

ComputerMathematik - Einführung in Maple / Teil I (einleitende Bemerkungen; Überblick)

Winfried Auzinger (SS 2010)

Institut für Analysis und Scientific Computing

Hinweis:

Die während der VO entstehende fertige Version dieses Vortrags wird nach der VO auf der CompMath - Homepage als **.mw** und **.pdf** zum Download bereitgestellt.

```
[> 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.
- 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 12.0** (Hersteller: Maplesoft, Waterloo, CA)

Als Studentensoftware erhältlich am LMZ: **Maple 13.02** (aktuell) siehe <http://www.sss.tuwien.ac.at/sss/map/>; Preis: EUR 9,-

Unterschiede zwischen Maple 12 und Maple 13 hier nicht besonders relevant. **Ich** verwende hier **Maple 13.02** unter Windows XP Pro.

-- *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

```
xmapple [myworksheet.mw]
```

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

per **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 um **kompetenten Umgang** und **Programmierung**.

Z.B. 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/Make a Tour of Maple**)
(zeigt insbesondere die Möglichkeiten zur Formatierung von Dokumenten).

-- **Voreinstellungen** (Empfehlung): **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 (**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 Ausführung von Befehlen (**Insert/Maple Input; <CTRL>j** bzw. **<CTRL>k**)

```
> 1+2
```

```
Warning, inserted missing semicolon at end of statement
```

```
3
```

```
> 1+2;
```

```
3
```

```
> # Ein Kommentar (unformatierter Text)
```

```
> Int(sin(x),x)=int(sin(x),x); # berechne Stammfunktion von sin
```

```
∫ sin(x) dx = -cos(x)
```

oder Definition von **Prozeduren**:

```
> sincos:=proc(x)
```

```
sin(x)*cos(x) # der zuletzt berechnete Ausdruck
```

```

                # wird als Wert zurückgegeben
end proc;
                sincos := proc(x) sin(x) * cos(x) end proc

> sincos(t); # Aufruf
                sin(t) cos(t)

> t:=1.0: sincos(t); # : statt ; unterdrückt Ausgabe
                0.4546487134

