ComputerMathematik - Einführung in Maple / Teil I

(einleitende Bemerkungen; Überblick)

I N H A L T T E I L I :

1. Hinweise zu Software und Konfiguration; Art der Verwendung

2. Dokumentation

3. LaTeX Export

4. Genauere Vorstellung des worksheet interface und der interaktiven Verwendung;
gleich mal einige wichtige Befehle; Variablen, Datentypen, einfache Prozeduren

5. Eine kurze Tour quer durch

6. Ein erstes Anwendungsbeispiel

7. Weiterer Aufbau der Vorlesung

ComputerMathematik - Einführung in Maple / Teil I

(einleitende Bemerkungen; Überblick)

Winfried Auzinger (SS 2011)

Institut für Analysis und Scientific Computing

```
> restart; # restart engine; clear workspace
```

1. Hinweise zu Software und Konfiguration; Art der Verwendung

-- **Maple** ist ein mächtiges, interaktives **Computeralgebra-System,**
eignet sich sowohl für symbolische als auch für numerische
Berechnungen; 2D und 3D Grafik; voll programmierbar.

Hersteller: Maplesoft, Waterloo, CA (www.maplesoft.com)

-- (Das wichtigste 'Konkurrenzprodukt': **Mathematica**
Hersteller: Wolfram Research (www.wolfram.com)

-- In dieser LVA geht es um die kompetente **Verwendung**,
insbesondere die **Programmierung** von Maple;
auf die zugrundeliegenden Algorithmen wird nur teilweise eingegangen
(soweit sinnvoll und für das Verständnis erforderlich).

-- Derzeitige Version auf lva.student ist **Maple 12**

Als Studentensoftware erhältlich: **Maple 14** (aktuell)
siehe www.sss.tuwien.ac.at/sss/map
Preis: EUR 6,90 (online); EUR 9,00 (DVD-ROM im LMZ)

Unterschiede zwischen Maple 12 und Maple 14 hier
nicht besonders relevant. Ich verwende hier **Maple 14.01**
unter Windows 7 Professional.

-- ***3 Kategorien von Maple-Befehlen***:

'Kernel' - die wichtigsten Funktionen,
intern als optimierter C-Code realisiert.

'Library' - Vordefinierte Befehle, als Maple-Prozeduren realisiert
(macht für die Verwendung keinen prinzipiellen Unterschied).

'Packages' - Viele zusätzliche Bibliotheken (z.B. **plots, LinearAlgebra, ...**);
sind bei Bedarf separat zu aktivieren.

-- ***Dateityp für Worksheets***: **.mw** (XML basiert)

-- ***Interaktives Interface***:

Aufruf auf lva.student mit

xmaple [***myworksheet*.mw**] &

... aktiviert Standard-Interface (Java-basiert);

per Menü: **Tools > Options > Interface > default format for new worksheet...** einstellbar ist:

'worksheet mode' ('klassisches' Interface; **HIER** verwendet)
oder

'document mode' (default; mit vielen Formatierungsmöglichkeiten
für fertige Dokumente, Präsentationen etc.)

Empfehlung: Für 'normales' Arbeiten (wenn es nicht um perfekte optische Präsentation geht)
verwendet man besten den **worksheet mode.**

'Document mode': ganz nett, wird aber hier nicht genauer besprochen.
Uns geht es in erster Linie nicht um Dokumentformatierung ,
sondern um **kompetenten Umgang** und **Programmierung.**

Auf Windows-Systemen gibt es auch noch das (alte) 'classic worksheet' Format
(verwendet kein Java, sondern generische Grafiklibraries; spart Ressourcen; Grafik O.K).

-- ***Zum Einstieg*:** Machen Sie mal selbst die 'Maple Tour' (**Help > Take a Tour of Maple**)
(zeigt insbesondere die Möglichkeiten zur Formatierung von Dokumenten).

-- ***Voreinstellungen*** (Empfehlung): im Menü: **Tools > Options > ...**
... **General**: *Generate a new engine for each document*
... **Display**: -- Input display: *Maple notation* ('klassisch', als unformatierter Text)
-- Output display: *2D-Math notation*
-- □ *Always insert new execution group after executing*
(erzeugt immer neuen Prompt > nach Kommandoausführung;
insbesondere für neu zu generierende Worksheets)
... **Interface**: Default format for new worksheet: *worksheet*
... **Precision** (für Gleitpunkt-Berechnungen): getrennt für interne Rechengenauigkeit
und Ausgabegenauigkeit einstellbar (z.B. 20, 10)
... etc.

-- **HIER** sehen Sie einen als **Text** formatierten Paragraphen (per Menü: **Insert > Text**)
dient für Dokumentation und Kommentare (Eingabe ähnlich wie in einer konventionellen Textverarbeitung,
mit font selection, Farben, etc.). Editieren ist jedoch zum Teil ein bisschen mühsam.

