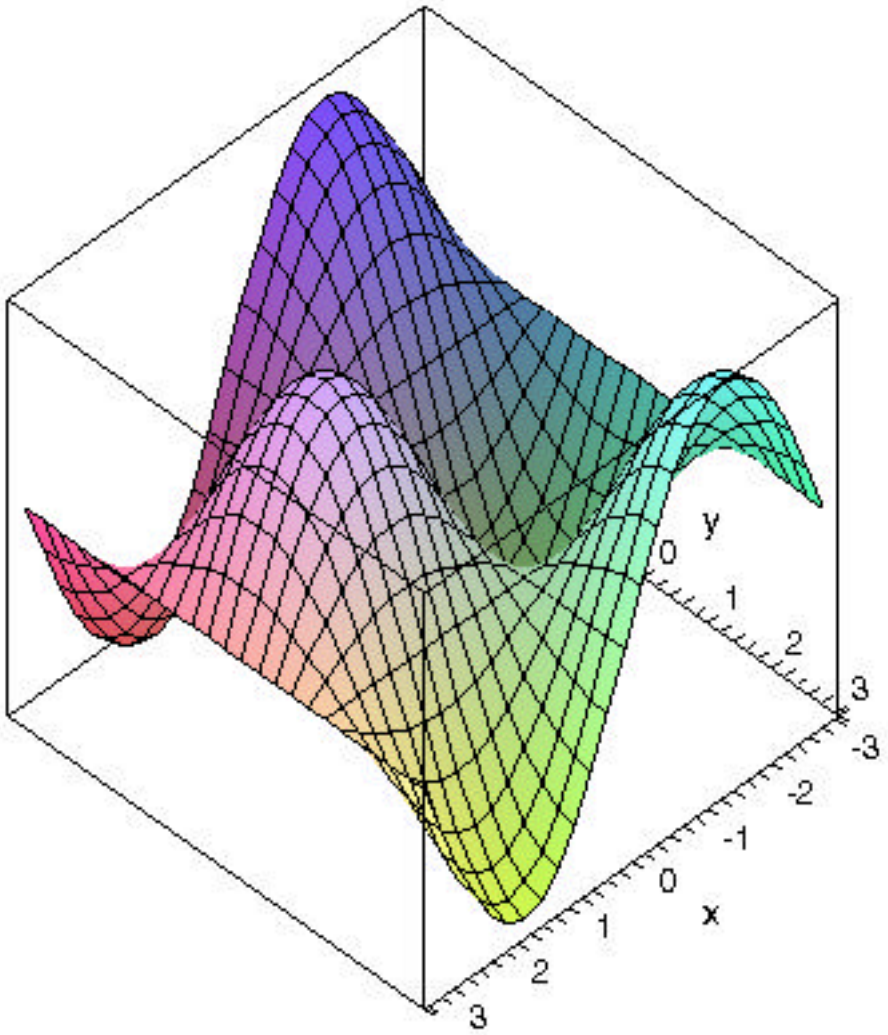
3-D Darstellungen

3-D Grafiken werden mit dem Kommando plot3d erstellt.

Optionen, die bei plot verwendet werden können, haben i.a. analoge Bedeutungen für plot3d.

```
> restart;
```

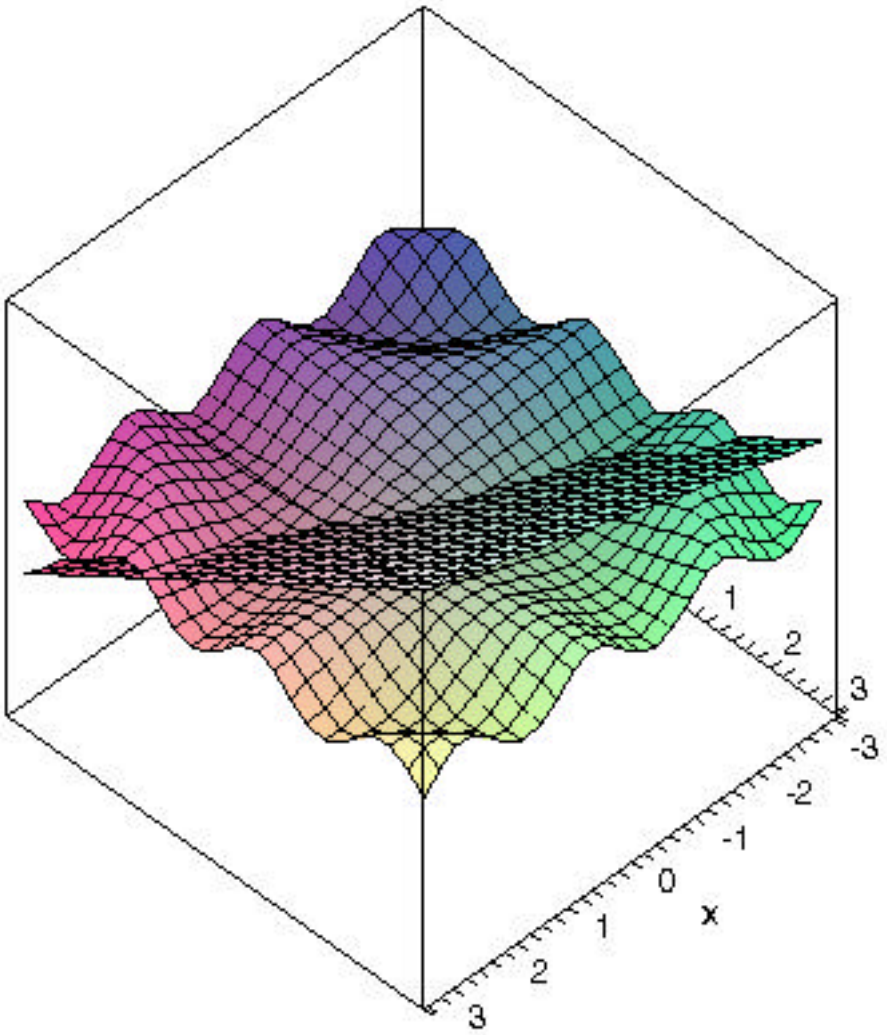
```
> plot3d(sin(x)*cos(y),x=-Pi..Pi,y=-Pi..Pi,axes=boxed);
```



Die Grafiken kann jetzt wieder interaktiv bearbeitet werden.

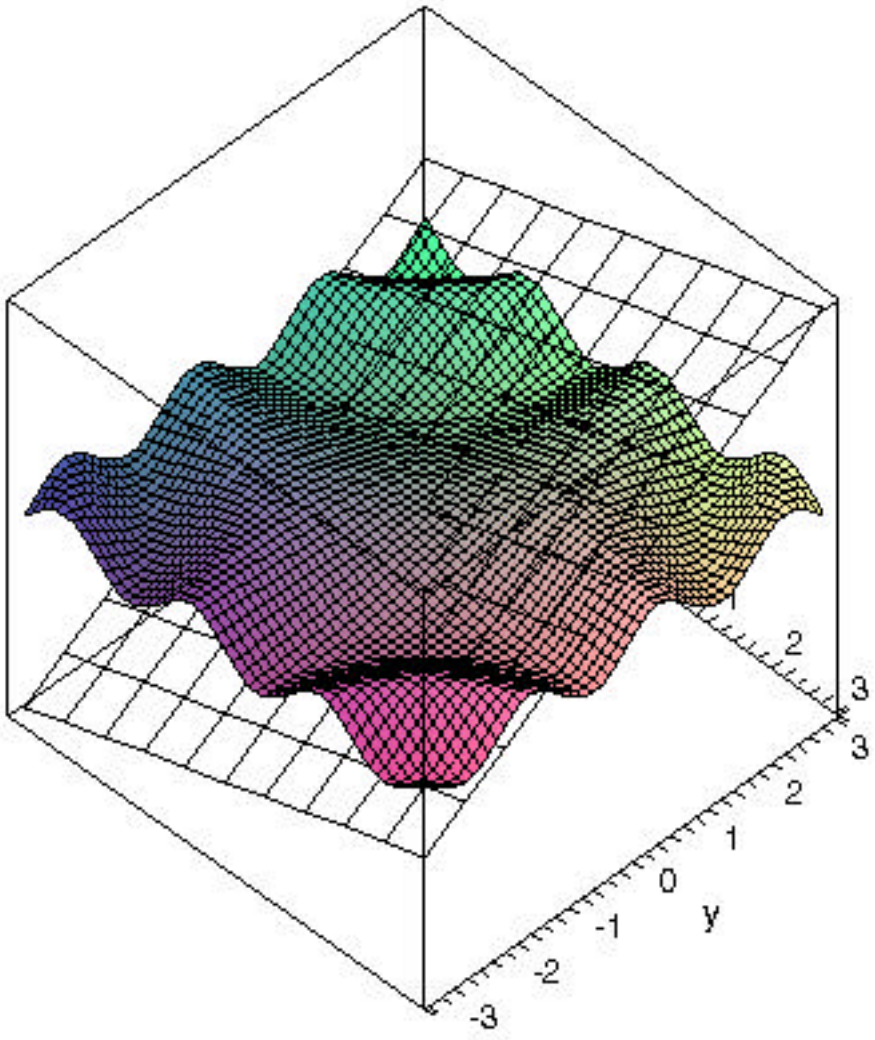
Mehrere Funktionen werden folgendermaßen mit einem plot3d-Aufruf dargestellt:

```
> plot3d({sin(x*y), x + 2*y},x=-Pi..Pi,y=-Pi..Pi,style=patch,
  axes=boxed);
```



Falls die Funktionen mit unterschiedlichen Attributen erstellt werden sollen, benutzt man dazu die display Funktion des plots Package (analog zum obigen 2-D Beispiel).

```
> plt1:=plot3d(sin(x*y),x=-Pi..Pi,y=-Pi..Pi,style=patch,axes=boxed,numpo
  ints=2500):
  plt2:=plot3d( x + 2*y,x=-Pi..Pi,y=-Pi..Pi,style=wireframe,
  color=black,numpoints=100):
  plots[display]({plt1,plt2},orientation=[-45,45]);
```