> diff(sincos(z),z); # Verwendung der Prozedur
                cos(z)2 - sin(z)2

> evalf(Pi);
                3.1415926536

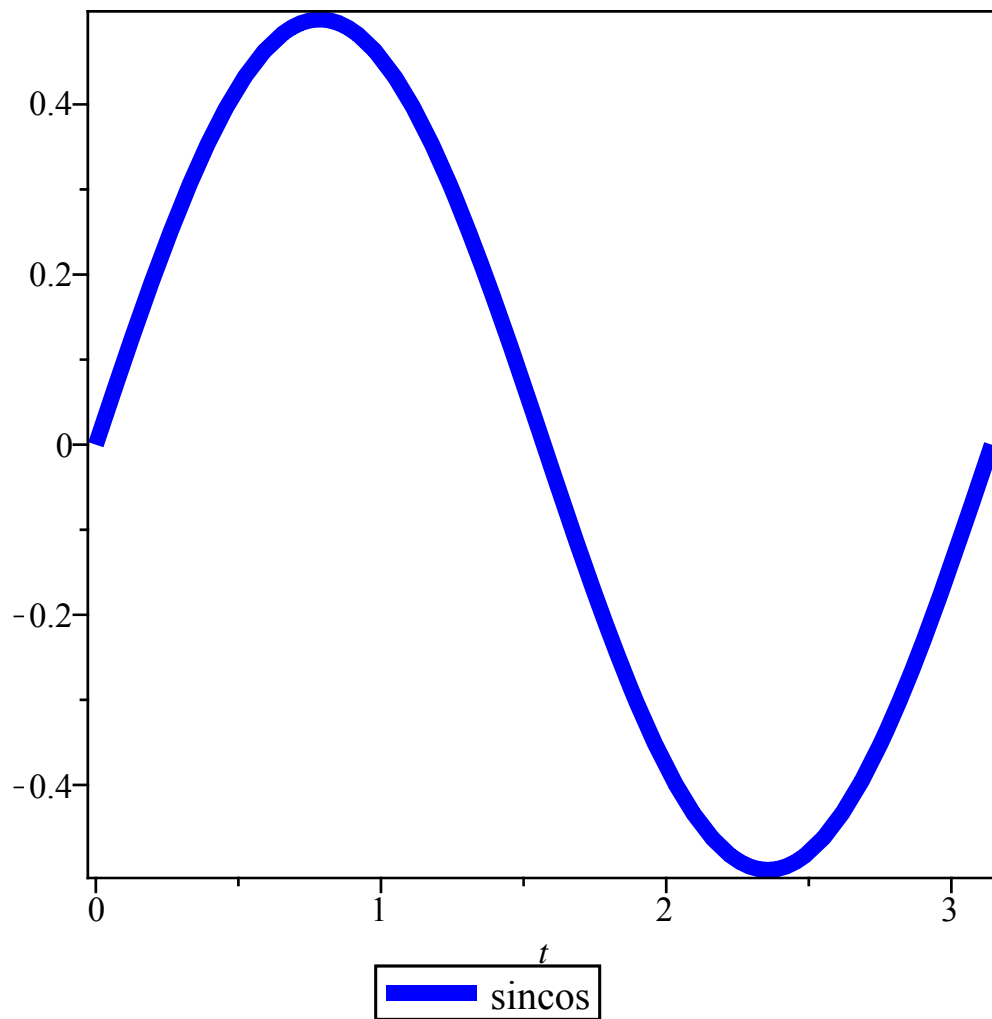
> pi, Pi;
                π, π

> plot(sincos(t),t=0..Pi,axes=boxed,thickness=6,color=blue,legend=
"sincos");
# ACHTUNG: t hat Wert von oben
Error, (in plot) unexpected option: 1.0 = 0 .. Pi

> t;
                1.0000000000

> plot(sincos('t'),'t'=0..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 `<CTRL><ENTER>`

-- Abschluss/Ausführen mittels `;` und `<ENTER>`

-- Begleitende Texte am einfachsten als Kommentar (alles nach `#`) oder Text wie hier; ist nicht 'optisch perfekt', aber am einfachsten und schnellsten zu editieren.

-- 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

http://www.maplesoft.com/documentation_center/

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

Maple 13 User Manual (43.7 MB, 540 Seiten)

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

Eine Referenz für diese VO ist der

Maple Introductory Programming Guide (MIPG; 19.9 MB, 398 Seiten)

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

Maple 13 Quick Reference Card (für Windows, Linus, Mac)

-- Eine (nicht ganz aktuelle, aber gut lesbare) **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 mit **Help...** oder direkt mit ?