-- **Direkte interaktive** Ausführung von Befehlen (per Menü: **Insert > Maple Input;** **<Strg>j** bzw. **<Strg>k**)

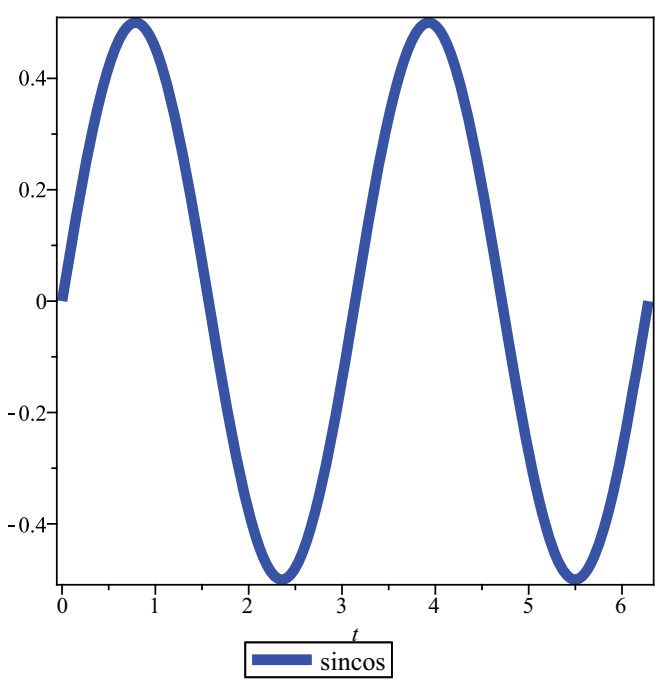
```
> sin(x+x);
```

$$\sin(2\,x)$$

```
> latex(sin(x+x)); # Ausgabe in LaTeX-Syntax
                   # (für Übernahme in LaTeX Dokument)
```
```
\sin \left( 2\,x \right)
```
```
> # Das ist ein Kommentar (unformatierter Text), wird bei
  Auswertung ignoriert
> int(sin(x),x); # berechne Stammfunktion von sin
```
$$-\cos(x)$$

oder Definition von **Prozeduren**:

```
> sincos := proc(x)
    sin(x)*cos(x) # der zuletzt berechnete Ausdruck
                  # wird als Wert zurückgegeben
  end proc;
```
$$sincos := \mathbf{proc}(x)\ \sin(x) * \cos(x)\ \mathbf{end\ proc}$$
```
> sincos(t);  # Aufruf der Prozedur mit x=t
```
$$0.4546487134$$
```
> t:=1.0:  # : statt ; unterdrückt Ausgabe
  sincos(t);
```
$$0.4546487134$$
```
> diff(sincos(z),z); # Verwendung der Prozedur (für
  Differentiation)
```
$$\cos(z)^2 - \sin(z)^2$$
```
> plot(sincos(t),t=0..Pi,axes=boxed,thickness=6,color=blue,
  legend="sincos");
  #   A C H T U N G:  t hat Wert von oben
```
```
Error, (in plot) unexpected option: 1.0 = 0 .. Pi
```
```
> t;
```
$$1.0000000000$$
```
> plot(sincos('t'),'t'=0..2*Pi,
       axes=boxed,thickness=6,color=blue,legend="sincos"); #
  Graph der Funktion
```

-- **ALSO:** Einfachste interaktive Verwendung von Maple: Eingabe über den Prompt

-- mehrzeilige Eingabe mittels <Strg><ENTER> (Cursor springt in nächste Zeile)

-- Abschluss / Ausführen mittels **;** oder **:** dann <ENTER>

-- Begleitende Texte am einfachsten als Kommentar (alles nach #)
oder als Text (per Menü: **Insert > Text**) bzw. <Strg>t

-- Viele Umformungen können auch über ein 'Kontextmenü' (rechte Maustaste) aktiviert werden;
auf solche Editier-Specials gehen wir aber im weiteren kaum ein
(ist auch eine Frage des persönlichen Arbeitsstils).

2. Dokumentation

-- Komplette Dokumentation kann im pdf-Format von

www.maplesoft.com/documentation_center

bezogen werden (Link auf CompMath Homepage; *kostenlose Registrierung*).
Empfehlung zum Download:

Maple 14 User Manual (18 MB, 440 Seiten)

... enthält auch einen Abschnitt 'Getting Started'.

Eine Referenz für diese VO ist der

Maple Introductory Programming Guide (**MIPG;** 23 MB, 390 Seiten**)**

auf den wir uns gelegentlich beziehen (als Referenz für Details). Weiters:

Maple 14 Quick Reference Card (für Windowx, Linux, Mac)

-- Eine kurze **Einführung in Maple** finden Sie ebenfalls
auf der **CompMath Homepage**, weiters eine **alphabetisch geordnete Liste der wichtigsten Kommandos und Funktionen.**

-- Es gibt sehr viele Bücher über Maple (aber teilweise veraltet).

-- Für die praktische Arbeit (**wichtig!**): Sehr gute

!!!! ==== O n l i n e - H i l f e ==== !!!!!