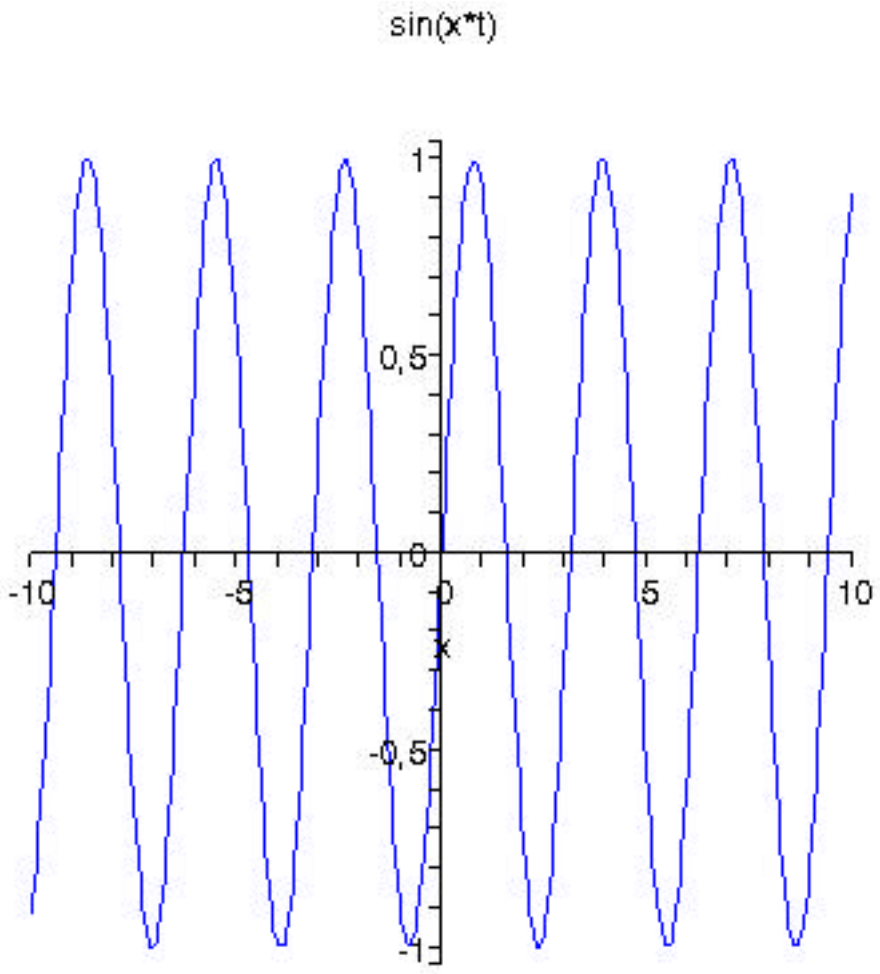


>

Animationen

Für die Erstellung animierter Grafiken hat Maple die Funktionen display, animate und animate3d aus dem plots Package.

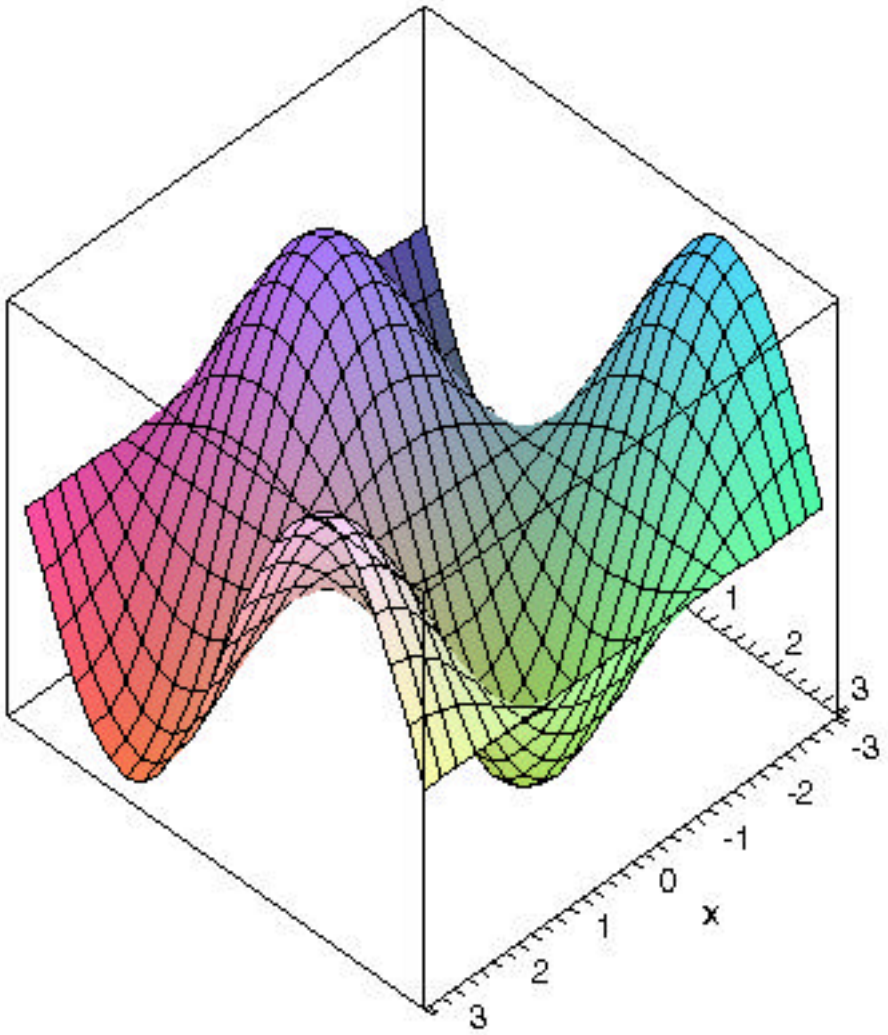
```
> plots[animate](
  sin(x*t),x=-10..10,t=1..2,frames=40,numpoints=200,color=blue,title=`si
  n(x*t)`);
```



Nach der Erstellung der Bilder wird das letzte Bild der Sequenz angezeigt. Nachdem dieses Bild mit der linken Maustaste ausgewählt wurde, kann die Animation über die am oberen Bildschirmrand eingeblendete Funktionsleiste abgespielt werden.

Animation 3-dimensionaler Dartsellungen mit animate3d:

```
> plots[animate3d](cos(t*x)*sin(t*y),x=-Pi..Pi,
  y=-Pi..Pi,t=-1..1,style=patch,          axes=boxed);
```



Animationen können auch mit dem Kommando display aus dem plots Package realisiert werden, wie das nachfolgende Beispiel zur Veranschaulichung einer numerischen Integration zeigt:

```
> f := x -> sin(x)*x+sin(x)+1;
  a:= 0; b:= 2*Pi; n := 30;
  with(plots):with(student):
  for i from 1 by 1 to n do
    summe    := evalf(middlesum(f(x), x=a..b, i)):
    text1    := convert(summe,string):
    text2    := convert(i, string):
    string1  := cat ("Summe = ", text1):
    string2  := cat ("N = ", text2):
    textplot1 := textplot([2,5,string1],font=[HELVETICA,BOLD,12]):
    textplot2 := textplot([2,7,string2],font=[HELVETICA,BOLD,12]):
    graph    := middlebox(f(x), x=a..b, i, color=blue):
    bild||i  := display([textplot1,textplot2,graph]):
  od:
```

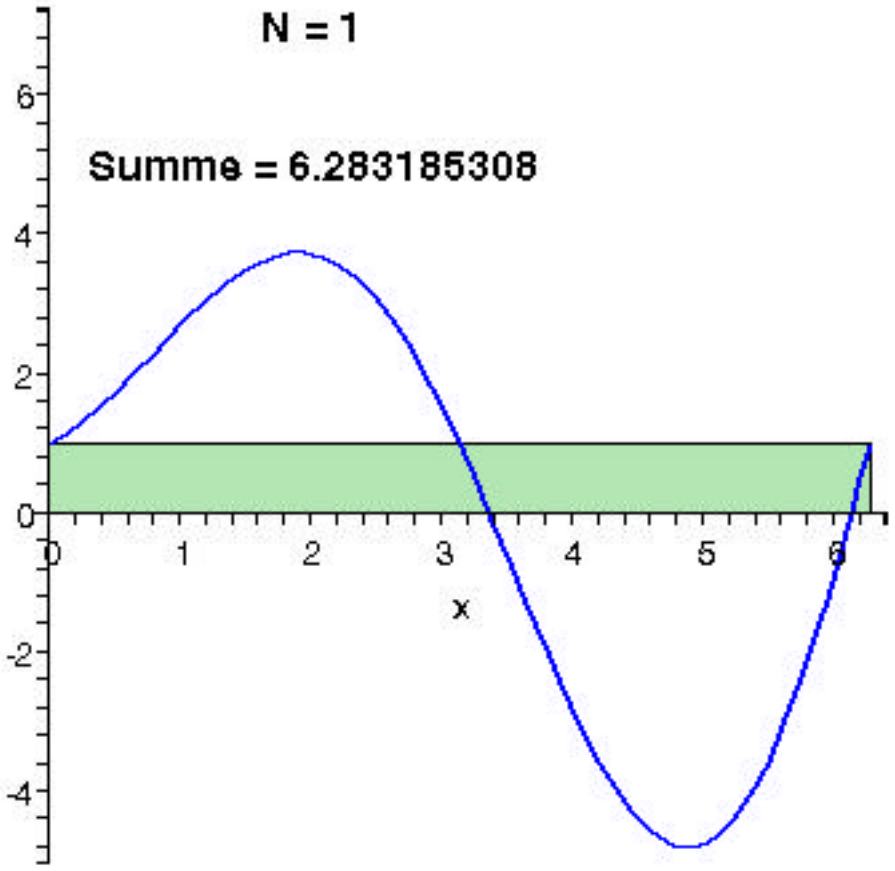
$$f := x \mapsto \sin(x)\,x + \sin(x) + 1$$

$$a := 0$$

$$b := 2\,\pi$$

$$n := 30$$

```
> display(seq(bild||i,i=1..n),insequence=true);
```




```
>
```

Wahl des Ausgabeformates

Um Grafiken in anderen Programmen zu verwenden, können sie z.B. über die Copy-Funktion (Grafik anklicken, danach mit der rechten Maustaste Funktion Copy auswählen) als bitmap-Dateien übernommen werden.