[> ? **sin**

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

<http://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 mit **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
```

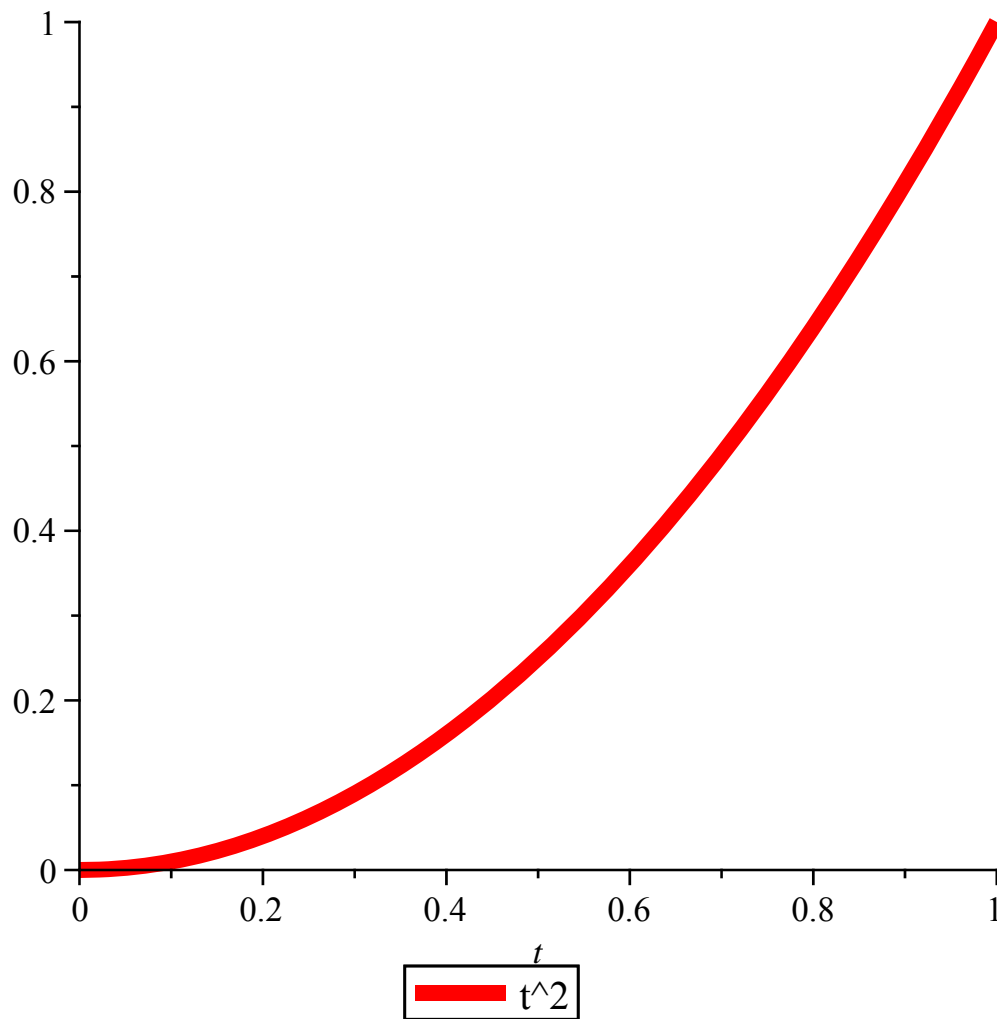
zu aktivieren (beachte Doppelpunkt).

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:...
> x;
                                     x
> x; # eine 'symbolische' Variable (ohne zugewiesenen Wert)
    # Typ nicht a priori definiert
    # explizite Typdeklaration normalerweise nicht erforderlich
                                     x
> x: # : statt ; unterdrückt Ausgabe
> whattype(x);
                                     symbol
> x:=1; # Wertzuweisung mit :=
                                     x:=1
> 'x', x; # Ein Variablenname und der zugewiesene Wert
    # , als Trennzeichen generiert eine 'sequence' von Objekten
                                     x, 1
> x,y,z := 1,2,3; # simultane Wertzuweisung für eine sequence
                                     x,y,z:=1,2,3
> x:='x'; x; # Wertzuweisung rückgängig gemacht
                                     x:=x
                                     x
> unassign('y'); y; # geht auch mit unassign()
                                     y
> t:=1;
                                     t:=1
> plot('t^2','t'=0..1,thickness=6,legend="t^2");

```



```
> whattype('t'), whattype(t);
```

```
symbol, integer
```

```
> s:=sin(710);
```

```
s := sin(710)
```

```
> s:=sin(710); c:=cos(710); # mehrere Befehle in einer Zeile OK ( ; als  
Trennzeichen)
```

```
s := sin(710)
```

```
c := cos(710)
```

Eingabe über mehrere Zeilen: (mit **<CTRL><ENTER>**)

```
> x:=
```

```
1;
```

```
x := 1
```

Dies benötigt man z.B. zum Editieren von Prozeduren:

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

```
  sqrt(1-cos(x)^2)
```

```
end proc;
```

```
mysin := proc(x) sqrt(1 - cos(x)^2) end proc
```

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

```
  sqrt(1-cos(x)^2)
```

```
end proc:
```

```
> mysin(Pi/2);
```

1

```
> mysin(pi/2);
```

$$\sqrt{1 - \cos\left(\frac{1}{2} \pi\right)^2}$$

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)

NEU in Maple 13: **Insert/Code Edit Region** öffnet ein kleines Editierfenster, was die Eingabe von Code etwas erleichtert. Funktioniert wie ein einfacher Texteditor; Eingabe kann mittels cut/paste in das Worksheet übertragen werden.