(mit vielen Beispielen), zu aktivieren per Menü: **Help...** oder direkt mit **?**

```
> ? sin
```

-- Im Web: Maplesoft Online Help System - zu finden unter

www.maplesoft.com/support/help

-- ***Empfehlung***: Spielen Sie die Worksheets aus den einzelnen Vorlesungen selbst durch (Verändern, Varianten ausprobieren), und verwenden Sie systematisch die Online-Hilfe. Maple mit den umfangreichen Zusatzpackages ist ein sehr offenes, umfangreiches System, wo man ständig Neues entdeckt und dazulernt.

Achtung: Syntax nicht durchgehend konsistent, viele verschiedene Datenstrukturen und Datentypen.

3. LaTeX Export

Worksheets können per Menü: **File > Export As... > LaTeX** in LaTeX-Code umgewandelt werden; für die Weiterverarbeitung in LaTeX auf lva.student sind die zugehörigen Style-Files mittels des bash-Kommandos

```
$ export TEXINPUTS=:/usr/local/maple12/etc
```

zu aktivieren (beachte Doppelpunkt).

4. Genauere Vorstellung des worksheet interface und der interaktiven Verwendung; gleich mal einige wichtige Befehle; Variablen, Datentypen, einfache Prozeduren

```
> Los geht's:
Error, missing operator or `;`
> # oops!
> # Los gehts:...
```

-- Die einfachste Art zu arbeiten: interaktiv (Eingabe / Ausgabe)

```
> x;
```

x

```
> x; # eine 'symbolische' Variable (kein Wert zugewiesen)
     # Typ nicht a priori definiert
     # explizite Typdeklaration normalerweise nicht
  erforderlich.
```

x

```
> x: # : statt ; unterdrückt Ausgabe
> whattype(x);
```

symbol

-- Wertzuweisung mit **:=**

```
> x := 1;
```

$x := 1$

-- '... ' unterdrückt Auswertung!

```
> 'x', x; # Eine Variable und der zugewiesene Wert
          # Anm.: , als Trennzeichen generiert eine Folge
          # (`sequence') von Objekten
```

$x, 1$

```
> x,y,z := 1,2,3; # simultane Wertzuweisung für eine Folge
```
$$x, y, z := 1, 2, 3$$

-- Rückgängig machen einer Wertzuweisung:

```
> x := 'x'; x;
```
$$x := x$$
$$x$$

```
> y := 1:
  unassign('y'); # wirkt wie y := 'y';
  y;
```
$$y$$

-- Mehrere Befehle in einer Zeile (**;** als Trennzeichen):

```
> s := sin(710); c := cos(710);
```
$$s := \sin(710)$$
$$c := \cos(710)$$

Übersichtlicher: Eingabe über mehrere Zeilen verteilt: (mit **<Strg><ENTER>**)

```
> s := sin(710);
  c := cos(710);
```
$$s := \sin(710)$$
$$c := \cos(710)$$

Dies verwendet man z.B. beim Editieren von Schleifen, etc., und Prozeduren:

```
> mysin := proc(x)
    sqrt(1-cos(x)^2)
  end proc;
```
$$mysin := \mathbf{proc}(x)\ \mathrm{sqrt}(1 - \cos(x)\,\hat{}\,2)\ \mathbf{end\ proc}$$

```
> mysin(Pi/2); # Aufruf der Prozedur
```
$$1$$

-- ACHTUNG: In Maple fehlt dzt. ein separater, externer Editor für die Eingabe von Prozeduren.

Daher: -- Kleine Prozeduren direkt in Worksheet eingeben
-- Längere Prozeduren ggfs. mit externem Editor erstellen (später)

Oder: **Insert > Code Edit Region** öffnet ein kleines Editierfenster, was die Eingabe von Code etwas erleichtert. Funktioniert wie ein einfacher Texteditor. Eingabe kann auch mittels cut/paste in das Worksheet übertragen werden.

```
mysin := proc(x)
```


$$mysin := \mathbf{proc}(x)\ \mathrm{sqrt}(1 - \cos(x)\,\hat{}\,2)\ \mathbf{end\ proc}$$

-- Anmerkung zu den [... Abschnitten (Text, bzw. Text mit Formeln):

Das sind sogenannte

execution groups **(Insert > Execution Group/...** **<Strg>j** bzw. **<Strg>k**).

Normalerweise erhält man den Maple-Prompt **>** zur Eingabe von Formeln bzw. Code; mit **Insert > Text (** **<Strg>t)** kann man auf textuelle Eingabe umschalten.