Mit der Funktion plotsetup kann man die Graphikausgabe aber auch in eine Datei umleiten und ein bestimmtes Grafikformat auswählen. Eine Übersicht der unterstützten Grafikformate erhält man mit ?plot[device]

```
> restart;
```

Die nachfolgende Grafik soll im PostScript-Format in die Datei archspir.ps geschrieben werden:

```
> plotsetup(ps,plotoutput=`C:/TEMP/archspir.ps`, plotoptions=`noborder,
```

```
    portrait,       color=rgb`);
> plot([t*cos(t),t*sin(t),t=0..5*Pi], title=`Archimedische Spirale`);

Jetzt werden wieder die Standardwerte für die Grafikausgabe gesetzt.
> plotsetup(default);
>
```

weiter

Funktionen zur Ein-/Ausgabe

N. Geers

Rechenzentrum

Universität Karlsruhe (TH)

e-mail: geers@rz.uni-karlsruhe.de

```
> restart;
```

Dateneingabe und Datenausgabe

```
>
```

Maple V kann Daten aus externen Dateien mittels folgender Funktionen einlesen:

- **readdata liest in Tabellenform vorliegende Daten von einer externen Datei.**
- **readline liest eine Textzeile aus einer Datei.**
- **sscanf, fscanf, scanf formatiertes Lesen**

Die Datei maple.dat enthält folgende Daten:

```
1   3  5
11 33 55
15  5 55
17 78 70
```

Mit readdata("maple.dat",2); werden die beiden ersten Spalten gelesen.

```
> daten1:=readdata("maple.dat",2);
```

$$daten1 := [[1., 3.], [11., 33.], [15., 5.], [17., 78.]]$$

```
> daten:=readdata("maple.dat",integer, 3);
```

$$daten := [[1, 3, 5], [11, 33, 55], [15, 5, 55], [17, 78, 70]]$$

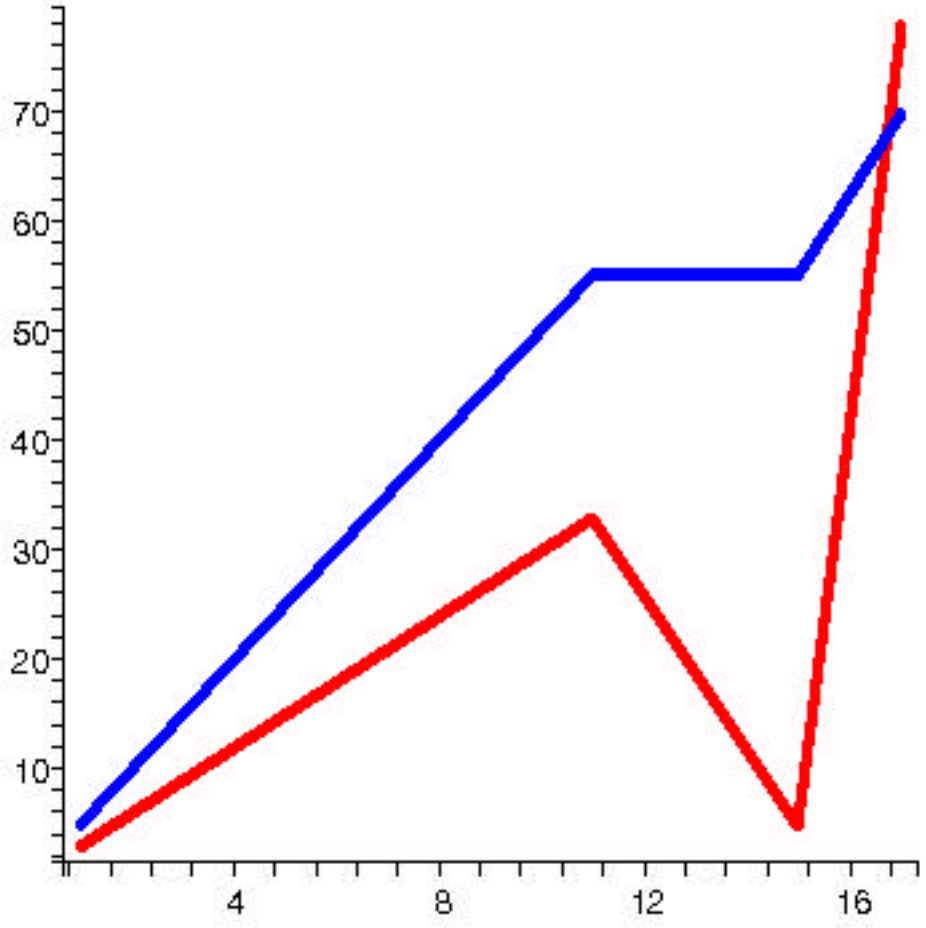
Mit den nachfolgenden Anweisungen werden die Spalten 1 und 3 als ganzzahlige Werte eingelesen.

```
> fd:=fopen("maple.dat",READ);
  daten2 := NULL:
  punkt := fscanf(fd,"%d %*d %d"):
  while punkt <> 0 do
    daten2 := daten2, punkt:
    punkt := fscanf(fd,"%d %*d %d"):
  od:
  fclose(fd):
  daten2 := [daten2];
```

$$fd := 0$$

$$daten2 := [[1, 5], [11, 55], [15, 55], [17, 70]]$$

```
> plot([daten1,daten2],color=[red,blue],thickness=3);
```




```
>
```

Mit der Funktion writedata könne numerische Daten in Tabellenform ausgegeben werden. Beispiel: Es wird eine Wertetabelle einer Funktion f, sowie ihrere ersten und zweiten Ableitung ausgegeben:

```
> f := x-> exp(-x^2) * sin(2*x); f1 := D(f); f2 := D(f1);
```

$$f := x \mapsto e^{(-x^2)} \sin(2 x)$$

$$f1 := x \mapsto -2 x e^{(-x^2)} \sin(2 x) + 2 e^{(-x^2)} \cos(2 x)$$

$$f2 := x \mapsto -6 e^{(-x^2)} \sin(2 x) + 4 x^2 e^{(-x^2)} \sin(2 x) - 8 x e^{(-x^2)} \cos(2 x)$$

```
> x1 := -5:
  x2 := 5:
  n:= 20:
  dx := (x2 -x1) / n:
```

```
> wertetabelle := [ seq (
    [x1+i*dx, f(x1+i*dx), f1(x1+i*dx), f2(x1+i*dx)], i = 0..n)]:
  wertetabelle[1];
```

$$\left[-5, -e^{(-25)}\sin(10), -10\,e^{(-25)}\sin(10)+2\,e^{(-25)}\cos(10), -94\,e^{(-25)}\sin(10)+40\,e^{(-25)}\cos(10)\right]$$

Um die Daten mit writedata ausgeben zu können, müssen sie zunächst mit der Funktion evalf umgewandelt werden.

```
> wertetabelle := map(evalf,wertetabelle):
> writedata ("wertetabelle.txt",wertetabelle,float);
>
```

Weitere Funktionen zur Ein-/Ausgabe

```
> restart;
```

Erstellen eines Funktionsvektors und der zugehörigen Jakobimatrix

```
>
```

Zu einem Funktionsvektor f soll die Jakobimatrix M der partiellen Ableitungen berechnet werden.

Zuerst werden drei Felder *M*, *f* und *x* vereinbart:

```
> M:= array(1..4,1..3):
  f:= array(1..4):
  x:= array(1..3):
> f[1] := x[1] + exp(x[1]-1) + (x[2] + x[3])^2 -Pi:
  f[2] := exp(x[2]-2) / x[1] + x[3]^2 -10:
  f[3] := x[3] + sin(x[2]-2) + x[2]^2 -7:
  f[4]:= sin(x[1]-x[2]^2):
```

Jetzt werden von Maple die partiellen Ableitungen berechnet und im Feld M abgelegt.

```
> for i from 1 by 1  to 4 do
    for j from 1 by 1 to 3 do
      M[i,j] := diff(f[i],x[j]);
    od;
  od;
> evalm(convert(f, matrix));
  evalm(M);
```

$$\begin{bmatrix} x_1 + e^{(x_1 - 1)} + (x_2 + x_3)^2 - p \\ \frac{e^{(x_2 - 2)}}{x_1} + x_3^2 - 10 \\ x_3 + \sin(x_2 - 2) + x_2^2 - 7 \\ \sin\left(x_1 - x_2^2\right) \end{bmatrix}$$

$$\begin{bmatrix} 1 + e^{(x_1 - 1)} & 2 x_2 + 2 x_3 & 2 x_2 + 2 x_3 \\ -\frac{e^{(x_2 - 2)}}{x_1^2} & \frac{e^{(x_2 - 2)}}{x_1} & 2 x_3 \\ 0 & \cos(x_2 - 2) + 2 x_2 & 1 \\ \cos\left(x_1 - x_2^2\right) & -2 \cos\left(x_1 - x_2^2\right) x_2 & 0 \end{bmatrix}$$

>

Weitere Kommandos zur Ein-/Ausgabe

Weitere Kommandos zur Ein-/Ausgabe:

writeto Die nachfolgende Ausgabe wird in eine Datei geschrieben

appendto Die nachfolgende Ausgabe wird an das Ende einer Datei angehängt

save Werte von Variablen werden in einer Datei gesichert

read Es wird die Eingabe von einer Datei gelesen

print Ausgabe der Werte einzelner Ausdrücke

lprint

printf formatierte Ausgabe

readdata Lesen einer Datendatei (ASCII)

readline Lesen einer Zeile von einer Datei

Die Ergebnisse der folgenden Anweisungen werden in die Datei maple.out geschrieben.

```
> writeto(`maple_out.txt`);
> evalm(M);
```

Jetzt wird die Ausgabe wieder auf den Bildschirm umgesteuert

```
> writeto(terminal);
> evalm(M);
```

$$\begin{bmatrix} 1 + e^{(x_1 - 1)} & 2x_2 + 2x_3 & 2x_2 + 2x_3 \\ -\frac{e^{(x_2 - 2)}}{x_1^2} & \frac{e^{(x_2 - 2)}}{x_1} & 2x_3 \\ 0 & \cos(x_2 - 2) + 2x_2 & 1 \\ \cos\left(x_1 - x_2^2\right) & -2\cos\left(x_1 - x_2^2\right) x_2 & 0 \end{bmatrix}$$

Jetzt soll auch der Vektor f als Spaltenvektor in die Datei maple.out geschrieben werden.

```
> appendto(`maple_out.txt`);
> print(`Der Vektor f als Spaltenvektor:`);
> evalm(convert(f,matrix));
```

Jetzt wird die Ausgabe wieder umgesteuert.

```
> writeto(terminal);
```

Mit save werden die ganze Sitzung oder Teile davon gesichert.