Noch eine Bemerkung zu diesen [... Abschnitten (Text, bzw. Text mit Formeln):
Das sind sogenannte

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

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

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

5. Eine kurze Tour quer durch

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, integer

```
> 2.0,whattype(2.0);
```

2.0000000000, float

```
> 4/6,whattype(4/6);
```

$\frac{2}{3}$, fraction

```
> type(2,numeric);
```

true

```
> type(2,fraction);
```

false

```
> eq := 0.5=1/2; eq;whattype(%); # eq ist eine GLEICHUNG (verwende =)
```

eq := 0.5000000000 = $\frac{1}{2}$

0.5000000000 = $\frac{1}{2}$

'='


```

> evalb(eq); # Diese Gleichung ist richtig.
           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
=
* Einfache algebraische Manipulationen:
=
> a+a;
           2 a
=
> sqrt(a); a^a;
           sqrt(a)
           a^a
=
> a+1, 1.-2;
           a + 1, -1.0000000000
=
> (sqrt(a))^2, a-a;
           a, 0
=
> simplify(1-cos(y)^2); # simplify für Vereinfachung von Ausdrücken
                       # Achtung: 'einfach' ist nicht eindeutig definiert
                       sin(y)^2
=
> q:=1+2*a+a^2;
           q := 1 + 2 a + a^2
=
> simplify(q);
           1 + 2 a + a^2
=
> factor(q); # Faktorisierung
           (a + 1)^2
=
> expand(%); # expandieren (z.B. ausmultiplizieren)
           1 + 2 a + a^2
=
* Ein bisschen Analysis:
=
> Diff(y^3,y);

```

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

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

$$3y^2$$

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

$$integral:=y^3$$

```
> integral+C;
```

$$y^3 + C$$

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

$$x:=x$$

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

$$1$$

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

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

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

$$\infty$$

```
* Gleichungen:
```

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

$$1=2$$

```
> evalb(%); # % ... 'Ditto'-operator
```

false

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

$$eq := x + 1 = y$$

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

false

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

$$-1 + y$$

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

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

```
* Einfachste Datenstrukturen: exprseq, set, list
```