Kombinieren von Text und Formeln innerhalb einer execution group ist möglich **(Insert > Maple Input**, **<Strg>m)** -- *alles leider derzeit ein bisschen umständlich.* --

5. Eine kurze Tour quer durch

```
> restart:
```

> 5.1 Datentypen :

Es gibt diverse Typen von Maple-Objekten (auch eine Hierarchie); der genaue Typ eines Objektes ist aber eher nur für fortgeschrittenere Programmierung wichtig.

```
> 2, whattype(2);
```
$$2, \mathit{integer}$$

```
> 2.0, whattype(2.0);
```
$$2.0000000000, \mathit{float}$$

```
> 4/6, whattype(4/6);
```
$$\frac{2}{3}, \mathit{fraction}$$

```
> type(2,numeric);
```
$$\mathit{true}$$

Z.B. gibt es auch den Datentyp 'Gleichung' (und viele andere...)

```
> eq := 0.5 = 1/2;
  whattype(eq);
```
$$eq := 0.5000000000 = \frac{1}{2}$$
$$`=`$$

```
> evalb(eq); # Diese Gleichung ist richtig.
```
$$\mathit{true}$$

```
> eq := u+1=2; # eine andere Gleichung
```
$$eq := u + 1 = 2$$

```
> evalb(eq); # 'generisch' falsch
```

false

```
> solve(eq,u); # Auflösen nach der Variablen u
```

1

```
> u; # Achtung: Lösungswert wird nicht automatisch zugewiesen.
```

u

```
> u := solve(eq,u): u;
```

1

```
> eq;
```

$2 = 2$

> 5.2 Einfache algebraische Manipulationen :

```
> a+a;
```

$2\,a$

```
> sqrt(a); a^a;
```

$\sqrt{a}$

a^a

```
> a+1, 1.0-2;
```

$a + 1,\ -1.0000000000$

```
> (sqrt(a))^2, a-a;
```

$a,\ 0$

simplify ist ein 08/15-Befehl zum Vereinfachen von Ausdrücken
('einfach' ist jedoch nicht präzise definiert) :

```
> simplify(1-cos(y)^2);
```

$\sin(y)^2$

```
> q := 1+2*a+a^2;
```

$q := 1 + 2\,a + a^2$

```
> simplify(q);
```

$1 + 2\,a + a^2$

factor versucht zu faktorisieren :

```
> q1 := factor(q);
```

$q1 := (a + 1)^2$

expand ist die 'Umkehrung' von factor:

```
> expand(q1);
```

$1 + 2\,a + a^2$

> 5.3 Ein bisschen Analysis :

Differenzieren, Integrieren, Grenzwert...

```
> Diff(y^3,y);  # 'fauler' Befehl
```

$\frac{d}{dy}\,(y^3)$

```
> diff(y^3,y);  #  'fleissiger' Befehl
```

$3\,y^2$

```
> Diff(y^3,y) = diff(y^3,y);
```

$\frac{d}{dy}\,(y^3) = 3\,y^2$

```
> integral := int(y^4,y); # ohne Integrationskonstante
```

$integral := \frac{1}{5}\,y^5$

```
> integral+C;
```

$\frac{1}{5}\,y^5 + C$

```
> Limit((exp(x)-1)/x,x=0,left) = limit((exp(x)-1)/x,x=0,left);
```

$\lim_{x \to 0^-} \frac{e^x - 1}{x} = 1$

```
> 1/a, 1./0.;
```

$\frac{1}{a},\ Float(\infty)$

```
> infinity+infinity; # infinity = Symbol für unendlich
```

∞

> 5.4 Gleichungen und Gleichungssysteme :

```
> 1 = 2; # eine Gleichung
```

$1 = 2$

```
> evalb(%); # % ... das vorhergehende Ergebnis
```

false

```
> eq := (x+1=y);
```

$eq := x + 1 = y$

```
> evalb(%);
```

false

```
> solve(eq,x); # Auflösen der Gleichung nach x
```

$y - 1$

```
> solve([x+y=1,x-y=alpha],[x,y]); # 2 lineare Gleichungen,
                                  # alpha beliebiger Parameter
```

$$\left[\left[x = \frac{1}{2} + \frac{1}{2}\,\alpha, y = \frac{1}{2} - \frac{1}{2}\,\alpha\right]\right]$$

> 5.5 Einfachste Datenstrukturen: exprseq (Folge), set (Menge), list (Liste)

```
> a,b,c := 1,2,3; whattype(%); # eine Folge (sequence; Typ
  exprseq)
                                  # (Basis-Konstrukt)
```

$$a, b, c := 1, 2, 3$$

$$exprseq$$

```
> myexprseq := a,a,b;
```

$$myexprseq := a, a, b$$

```
> {myexprseq}; whattype(%); # eine MENGE
```

$$\{a, b\}$$

$$set$$

```
> [myexprseq]; whattype(%); # eine (geordnete) LISTE
```

$$[a, a, b]$$

$$list$$

Anmerkung: Das sind alles endliche Objekte (z.B. kann man mit **set** keine unendlichen Mengen 'speichern').