Falls der Dateiname mit .m endet, wird in einem internen Format, sonst in einem lesbaren Format geschrieben.

In beiden Fällen kann die Datei wieder mit read eingelesen werden.

Das nächste Kommando erzeugt eine Datei, die Maple-Anweisungen zur Definition von f und M enthält.

```
> save (f, M,`fMsave`);
```

Jetzt werden f und M überschrieben und anschließend wieder gelesen.

```
> f:= 'f'; M:='M';
```

$$f := f$$

$$M := M$$

```
> evalm(M);
```

$$M$$

```
> read (`fMsave`):
> evalm(M);
```

$$\begin{bmatrix} 1 + \mathbf{e}^{(x_1 - 1)} & 2\,x_2 + 2\,x_3 & 2\,x_2 + 2\,x_3 \\ -\dfrac{\mathbf{e}^{(x_2 - 2)}}{x_1^{\,2}} & \dfrac{\mathbf{e}^{(x_2 - 2)}}{x_1} & 2\,x_3 \\ 0 & \cos(x_2 - 2) + 2\,x_2 & 1 \\ \cos\left(x_1 - x_2^{\,2}\right) & -2\cos\left(x_1 - x_2^{\,2}\right) x_2 & 0 \end{bmatrix}$$

```
>
```

LaTeX Ausgabe

Mit der Maple Funktion latex können Ausdrücke im LaTeX-Format ausgegeben werden

```
> 'f[1]' = f[1];
  'f[2]' = f[2];
  'f[3]' = f[3];
  'f[4]' = f[4];
  latex(f);
```

$$f_1 = x_1 + \mathbf{e}^{(x_1 - 1)} + (x_2 + x_3)^{2} - \mathbf{p}$$

$$f_2 = \frac{\mathbf{e}^{(x_2 - 2)}}{x_1} + x_3^{\,2} - 10$$

$$f_3 = x_3 + \sin(x_2 - 2) + x_2^{\,2} - 7$$

$$f_4 = \sin\left(x_1 - x_2^{\,2}\right)$$

```
[x_{{1}}+{e^{x_{{1}}-1}}+ \left( x_{{2}}+x_{{3}} \right) ^{2}-\pi ,{
\frac {{e^{x_{{2}}-2}}}{x_{{1}}}}+{x_{{3}}}^{2}-10,x_{{3}}+\sin
 \left( x_{{2}}-2 \right) +{x_{{2}}}^{2}-7,\sin \left( x_{{1}}-{x_{{2}
}}^{2} \right) ]
```

```
> evalm(M);
  latex (M);
```

$$\begin{bmatrix} 1+e^{(x_1-1)} & 2x_2+2x_3 & 2x_2+2x_3 \\ -\frac{e^{(x_2-2)}}{x_1^2} & \frac{e^{(x_2-2)}}{x_1} & 2x_3 \\ 0 & \cos(x_2-2)+2x_2 & 1 \\ \cos\left(x_1-x_2^2\right) & -2\cos\left(x_1-x_2^2\right)x_2 & 0 \end{bmatrix}$$

```
 \left[ \begin {array}{ccc} 1+{e^{x_{{1}}-1}}&2\,x_{{2}}+2\,x_{{3}}&2
\,x_{{2}}+2\,x_{{3}}\\\noalign{\medskip}-{\frac {{e^{x_{{2}}-2}}}{{x_{
{1}}}^{2}}}&{\frac {{e^{x_{{2}}-2}}}{x_{{1}}}}&2\,x_{{3}}
\\\noalign{\medskip}0&\cos \left( x_{{2}}-2 \right) +2\,x_{{2}}&1
\\\noalign{\medskip}\cos \left( x_{{1}}-{x_{{2}}}^{2} \right) &-2\,
\cos \left( x_{{1}}-{x_{{2}}}^{2} \right) x_{{2}}&0\end {array}
 \right]
```

Falls als zusätzlicher Parameter beim Kommando latex ein Dateiname angegeben wird, erfolgt die Ausgabe in diese Datei.

Auch unter Windows ist als Trennzeichen der einzelnen Verzeichnisse ' / ' anzugeben!

```
> latex(f,`f.tex`);
```

```
> latex(convert(f,matrix),`f_mat.tex`);
```

```
> latex(M,`M.tex`);
```

```
>
```

Erzeugen von Fortran Quellcode

Mit der Funktion fortran wird FORTRAN Quelltext erzeugt.

```
> codegen[fortran](f);
```

```
      f(1) = x(1)+exp(x(1)-1)+(x(2)+x(3))**2-0.3141592653589793D1
      f(2) = exp(x(2)-2)/x(1)+x(3)**2-10
```

```
      f(3) = x(3)+sin(x(2)-2)+x(2)**2-7
      f(4) = sin(x(1)-x(2)**2)
```

Über die Variable precision wird festgelegt, ob im FORTRAN Quelltext der Datentyp REAL oder DOUBLE PRECISION benutzt wird.

```
> precision:=double;
  codegen[fortran](f);
```

$$precision := double$$

```
      f(1) = x(1)+exp(x(1)-1)+(x(2)+x(3))**2-0.3141592653589793D1
      f(2) = exp(x(2)-2)/x(1)+x(3)**2-10
      f(3) = x(3)+sin(x(2)-2)+x(2)**2-7
      f(4) = sin(x(1)-x(2)**2)
```

Durch die Option mode=generic in der nachfolgenden Anweisung wird erreicht, dass die generische Form der Fortran Standardfunktionen in den arithmetischen Ausdrücken verwendet wird.
Die Option optimized bewirkt, dass der Fortran Quelltext optimiert wird, d.h. gemeinsame Teilausdrücke werden nur einmal berechnet.

Jetzt soll der Quelltext in eine Datei geschrieben werden.

Die Ausgabe wird jeweils an das Ende der Datei anhängt.

```
> codegen[fortran](f,filename=`fmat.f`,mode=generic, optimized);
  codegen[fortran](M,filename=`fmat.f`,mode=generic, optimized);
```

Auch Maple Prozeduren (z.B. Funktionen) können als Fortran Unterprogramme ausgegeben werden:

```
> funktion := x -> sin(x^2)-cos(x)/x^4;
  f1      := D(funktion);
  codegen[fortran](f1, mode=generic);
```

$$funktion := x \mapsto \sin(x^2) - \frac{\cos(x)}{x^4}$$

$$f1 := x \mapsto 2\cos(x^2)\,x + \frac{\sin(x)}{x^4} + \frac{4\cos(x)}{x^5}$$

```
c The options were    : operatorarrow
      doubleprecision function f1(x)
      doubleprecision x

        f1 = 2*cos(x**2)*x+sin(x)/x**4+4*cos(x)/x**5
        return
      end
```

>

Erzeugen von C Quellcode

Analog zu FORTRAN kann auch C-Code mit der Funktion C erzeugt werden:

```
> codegen[C](f);
```

```
      f[0] = x[0]+exp(x[0]-1.0)+pow(x[1]+x[2],2.0)-0.3141592653589793E1;
      f[1] = exp(x[1]-2.0)/x[0]+x[2]*x[2]-10.0;
      f[2] = x[2]+sin(x[1]-2.0)+x[1]*x[1]-7.0;
      f[3] = sin(x[0]-x[1]*x[1]);
```

```
> codegen[C] (M);
```

```
      M[0][0] = 1.0+exp(x[0]-1.0);
      M[0][1] = 2.0*x[1]+2.0*x[2];
      M[0][2] = 2.0*x[1]+2.0*x[2];
      M[1][0] = -exp(x[1]-2.0)/(x[0]*x[0]);
      M[1][1] = exp(x[1]-2.0)/x[0];
      M[1][2] = 2.0*x[2];
      M[2][0] = 0.0;
      M[2][1] = cos(x[1]-2.0)+2.0*x[1];


      M[2][2] = 1.0;
      M[3][0] = cos(x[0]-x[1]*x[1]);
      M[3][1] = -2.0*cos(x[0]-x[1]*x[1])*x[1];
      M[3][2] = 0.0;
```

```
> codegen[C] (f1, optimized);
```

```
/* The options were    : operatorarrow */
#include <math.h>
double f1(x)
double x;
{
  double t1;
  double t2;
  double t5;
  double t6;
  double t9;
  {
    t1 = x*x;
    t2 = cos(t1);
    t5 = sin(x);
    t6 = t1*t1;
    t9 = cos(x);
    return(2.0*t2*x+t5/t6+4.0*t9/t6/x);
  }
}
```

```
> codegen[C] (f,filename=`fmat.c`);
```

```
> codegen[C] (M,filename=`fmat.c`);
```

Wenn die Option optimized als Parameter von fortran bzw. C angegeben wird, wird optimierter Quelltext erzeugt.

Mit der Funktion cost aus der Miscellaneous Library kann die Anzahl der notwendigen arithmetischen Operationen bestimmt werden.