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

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

exprseq

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

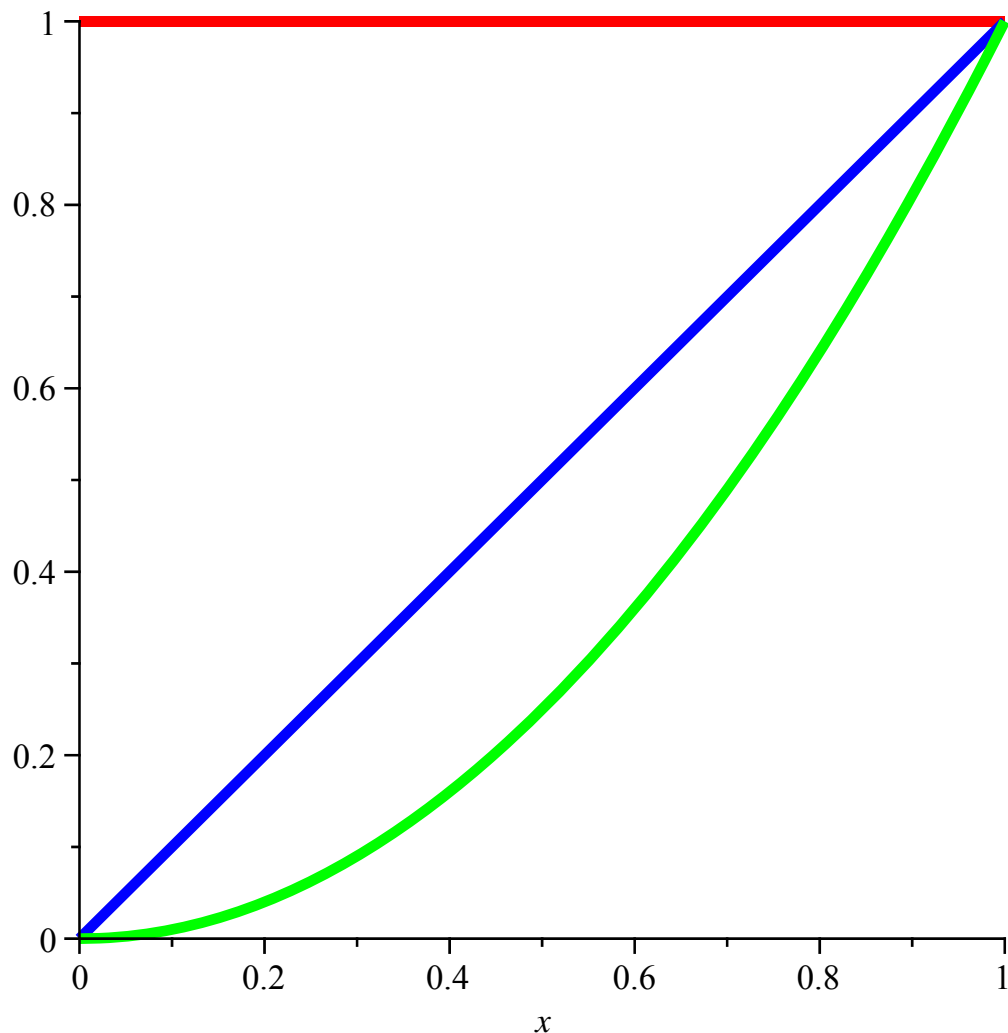
$$myexprseq := 1, 1, 2$$

```
> {myexprseq}; whattype(%); # eine MENGE
      {1, 2}
      set
```

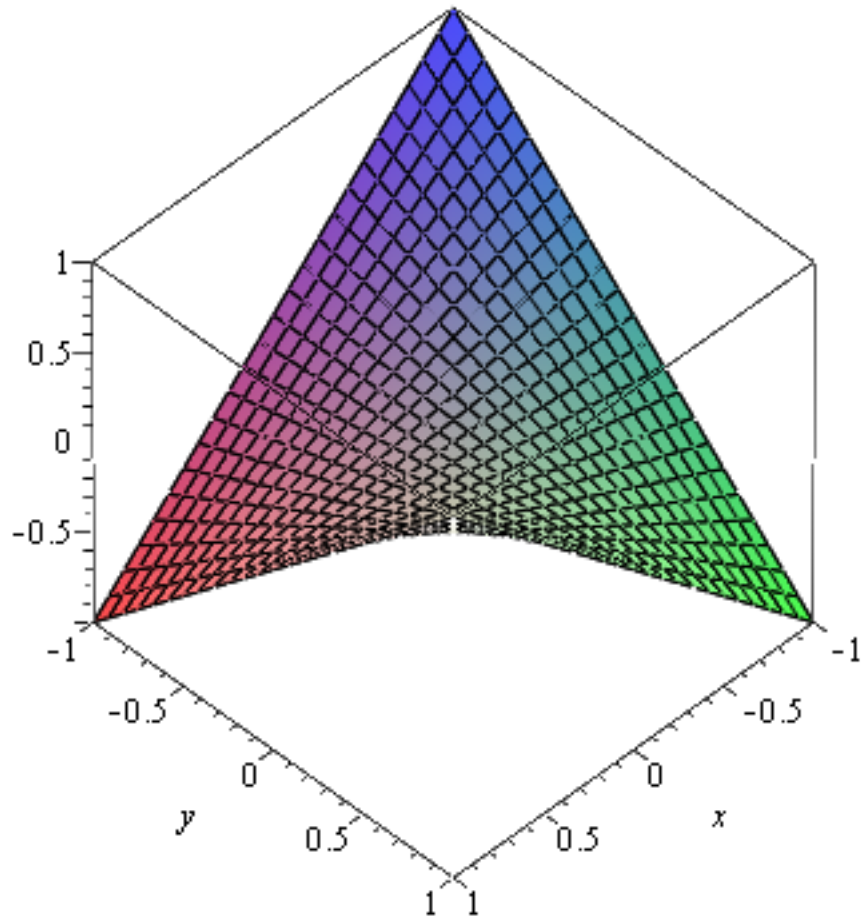
```
> [myexprseq]; whattype(%); # eine (geordnete) LISTE
      [1, 1, 2]
      list
```