Die wichtigsten Datenstrukturen:

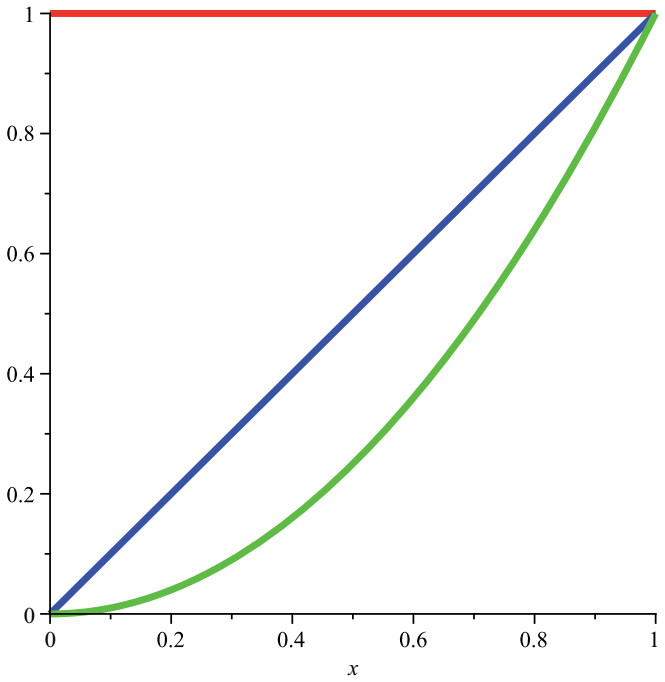
-- exprseq, set, list (siehe oben)

-- Array (ähnlich wie in Matlab), table, rtable (Details später)

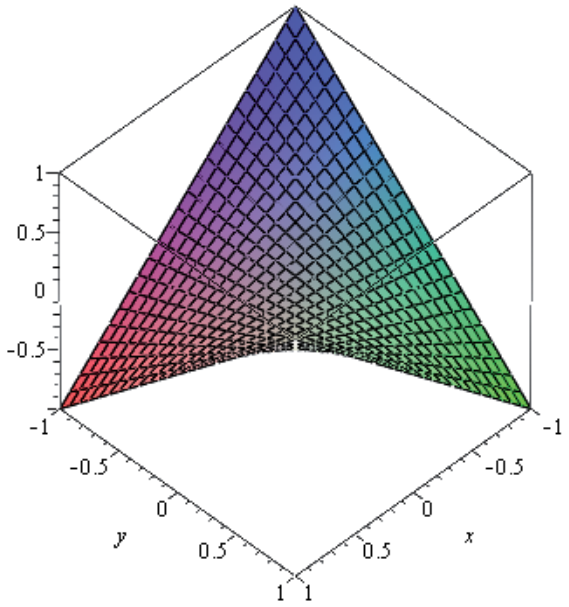
-- Vector, Matrix (für Lineare Algebra; Details später)

> 5.6 Grafik: viele Tools (teilweise in eigenem package 'plots')

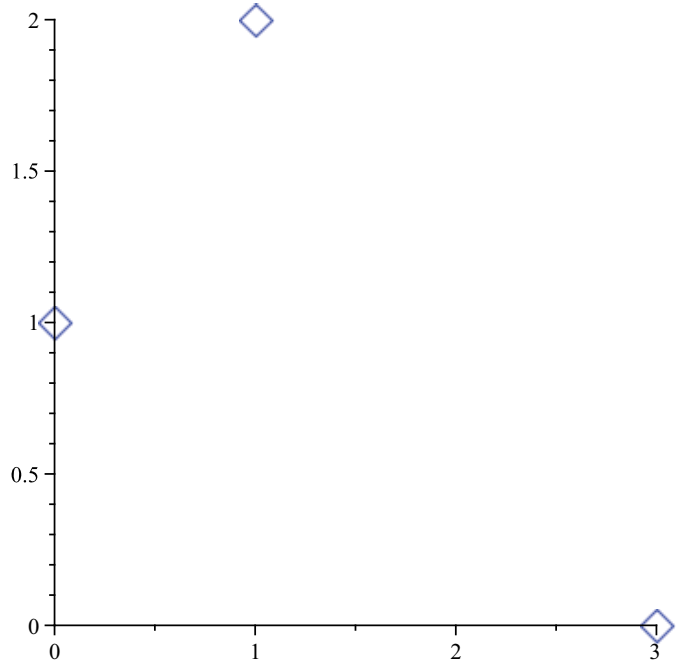
```
> plot([1,x,x^2],x=0..1, # .. steht für 'von ... bis'
  color=[red,blue,green],thickness=4,axes=normal);
```