Programmieren in Maple

N. Geers

Rechenzentrum

Universität Karlsruhe (TH)

e-mail: geers@rz.uni-karlsruhe.de

In diesem Maple Arbeitsblatt sind einige einfache Beispiele zur Erstellung von Maple Programmen aufgeführt. Weitere Informationen zur Programmierung mit Maple findet man in der umfangreichen Literatur zu Maple.

```
> restart;
```

Eine einfache Maple Prozedur

```
> addition:=proc(a,b)
  # Diese Prozedur addiert die beiden Argumente a und b
    a+b;
  end;
```

$$addition := \mathbf{proc}(a, b)\ a + b\ \mathbf{end\ proc}$$

```
> addition(5,6);
```

$$11$$

```
> u := x^2 - 6;
  v := 3 *x^2 - 5 * x;
  addition(u,v);
```

$$u := x^2 - 6$$

$$v := 3\,x^2 - 5\,x$$

$$4\,x^2 - 6 - 5\,x$$

```
> zz := addition(y, x^2*b);
```

$$zz := y + x^2\,b$$

```
> mittelwert := proc(l)
  # Berechnung des Mittelwerts der Elemente der Liste L
    local summe, i;                 # <-----
    n := nops(l);
    summe := sum(l[i], i = 1..n);
    summe/n;                        # <-----
  ### WARNING: `n` is implicitly declared local
```

```
    end;
```

```
Warning, `n` is implicitly declared local to procedure `mittelwert`
```

```
mittelwert := proc(l) local summe, i, n; n := nops(l); summe := `sum`(l[i], i = (1 .. n)); summe/n end proc
```

```
> liste := [ 1, 3, 5, 9];
  mittelwert(liste);
  m := mittelwert([1, 2, 3, 4, 5, 6, 7, 8, 9, 10]);
```

$$liste := [1, 3, 5, 9]$$

$$\frac{9}{2}$$

$$m := \frac{11}{2}$$

```
>
```

Typedefinition der Parameter

```
>
```

```
> mittelwert := proc(l :: list)   # <-----
  # Berechnung des Mittelwerts der Elemente der Liste L
    local summe, n, i;
    n := nops(l);
    summe := sum(l[i], i = 1..n);
    summe / n;
  end;
```

```
mittelwert := proc(l::list)
local summe, n, i;
      n := nops(l);
      summe := `sum`(l[i], i = (1 .. n));
      summe/n
end proc
```

```
>
  mittelwert (1,b);
```

```
Error, invalid input: mittelwert expects its 1st argument, l, to be of type list,
but received 1
```

```
> mittelwert(liste);
```

$$\frac{9}{2}$$

```
> mittelwert := proc(l :: { list, set } )      # <-----
  # Berechnung des Mittelwerts der Elemente der Liste L
    local summe, n, i;
    n := nops(l);
    summe := sum(l[i], i = 1..n);
    summe/n;
```

```
    end;

mittelwert := proc(l::{list, set})
local summe, n, i;
      n := nops(l);
      summe := `sum`(l[i], i = (1 .. n));
      summe/n
end proc
```

```
> menge := { 2, 3, 5, 6, 3 };
  liste := [a, b, c];
  m := mittelwert(menge);
  l := mittelwert(liste);
```

$$menge := \{2, 3, 5, 6\}$$

$$liste := [a, b, c]$$

$$m := 4$$

$$l := \frac{a}{3} + \frac{b}{3} + \frac{c}{3}$$

```
>
```

Weiteres zur Parameterübergabe

```
> maximum := proc ()
    local i, max, n;
    n := nargs;                    # <-----
    if n = 0 then
      ERROR(`Es muss mindestens ein Parameter übergeben werden.`)
    fi;
    max := args[1];                # <-----
    for i from 2 to n do
      if max < args[i] then
        max := args[i]
      fi
    od;
    max;
  end:
```

```
> maximum(5, 7);
  maximum(1, 8, 4, 2);
  maximum();
```

$$7$$

$$8$$

```
Error, (in maximum) Es muss mindestens ein Parameter übergeben werden.
```

```
>
```

```
> test_prozedur := proc()
    nargs
```

```
end;
```

```
test_prozedur := proc() nargs end proc
```

```
> s:= seq(i, i = 1..100):
  l:= [s]:
  test_prozedur(s);
  test_prozedur(l);
  test_prozedur(s, l);
```

$$100$$

$$1$$

$$101$$

Wenn eine Folge als Parameter an eine Prozedur übergeben wird, wird jedes einzelne Element der Folge als separater Parameter angesehen.

```
>
```

Ergebnis des Prozeduraufrufs, RETURN

```
> my_member := proc (x :: anything, l :: {set, list} )
  # my_member bestimmt, ob das Element x in der Liste oder Menge l
  # enthalten ist.
  local i;
  for i from 1 by 1 to nops(l) do
    if x = l[i] then
      RETURN(true)                    # <-----
    fi
  od;
  false
  end;
```

```
my_member := proc(x::anything, l::{list, set})
local i;
    for i to nops(l) do if x = l[i] then RETURN(true) end if end do
    false
end proc
```

```
> liste := [ x, x^2, x^3, y^2, z, 5 ];
  my_member(x^2, liste);
  my_member(x*x, liste);
  my_member(t^2, liste);
```

$$liste := [x, x^2, x^3, y^2, z, 5]$$

$$true$$

$$true$$

$$false$$

```
> x:= 'x': sin(x);
  maximum(1,x, 3);
```

$$\sin(x)$$

```
Error, (in maximum) cannot determine if this expression is true or false: -x < -1
```

```
> maximum := proc ()
    local i, max, n;
    n := nargs;
    if n = 0 then
      ERROR(`Es muss mindestens ein Parameter uebergeben werden.`)
    fi;
    max := args[1];
    for i from 2 to n do

  # Das Maximum kann nur dann bestimmt werden,
  # wenn alle Argumente reellen Konstanten sind.

      if not type(args[i], realcons) then   # <-----
        RETURN('procname'(args));         # <-----
       fi;
      if max < args[i] then
        max := args[i]
      fi
    od;
    RETURN(max);
  end:
```

```
> u:= 50/3;
  maximum(1, 7, 5.5, u);
  maximum(1, 3, 2+3*I);
  maximum(a, b, u^2);
```

$$u := \frac{50}{3}$$

$$\frac{50}{3}$$

$$\mathit{maximum}(1, 3, 2 + 3\,I)$$

$$\mathit{maximum}\left(a, b, \frac{2500}{9}\right)$$

```
>
```

```
> my_member_1 := proc (x :: anything, l :: {set, list}, position :: name
  )
  # my_member_1 bestimmt, ob das Element x in der Liste oder Menge l
  # enthalten ist.
  local i;
  for i from 1 by 1 to nops(l) do
    if x = l[i] then
      if nargs = 3 then position := i fi;   # <-----
      RETURN(true)
    fi
```

```
    od;
    false
    end:
```

```
> liste := [ x, x^2, x^3, y^2, z, 5 ];
  my_member_1(x^2, liste);
  my_member_1(x*x, liste, 'i');
  i;
```

$$liste := [x, x^2, x^3, y^2, z, 5]$$

$$true$$

$$true$$

$$2$$

```
> my_member_1(x*x, liste, i);
Error, invalid input: my_member_1 expects its 3rd argument, position, to be of type
name, but received 2
```

```
> my_member_2 := proc (x :: anything, l :: {set, list}, position ::
  evaln   )
  # my_member_2 bestimmt, ob das Element x in der Liste oder Menge l
  # enthalten ist.
  local i;
  for i from 1 by 1 to nops(l) do
    if x = l[i] then
      if nargs = 3 then position := i fi;   # <-----
      RETURN(true)
    fi
  od;
  false
  end:
```

```
> liste := [ x, x^2, x^3, y^2, z, 5 ];
  my_member_2(x^3, liste);
  my_member_2(x*x^2, liste, i);
  i;
  my_member_2(x*x^2, liste, 'i');
```

$$liste := [x, x^2, x^3, y^2, z, 5]$$

$$true$$

$$true$$

$$3$$

```
Error, illegal use of an object as a name
```

```
>
```

Option remember

```
>
```

```
> restart;
```

```
> fibonacci_1 := proc(n :: nonnegint)
      if n < 2 then
         n
      else
         fibonacci_1(n-1) + fibonacci_1(n-2)
      fi
   end;
```

```
fibonacci_1 := proc(n::nonnegint)
      if n < 2 then n else fibonacci_1(n - 1) + fibonacci_1(n - 2) end if
end proc
```

```
> fibonacci_2 := proc(n :: nonnegint)
      option remember;              # <-----
      if n < 2 then
         n
      else
         fibonacci_2(n-1) + fibonacci_2(n-2)
      fi
   end;
```

```
fibonacci_2 := proc(n::nonnegint)
option remember;
      if n < 2 then n else fibonacci_2(n - 1) + fibonacci_2(n - 2) end if
end proc
```

```
> t1:=time(): f1:=fibonacci_1(25); t1 := time() - t1;
  t2:=time(): f2:=fibonacci_2(25); t2 := time() - t2;
```

$$f1 := 75025$$

$$t1 := 1.623$$

$$f2 := 75025$$

$$t2 := 0.$$

```
>
```

Speichern einer Prozedur im internen Format

```
> fibonacci:=fibonacci_2;
```

$$fibonacci := fibonacci\_2$$

Eine Prozedur kann mittels save z.B. als .m File gesichert werden.

```
> save(fibonacci,`fibonacci.m`);
```

```
> fibonacci:='fibonacci';
  fibonacci(10);
```

$$fibonacci := fibonacci$$

fibonacci(10)

Ein solche .m file wird mittels read wieder eingelesen.

```
> read("fibonacci.m");
> fibonacci(10);
```

55

Prozeduren können selbstverständlich auch mit einem Texteditor als ASCII-Datei erstellt und von Maple mit der read -Anweisung eingelesen werden.

```
>
```

Quelltexte bereits existierender Maple Prozeduren

```
>
> interface(verboseproc=2);
> print(sin);
>
```

Erstellen eigener Packages

Ein Package wird als Tabelle definiert, wobei jeder Tabelleneintrag eine Prozedur des Packages bezeichnet.
Im Folgenden wird ein Package kdis mit verschiedenen Funktionen zur Kurvendiskussion einer Funktion einer reellen Variablen bereitgestellt. Das Package enthält die Prozeduren

- **nullstellen zur Berechnung der Nullstellen,**
- **extrema zur Berechnung der Hoch- und Tiefpunkte,**
- **wendepunkte zur Berechnung der Wendepunkte des Funktionsgraphen und**
- **zeichne zum zeichnen der Funktion, ihrer ersten und zweiten Ableitung sowie der zuvor berechneten Nullstellen, Extrem- und Wendepunkte**

```
> restart;
> kdis := table();
```

kdis := *TABLE*([])

Die Prozedur nullstellen

```
> kdis[nullstellen] := proc(f :: procedure)
  # Bestimme die reellen Nullstellen einer Funtion f(x)

    local loesungen, ergebnis, s;
    ergebnis := NULL;
    loesungen := solve (f(x) = 0, x);
    for s in loesungen do
     if type(s, realcons) then ergebnis := ergebnis, evalf(s); fi;
    od;
```

```
    [ergebnis];
  end:
```

Die Prozedur extrema

```
> kdis[extrema] := proc(f:: procedure, hochpunkte :: evaln,
  tiefpunkte     :: evaln)
  # Berechne die Extremstellen einer Funktion f(x)

    local f1, f2, xl, i, x1, x2, f1x1, f1x2, hp, tp;

    hp := NULL; tp := NULL;

    f1 := D(f);
    xl := sort ( kdis [nullstellen] (f1) );
  #
  # Mögliche Extremstellen sind die Nullstellen der ersten Ableitung.
  # Bestimme jetzt die Extremstellen mit dem
  Vorzeichenwechselkriterium.
  #
    x1 := xl[1] - 1;
    f1x1 := evalf( f1(x1) );
    for i from 1 by 1 to nops(xl) do
      if i = nops(xl) then
        x2 := xl[i] + 1;
      else
        x2 := (xl[i] + xl[i+1] ) / 2;
      fi;
      f1x2 := evalf( f1(x2) );
      if ( f1x1 < 0 and f1x2 > 0 ) then
        tp := tp, [ evalf(xl[i]), evalf(f(xl[i])) ] ;
      elif ( f1x1 > 0  and f1x2 < 0 ) then
        hp := hp, [ evalf(xl[i]), evalf(f(xl[i])) ] ;
      fi;
      x1 := x2; f1x1 := f1x2;
    od:
    hochpunkte := [hp]; tiefpunkte := [tp];
    RETURN(nops([hp,tp]));
  end:
```

Die Prozedur wendepunkte

```
> kdis[wendepunkte] := proc(f :: procedure, punkte :: evaln)
  # Berechne die Wendepunkte eines Funktionsgraphen.
  # Eine Wendestelle ist eine Extremstelle des Graphen der
  Ableitungsfunktion.

    local f1, i, n, hochpunkte, tiefpunkte, p;

    n := kdis[extrema] (D(f), hochpunkte, tiefpunkte);
    p := [ hochpunkte[], tiefpunkte[] ];
    punkte := [ seq ( [ p[i][1], f(p[i][1]) ], i = 1..nops(p)) ];
    RETURN(n);
  end:
```

Die Prozedur zeichne

```
> kdis[zeichne] := proc(f :: procedure, x_bereich :: range, y_bereich
  :: range)
  # Zeiche das Ergebnis der Kurvendiskussion

  local nst, n_extrema, n_wendepunkte,
      hochpunkte, tiefpunkte, wendepkt, l,
      plot_graph, plot_nullstellen, plot_hochpunkte,
      plot_tiefpunkte, plot_wendepunkte,
      graphen,
      i;

  nst := kdis[nullstellen] (f);
  n_extrema := kdis[extrema] (f, hochpunkte, tiefpunkte);
  n_wendepunkte := kdis[wendepunkte] (f, wendepkt);

  graphen := plot([f(x), D(f)(x), (D@@2)(f)(x)],  x = x_bereich, y =
  y_bereich,     discont = true, color=[black, red, blue]):

  if (nops(nst) <> 0) then
   l := [ seq( [ nst[i], f(nst[i]) ], i=1..nops(nst)) ];
   graphen := graphen, plots[pointplot]( l, symbol = circle,
                color = red):
  fi:

  if (nops(hochpunkte) <> 0) then
   graphen := graphen, plots[pointplot]( hochpunkte,
                symbol = diamond, color = blue):
  fi:
  if (nops(tiefpunkte) <> 0) then
   graphen := graphen, plots[pointplot]( tiefpunkte,
                symbol = diamond, color = blue):
  fi:
  if (nops(wendepkt) <> 0) then
   graphen := graphen,plots[pointplot]( wendepkt,
                symbol = box, color = cyan):
  fi:

  plots[display](graphen);

  end:
```

Aufruf der Prozeduren

**Nachdem ein Package erstellt wurde, können die einzelnen Prozeduren in der Form *packagename*[*prozedurname*] (*argumente*);
aufgerufen werden.**

```
> g:= x-> (x-1.3) * (x+2) *(x+1) * (x-3);
```

$$g := x \mapsto (x - 1.3)\,(x + 2)\,(x + 1)\,(x - 3)$$

```
> nst := kdis[nullstellen](g):
  e := kdis[extrema](g, hp, tp):
  w := kdis[wendepunkte](g, wp):
  txt := convert (nst, symbol):
```

```
print (`Die Nullstellen sind: ` || txt);
print (`Es gibt ` || e || ` Extremstellen.`):
if (nops(hp) <> 0) then
  txt := convert (hp, symbol):
  print (`Die Hochpunkte sind: ` || txt) fi:
if (nops(tp) <> 0) then
  txt := convert (tp, symbol):
  print (`Die Tiefpunkte sind: ` || txt) fi:
print (`Es gibt ` || w || ` Wendepunkte.`):
if (nops(wp) <> 0) then
  txt := convert (wp, symbol):
  print (`Die Wendepunkte sind: ` || txt) fi:
kdis[zeichne](g,-2.5..3.5, -40 ..40);
```

Die Nullstellen sind: [-1., -2., 3., 1.300000000]

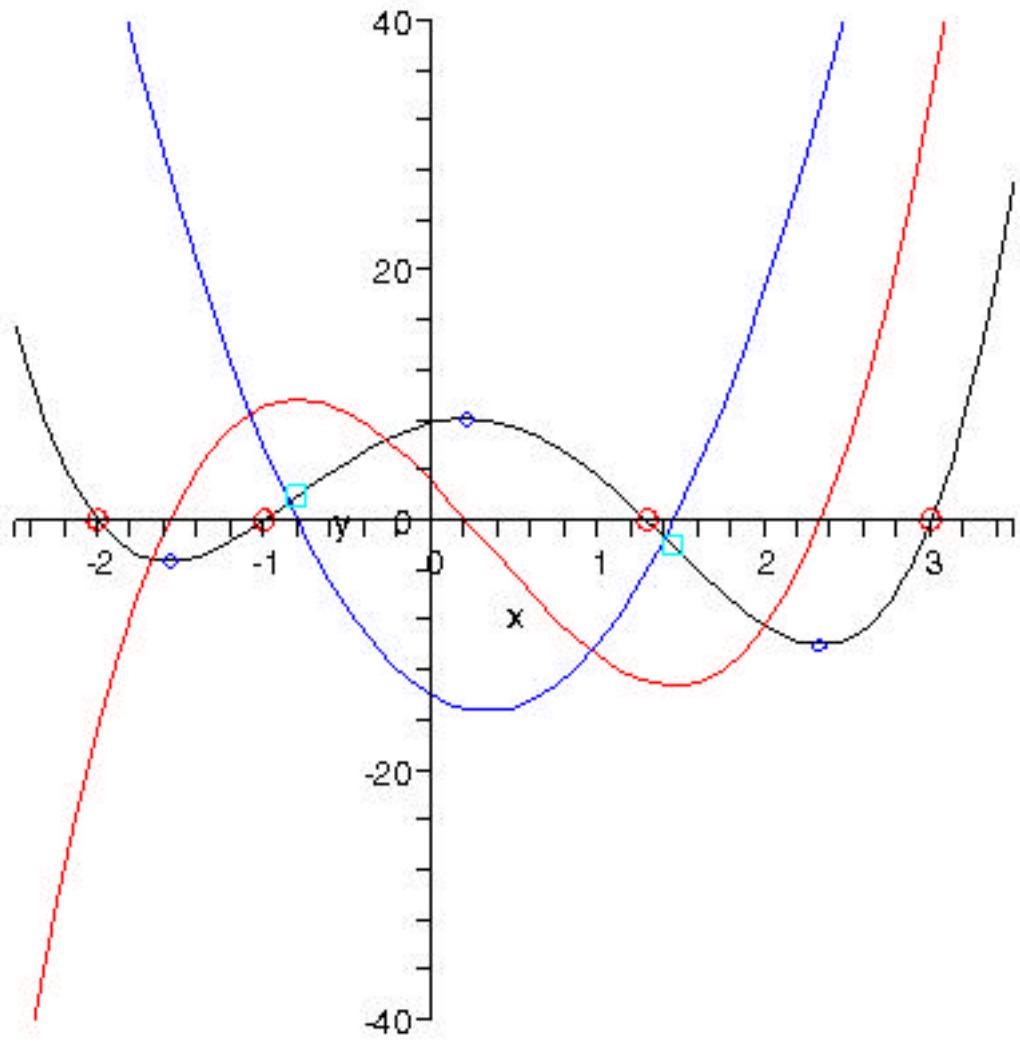
Es gibt 3 Extremstellen.

Die Hochpunkte sind: [[.2116579890, 8.132226279]]

Die Tiefpunkte sind: [[-1.569544050, -3.214709980], [2.332886061, -9.950620593]]

Es gibt 2 Wendepunkte.