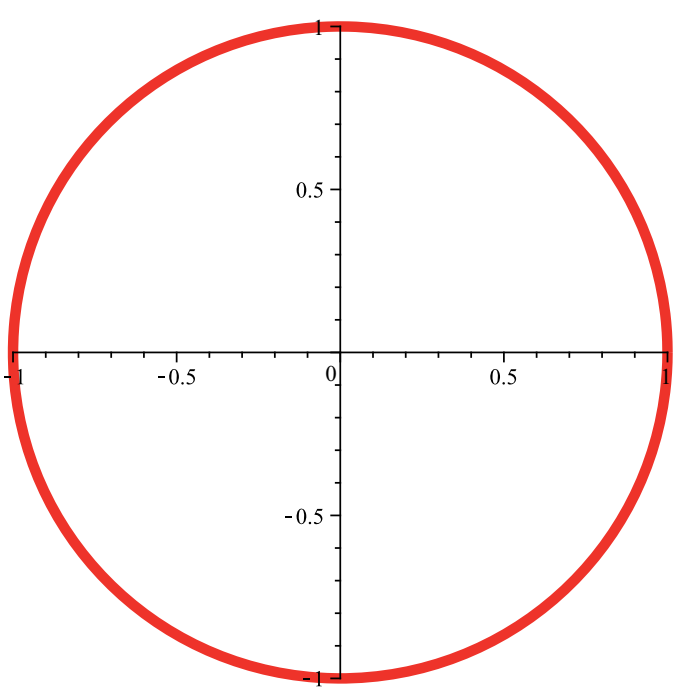

```
> plot3d(x*y,x=-1..1,y=-1..1,axes=boxed);
```




```
> plots[pointplot]([[0,1],[1,2],[3,0]],symbolsize=36,color=
  blue);
```


```
> plots[complexplot](exp(I*phi),phi=0..2*Pi,thickness=6);
```



> 5.7 Beispiel für ein wichtiges PACKAGE: LinearAlgebra

```
> with(LinearAlgebra); # aktiviert package
```

[&x, Add, Adjoint, BackwardSubstitute, BandMatrix, Basis, BezoutMatrix, BidiagonalForm, BilinearForm, CARE, CharacteristicMatrix, CharacteristicPolynomial, Column, ColumnDimension, ColumnOperation, ColumnSpace, CompanionMatrix, ConditionNumber, ConstantMatrix, ConstantVector, Copy, CreatePermutation, CrossProduct, DARE, DeleteColumn, DeleteRow, Determinant, Diagonal, DiagonalMatrix, Dimension, Dimensions, DotProduct, EigenConditionNumbers, Eigenvalues, Eigenvectors, Equal, ForwardSubstitute, FrobeniusForm, GaussianElimination, GenerateEquations, GenerateMatrix, Generic, GetResultDataType, GetResultShape, GivensRotationMatrix, GramSchmidt, HankelMatrix, HermiteForm, HermitianTranspose, HessenbergForm, HilbertMatrix, HouseholderMatrix, IdentityMatrix, IntersectionBasis, IsDefinite, IsOrthogonal, IsSimilar, IsUnitary,

JordanBlockMatrix, JordanForm, KroneckerProduct, LA_Main, LUDecomposition, LeastSquares, LinearSolve, LyapunovSolve, Map, Map2, MatrixAdd, MatrixExponential, MatrixFunction, MatrixInverse, MatrixMatrixMultiply, MatrixNorm, MatrixPower, MatrixScalarMultiply, MatrixVectorMultiply, MinimalPolynomial, Minor, Modular, Multiply, NoUserValue, Norm, Normalize, NullSpace, OuterProductMatrix, Permanent, Pivot, PopovForm, QRDecomposition, RandomMatrix, RandomVector, Rank, RationalCanonicalForm, ReducedRowEchelonForm, Row, RowDimension, RowOperation, RowSpace, ScalarMatrix, ScalarMultiply, ScalarVector, SchurForm, SingularValues, SmithForm, StronglyConnectedBlocks, SubMatrix, SubVector, SumBasis, SylvesterMatrix, SylvesterSolve, ToeplitzMatrix, Trace, Transpose, TridiagonalForm, UnitVector, VandermondeMatrix, VectorAdd, VectorAngle, VectorMatrixMultiply, VectorNorm, VectorScalarMultiply, ZeroMatrix, ZeroVector, Zip]

```
> A := Matrix([[3/2,2],
               [3  ,4]]); # eine Matrix
```

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

```
> b := Vector([alpha,beta]);
```

$$b := \begin{bmatrix} \alpha \\ \beta \end{bmatrix}$$

```
> x := A^(-1).b;
```

Error, (in rtable/Power) singular matrix

> 5.8 Funktionen :

```
> x -> x^3; # eine Funktion in 'arrow'-Notation
```

$$x \rightarrow x^3$$

```
> (x -> x^3)(2);
```

$$8$$

Wir geben der Funktion einen Namen:

```
> myfun := x->x^3; myfun(2);
```

$$myfun := x \rightarrow x^3$$

$$8$$

```
> f := (u,v) -> u^3+v; f(x,y); # 2 Variablen
```

$$f := (u, v) \rightarrow u^3 + v$$

$$x^3 + y$$

Anmerkung: Variablennamen in der Funktionsdefinition sind willkürlich wählbar (formale Parameter).

> 5.9 Grundlegende Programmierkonstrukte (Schleifen, Verzweigungen) :

```
> for i from 1 to 3 do  # Schleife
    i
  end do;
```

$$1$$
$$2$$
$$3$$

```
> x:=1;
```

$$x := 1$$

```
> if x=1 then # if-konstrukt;
     "eins" # (Beispiel für eine Zeichenkette)
  else
     "nicht eins"
  end if;
```

$$\text{"eins"}$$

Oft auch: 'implizite' Schleifenkonstrukte

```
> a := 'a';
```

$$a := a$$

```
> p := z -> add(a[k]*z^k,k=1..5); # eine Polynomfunktion
```

$$p := z \rightarrow add\left(a_k z^k, k = 1..5\right)$$

```
> p(8);
```

$$8\,a_1 + 64\,a_2 + 512\,a_3 + 4096\,a_4 + 32768\,a_5$$

Beachte a[1] usw. sind hier normale Variablen, aber *indiziert*

```
> a;
```

$$a$$

```
> a[1]; whattype(%);
```

$$a_1$$

$$indexed$$

6. Ein erstes Anwendungsbeispiel

Herleitung einer '**Quadraturformel**', die Polynome vom Grad <= n exakt integriert

über dem Intervall [0,1]; 'Abtastung' des Integranden an den n+1 Punkten 0,h,2h,...,1 in [0,1] mit Abstand h = 1/n

```
> restart;
> n := 3; # konkrete Wahl für n
          # Code funktioniert für beliebige n
          # und für beliebige Wahl von Knoten
```

$$n := 3$$

```
> h := 1/n; # Knotenabstand (äquidistant)
```

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

```
> for i from 0 to n do
    node[i]:=i*h
  end do; # die Knoten
```

$$node_0 := 0$$
$$node_1 := \frac{1}{3}$$
$$node_2 := \frac{2}{3}$$
$$node_3 := 1$$

```
> unassign(alpha); # stelle sicher, dass die alpha[i]
  undefiniert sind
```

Ansatz für die Quadraturformel Q = Q(f) :
(Beachte: Q ist eine Funktion, die auf eine Funktion f wirkten soll - ein sogenanntes 'Funktional')

```
> Q := f -> add(alpha[i]*f(node[i]),i=0..n); # die alpha[i]
  sind zu bestimmen
```

$$Q := f \rightarrow add\left(\alpha_i f\left(node_i\right), i = 0..n\right)$$

Wir stellen jetzt n+1 Gleichungen für die alpha[i] auf,
indem wir fordern, dass f(x)=1, f(x)=x, ... , f(x)=x^n exakt integriert wird.

```
> for k from 0 to n do
      eq[k] := Q(x->x^k) = int(x^k,'x'=0..1)
  end do;
```

$$eq_0 := \alpha_0 + \alpha_1 + \alpha_2 + \alpha_3 = 1$$
$$eq_1 := \frac{1}{3}\,\alpha_1 + \frac{2}{3}\,\alpha_2 + \alpha_3 = \frac{1}{2}$$
$$eq_2 := \frac{1}{9}\,\alpha_1 + \frac{4}{9}\,\alpha_2 + \alpha_3 = \frac{1}{3}$$
$$eq_3 := \frac{1}{27}\,\alpha_1 + \frac{8}{27}\,\alpha_2 + \alpha_3 = \frac{1}{4}$$

Lösen dieser Gleichungen:

```
> solve({seq(eq[k],k=0..n)},{seq(alpha[i],i=0..n)});
```

$$\left\{\alpha_0 = \frac{1}{8}, \alpha_1 = \frac{3}{8}, \alpha_2 = \frac{3}{8}, \alpha_3 = \frac{1}{8}\right\}$$

```
> map(assign,%); # Zuweisung der Lösungswerte an die Variablen
```

$$\{\}$$

```
> seq(alpha[i],i=0..n);
```

$$\frac{1}{8}, \frac{3}{8}, \frac{3}{8}, \frac{1}{8}$$

Test: Wähle jetzt ein beliebiges Polynom vom Grad n

```
> p := x->add(c[k]*x^k,k=0..n);
```

$$p := x \rightarrow add\left(c_k x^k, k = 0..n\right)$$

```
> int(p(x),x=0..1);
```

$$c_0 + \frac{1}{2} c_1 + \frac{1}{3} c_2 + \frac{1}{4} c_3$$

```
> Q(p);
```

$$c_0 + \frac{1}{2} c_1 + \frac{1}{3} c_2 + \frac{1}{4} c_3$$

```
> %-%%; # passt
```

$$0$$

Eine derartige Quadraturformel liefert auch für allgemeinere Funktionen
eine gute Approximation für das Integral:

```
> Integral := int(sin*exp,0..1); # Kurzschreibweise
                                 # (kein expliziter
  Variablenname)
```

$$Integral := \frac{1}{2} - \frac{1}{2}\cos(1)\,e + \frac{1}{2}\sin(1)\,e$$

```
> evalf(%); # Gleitpunkt-Auswertung
```

$$0.9093306736$$

```
> Quadratur := Q(sin*exp);
```

$$Quadratur := \frac{3}{8}\sin\left(\frac{1}{3}\right)e^{\frac{1}{3}} + \frac{3}{8}\sin\left(\frac{2}{3}\right)e^{\frac{2}{3}} + \frac{1}{8}\sin(1)\,e$$

```
> evalf(%);
```

$$0.9088157483$$

```
> evalf(Integral-Quadratur); # der Quadraturfehler
```

$$0.0005149254$$

Das ist z.B. viel genauer als eine Riemann-Summe
über die kleinen Teilintervalle:

```
> R := f->h*add(f(node[i]+h/2),i=0..n-1);
```

$$R := f \rightarrow h\, add\left(f\left(node_i + \frac{1}{2}\,h\right), i = 0..n-1\right)$$

```
> evalf(R(sin*exp));
```

$$0.8965171004$$

Obige Quadraturformel beruht auf **Interpolation**:
Der Integrand wird an den Knoten durch ein Polynom vom Grad n interpoliert,
und das Polynom wird integriert.