Die Wendepunkte sind: [[-.8029590714, 1.886331066], [1.452959071, -2.004281756]]



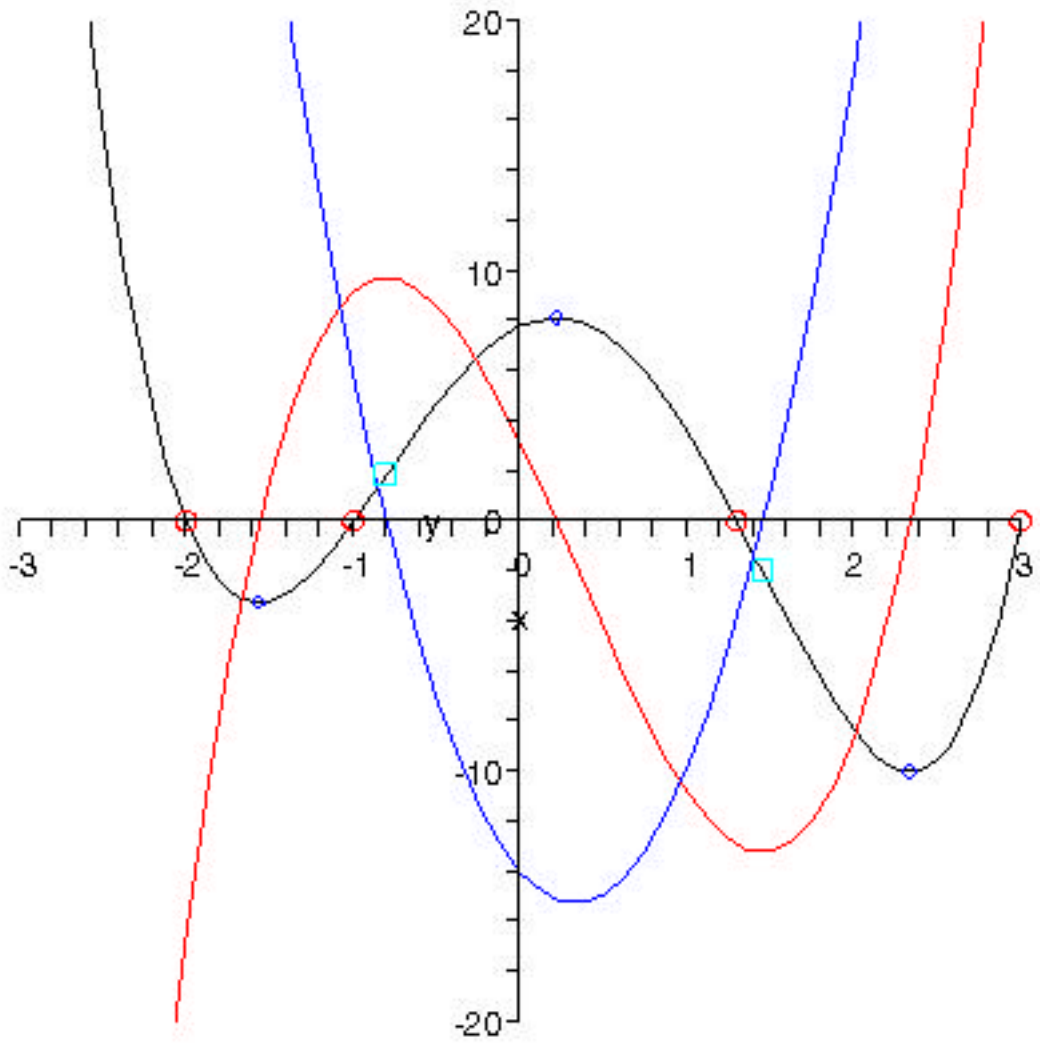

```
> with(kdis);
```

```
Warning, the protected name extrema has been redefined and unprotected
```

[*extrema*, *anullstellen*, *wendepunkte*, *zeichne*]

```
> nst := nullstellen(g);
  zeichne(g,-3..3,-20..20);
```

nst := [-1., -2., 3., 1.300000000]




```
>
```

Speichern und Laden eines Package

```
>
```

```
> save(kdis, `kdis.m`);
```

```
> restart;
  kdis;
  nullstellen(g);
```

kdis

nullstellen(*g*)

```
> libname;
```

"C:\Programme\Maple 9/lib"

```
> libname := "./", libname;
```

$$libname := \text{"./"}, \text{"C:\Programme\Maple 9/lib"}$$

```
>
> with(kdis);

Warning, the protected name extrema has been redefined and unprotected
```

$$[extrema, nullstellen, wendepunkte, zeichne]$$

```
> f := x -> 2*(x-2.4)*(x-2.8)*(x+1)*(x+2)/(x-2.5);
  nullstellen(f);
  extrema(f, hp, tp); hp; tp;
  wendepunkte(f, wp); wp;
  zeichne(f,-3..3.5,-30..30);
```

$$f := x \mapsto \frac{2\,(x - 2.4)\,(x - 2.8)\,(x + 1)\,(x + 2)}{x - 2.5}$$

$$[-1., -2., 2.400000000, 2.800000000]$$

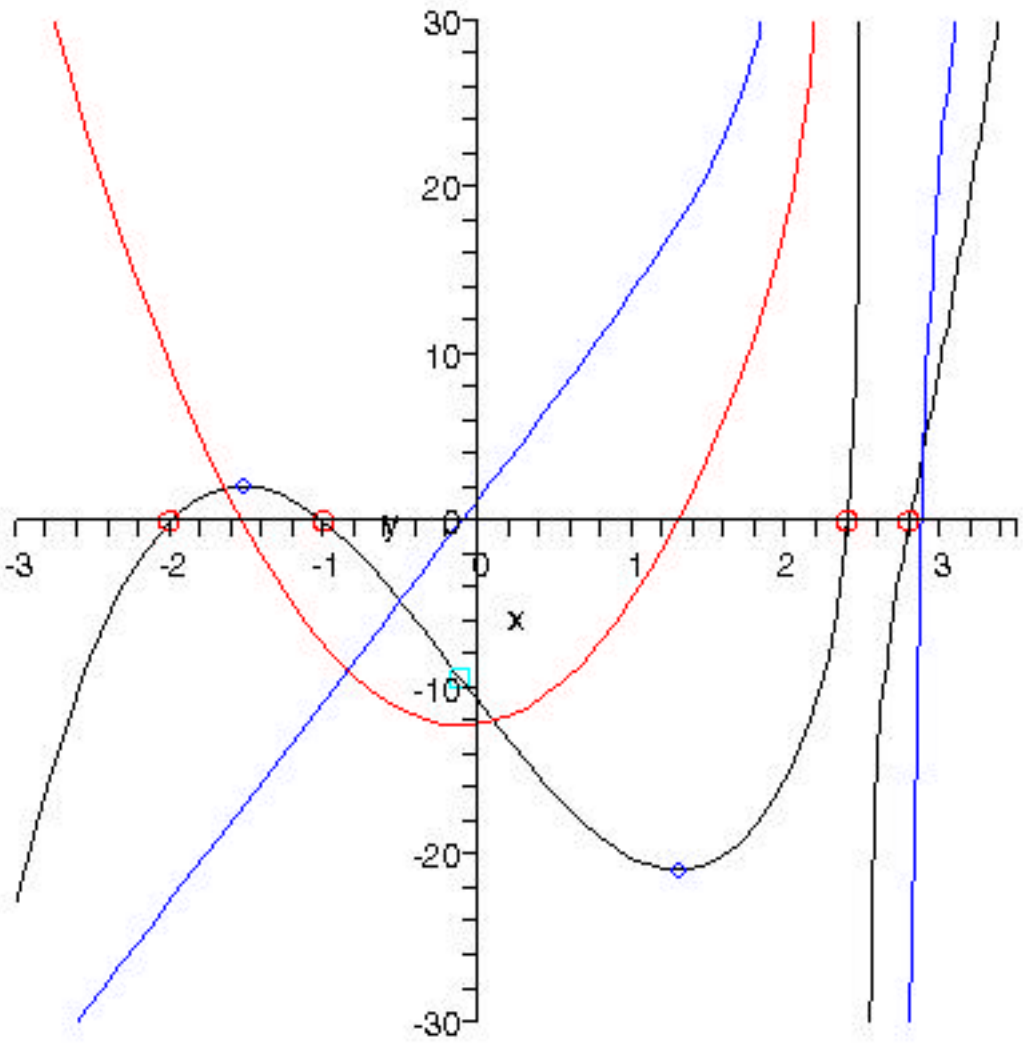
$$2$$

$$[[-1.529557086, 2.103678996]]$$

$$[[1.294609075, -20.87275926]]$$

$$1$$

$$[[-.1088699924, -9.428497294]]$$



v

zurück

Lineare Algebra mit Maple

N. Geers

Rechenzentrum

Universität Karlsruhe (TH)

e-mail: geers@rz.uni-karlsruhe.de

```
> restart;
```

Das Paket linalg

Zuerst muss das Package für Lineare Algebra geladen werden:

```
> with(linalg):
```

```
Warning, the protected names norm and trace have been redefined and unprotected
```

Wird der Befehl mit Semikolon anstatt mit Doppelpunkt abgeschlossen, so werden die verschiedenen zur Verfügung stehenden Befehle des Packages angezeigt .

Definition von Matrizen

Erzeugung von Matrizen:

```
> A:=matrix([[3,5],[7,8]]);
```

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

```
> B:=randmatrix(2,2);
```

$$B := \begin{bmatrix} -85 & -55 \\ -37 & -35 \end{bmatrix}$$

```
> C:=matrix(2,2,(i,j)->i^j);
```

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

```
> DMAT:=matrix(2,2,[a,b,c,d]);
```

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

```
> SYM:=array(1..4,1..4,symmetric);
```

$$SYM := array(symmetric, 1 .. 4, 1 .. 4, [])$$

```
> for i from 1 by 1 to 4 do
>     for j from 1 by 1 to i do
>         SYM[i,j] :=  x^i + y^j
>     od
> od;
```

Oder in Kurzschreibweise:

```
for i to 4 do for j  to i do SYM[i,j] := x^i + y^j od od;
```

```
> evalm(SYM);
```

$$\begin{bmatrix} x+y & x^2+y & x^3+y & x^4+y \\ x^2+y & x^2+y^2 & x^3+y^2 & x^4+y^2 \\ x^3+y & x^3+y^2 & x^3+y^3 & x^4+y^3 \\ x^4+y & x^4+y^2 & x^4+y^3 & x^4+y^4 \end{bmatrix}$$

```
> f:=proc(i,j) if i=j then i+j-1 else i+j+1 fi end;
```

$$f := \mathbf{proc}(i, j)\ \mathbf{if}\ i = j\ \mathbf{then}\ i + j - 1\ \mathbf{else}\ i + j + 1\ \mathbf{end\ if}\ \mathbf{end\ proc}$$

```
> H:=matrix(3,3,f);
```

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

```
> M:=matrix([[2,-1,2],[-1,2,-2],[2,-2,5]]);
```

$$M := \begin{bmatrix} 2 & -1 & 2 \\ -1 & 2 & -2 \\ 2 & -2 & 5 \end{bmatrix}$$

Matrix- und Vektoroperationen

Verändern eines Elements:

```
> A[1,2]:=17:
```

```
> evalm(A);
```

$$\begin{bmatrix} 3 & 17 \\ 7 & 8 \end{bmatrix}$$

Bildung der Inversen bei Quadratischen Matrizen

```
> Ainv := inverse(A);
```

$$Ainv := \begin{bmatrix} \frac{-8}{95} & \frac{17}{95} \\ \frac{7}{95} & \frac{-3}{95} \end{bmatrix}$$

Dies wäre auch durch die Notation evalm(A^(-1)) oder evalm(1/A) möglich gewesen:

Multipliziert werden Matizen mit dem Operator &*. Der Operator * bezeichnet die Multiplikation mit einem Skalar.

```
> B &* C + 2 * C;
```

$$\&*(B, C) + 2\,C$$

```
> evalm(%);
```

$$\begin{bmatrix} -193 & -303 \\ -103 & -169 \end{bmatrix}$$

```
> EI:=evalm(A &* Ainv);
```

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

```
> evalm(A &* A^(-1));
```

$$\&*()$$

Mit evalm() werden nicht nur die einzelnen Elemente der Matrix berechnet, sondern auch Matrixausdrücke ausgewertet.

Die Schreibweise &*() steht in Maple für die Einheitsmatrix. Falls man eine andere Schreibweise bevorzugt, kann dies z.B. mit der alias-Funktion definiert werden.

```
> alias(Id = &*());
  evalm(A &* A^(-1));
```

$$Id$$

$$Id$$

```
> S:=vector(2,n->x^n);
```

$$S := [x, x^2]$$

```
> evalm(S); S[1];S[2];
```

$$[x, x^2]$$

$$x$$

$$x^2$$

Matrix-Vektor-Multiplikationen:

```
> evalm(DMAT);
  evalm(DMAT &* S);
  evalm(S &* DMAT);
```

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

$$[a x + b x^2, c x + d x^2]$$

$$[a x + x^2 c, x b + d x^2]$$

Um Zeilenvektoren in Spaltenvektoren umzuwandeln oder umgekehrt, müssen wir mit der Funktion convert den Vektor in eine Matrix umwandeln und den transpose Befehl anwenden.

```
> F:=convert(S,matrix);
```

$$F := \begin{bmatrix} x \\ x^2 \end{bmatrix}$$

```
> FT:=transpose(F);
```

$$FT := [x \quad x^2]$$

```
> GL:=evalm(DMAT &* F);
```

$$GL := \begin{bmatrix} a x + b x^2 \\ c x + d x^2 \end{bmatrix}$$

Mit der map Prozedur kann eine Operation auf alle Matrixelemente angewendet werden.

```
> map(factor,GL);
```

$$\begin{bmatrix} x (x b + a) \\ x (c + x d) \end{bmatrix}$$

```
> map(sqrt,A);
```

$$\begin{bmatrix} \sqrt{3} & \sqrt{17} \\ \sqrt{7} & 2\sqrt{2} \end{bmatrix}$$

```
> U1:=vector([3,6,9]);U2:=vector([4,1,1]);
```

$$U1 := [3, 6, 9]$$

$$U2 := [4, 1, 1]$$

Die k-Norm eines Vektors wird mit der Funktion norm berechnet:

```
> norm(U1,2);normalize(U1);
```

$$3\sqrt{14}$$

$$\left[\frac{1}{14}\sqrt{14}, \frac{1}{7}\sqrt{14}, \frac{3}{14}\sqrt{14}\right]$$

Winkel zwischen U1 und U2:

```
> angle(U1,U2);evalf(%*180/Pi);
```

$$\arccos\left(\frac{1}{84}\sqrt{126}\sqrt{18}\right)$$

$$55.46241623$$

Die Multiplikation von Vektoren läßt sich mit den beiden Befehlen dotprod für das Skalareprodukt und crossprod für das Kreuzprodukt durchführen. Beide Kommandos verarbeiten nur Zeilenvektoren!

```
> dotprod(U1,U2);
```

$$27$$

```
> V1:=vector([b1,b2,b3]);V2:=vector([a1,a2,a3]);
```

$$V1 := [b1, b2, b3]$$

$$V2 := [a1, a2, a3]$$

```
> crossprod(V1,V2);
```

$$[b2\,a3 - b3\,a2, b3\,a1 - b1\,a3, b1\,a2 - b2\,a1]$$

```
> convert(%,matrix);
```

$$\begin{bmatrix} b2\,a3 - b3\,a2 \\ b3\,a1 - b1\,a3 \\ b1\,a2 - b2\,a1 \end{bmatrix}$$

Lineare Gleichungssysteme

Gleichungssysteme der Form A* x= b werden mit linsolve gelöst.

```
> print(A);b:=vector([1,1]);
```

$$\begin{bmatrix} 3 & 17 \\ 7 & 8 \end{bmatrix}$$

$$b := [1, 1]$$

```
> Erg:=linsolve(A,b);
```

$$Erg := \begin{bmatrix} \frac{9}{95}, & \frac{4}{95} \end{bmatrix}$$

Eigenwerte und Eigenvektoren

Berechnung der Eigenwerte und Eigenvektoren

Bestimmung der Nullstellen des characteristischen Polynoms

```
> alias(Id=&*());
```

$$Id$$

Id kann jetzt als Schreibweise für die Einheitsmatrix verwendet werden. (I steht für die Imaginäre Einheit.)

```
> CM:=evalm(x&*Id -M);
```

$$CM := \begin{bmatrix} -2 + x & 1 & -2 \\ 1 & -2 + x & 2 \\ -2 & 2 & -5 + x \end{bmatrix}$$

```
> p:=det(%);
```

$$p := -7 + 15\,x - 9\,x^2 + x^3$$

```
> lambda:=solve(p=0,x);
```

$$l := 7, 1, 1$$

Einsetzen der Eigenwerte

```
> CM1:=subs(x=lambda[1],evalm(CM));
```

$$CM1 := \begin{bmatrix} 5 & 1 & -2 \\ 1 & 5 & 2 \\ -2 & 2 & 2 \end{bmatrix}$$

```
> CM2:=subs(x=lambda[2],evalm(CM));
```

$$CM2 := \begin{bmatrix} -1 & 1 & -2 \\ 1 & -1 & 2 \\ -2 & 2 & -4 \end{bmatrix}$$

```
> ew1:=linsolve(CM1,vector(3,0));
```

$$ew1 := [\_t_1, -\_t_1, 2\,\_t_1]$$

```
> convert(ew1,matrix);
```

$$\begin{bmatrix} \_t_1 \\ \\ -\_t_1 \\ \\ 2\,\_t_1 \end{bmatrix}$$

```
> ew2:=convert(linsolve(CM2,vector(3,0)),matrix);
```

$$ew2 := \begin{bmatrix} \_t_1 - 2\,\_t_2 \\ \\ \_t_1 \\ \\ \_t_2 \end{bmatrix}$$

Maple Funktionen zur Eigenwert- und Eigenvektorberechnung

Berechnung der Eigenwerte und Eigenvektoren mit den Funktionen des Package linalg:

```
> charpoly(M,x);
```

$$-7 + 15\,x - 9\,x^2 + x^3$$

```
> ew:=eigenvals(M);
```

$$ew := 7, 1, 1$$

Da die Eigenwerte als Liste dargestellt werden, können wir mit dem op() Kommando oder über den aktuellen Index auf die einzelnen Werte zugreifen.

```
> op(1,[ew]);op(2,[ew]);
```

$$7$$

$$1$$

```
> ev:=eigenvects(M);
```

$$ev := [7, 1, \{[1, -1, 2]\}], [1, 2, \{[1, 1, 0], [-2, 0, 1]\}]$$

Das Ergebnis ist recht umfangreich und besteht aus einer Liste, wobei jedes Element dieser Liste wiederum eine Liste ist, die aus Eigenwert , Vielfachheit (Entartung) und einer Basismenge der Eigenvektoren besteht.

Zugriff auf einzelne Elemente:

ev[i][1] ist das erste Listenelement des i-ten Ausdrucks der Lösungsfolge

Soll auf den 1. Eigenvektor des i-ten Eigenwertes zugegriffen werden, so ist dies mit ev[i][3][1] möglich.

```
> ev[1][3][1];
```

$$[1, -1, 2]$$

```
> ev[1][1]; ev[1][2]; ev[2][1]; ev[2][2];
```

7

1

1

2

Mit der nachfolgenden Befehlen lassen sich die Eigenwerte in handlichen Variablen speichern.

```
> ev1='ev1'; ev2:='ev2'; ew1:='ew1';
```

$$ev1 = ev1$$

$$ev2 := ev2$$

$$ew1 := ew1$$

```
> seq(assign(evaln(ew||i),ev[i][1]),i=1..2);
> seq(assign(evaln(ev||i),ev[i][3]),i=1..2);
> seq([ew||i,eval(ev||i)],i=1..2);
```

$$[7, \{[1, -1, 2]\}], [1, \{[1, 1, 0], [-2, 0, 1]\}]$$

```
> ew1; ev1;  ew2; ev2;
```

7

{[1, -1, 2]}

1

{[1, 1, 0], [-2, 0, 1]}

```
>
```