Als Übung stellen wir noch das Interpolationspolynom
explizit auf und plotten den Fehlerverlauf.
Aber diesmal gleich alles in Gleitpunktarithmetik.

Ansatz für Interpolationspolynom:

```
> unassign('c');
  p := x->add(c[k]*x^k,k=0..n);
```

$$p := x \rightarrow add\left(c_k x^k, k = 0..n\right)$$

Die zu interpolierende Funktion:

```
> f := sin*exp;
```

$$f := \sin \exp$$

Die Interpolationsbedingungen (Gleichungen):

```
> for i from 0 to n do
    eq[i] := evalf(p(node[i])-f(node[i]))=0
  end do;
```

$$eq_0 := c_0 = 0$$

$$eq_1 := c_0 + 0.3333333333\,c_1 + 0.1111111111\,c_2 + 0.0370370370\,c_3 - 0.4566369843 = 0$$

$$eq_2 := c_0 + 0.6666666667\,c_1 + 0.4444444444\,c_2 + 0.2962962963\,c_3 - 1.2044199154 = 0$$

$$eq_3 := c_0 + c_1 + c_2 + c_3 - 2.2873552872 = 0$$

```
> fsolve({seq(eq[i],i=0..n)},{seq(c[k],k=0..n)}); # numerischer
  solver
```

$$\{c_0 = -0.0000000000, c_1 = 0.9771985263, c_2 = 1.1121275388, c_3 = 0.1980292221\}$$

```
> map(assign,%);
```

$$\{\}$$

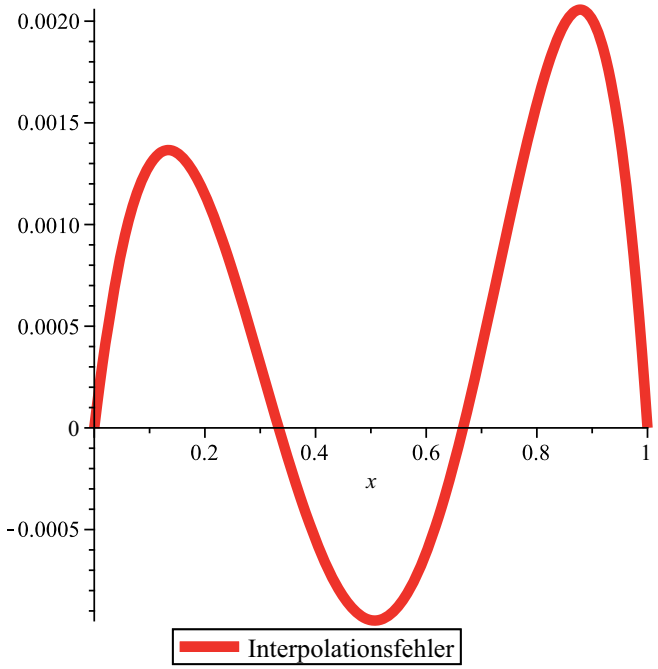
Das Polynom:

```
> evalf(p(x));
```

$$0.9771985263\,x + 1.1121275388\,x^2 + 0.1980292221\,x^3$$

```
> plot([f(x)-p(x)],x=0..1,thickness=6,legend=
```

```
"Interpolationsfehler");
```



7. Weiterer Aufbau der Vorlesung

-- Schwerpunkt auf **Datentypen**, **Datenstrukturen** und **Programmierung**

-- Befehle werden systematisch (manchmal auch 'en passant' erklärt); Beispiele

-- Eingehen auf mathematischen Hintergrund, soweit sinnvoll und notwendig

-- Gelegentlich Verweis auf 'Maple Introductory Programming Guide' (**MIPG**)

-- **Satz von Auzinger:** Computeralgebra ist extrem cool und ein sehr wertvolles Werkzeug für den Mathematiker.

-- Korollar 1: Vertraue dem System nicht blind. Man muss wissen, was man in etwa

zu erwarten hat. Vertrauen ist gut, Kontrolle ist besser.

-- Korollar 2: Ein Computeralgebra-System ist auch ein 'mathematisches Experimentierlabor' . Kompetenter Umgang bedeutet jedoch, dass man ein Ziel vor Augen hat und nicht nur unsystematisch 'herumprobiert'.

-- Folgerung: Die Kompetenz muss auch vom Benützer kommen - schließlich geht es um ComputerMATHEMATIK.

-- Wir werden auch auf typische 'Fallen' und seltsame Features (bugs?) eingehen.

=== Ende Maple / Teil I ===