* Grafik: viele Tools (teilweise in plots-package)

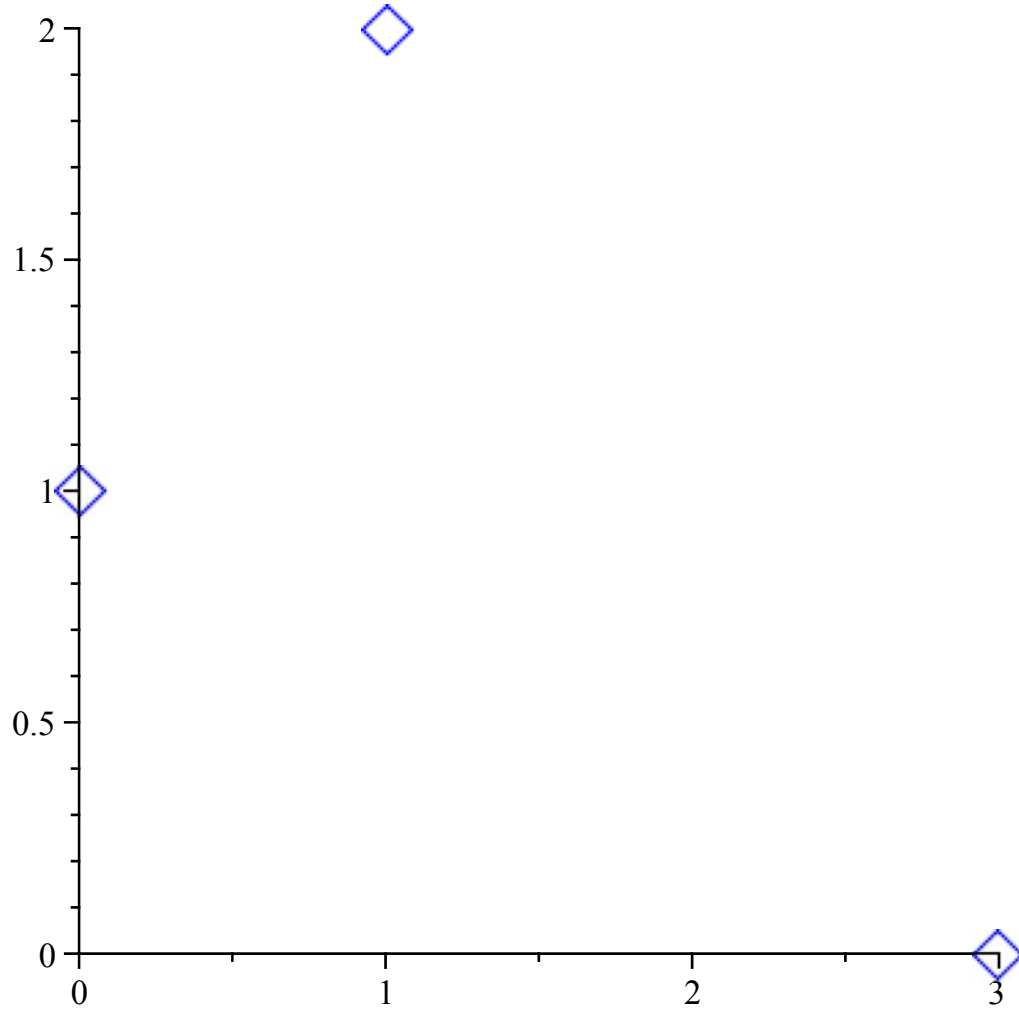
```
> plot([1,x,x^2],x=0..1,
      color=[red,blue,green],thickness=4,axes=normal);
```



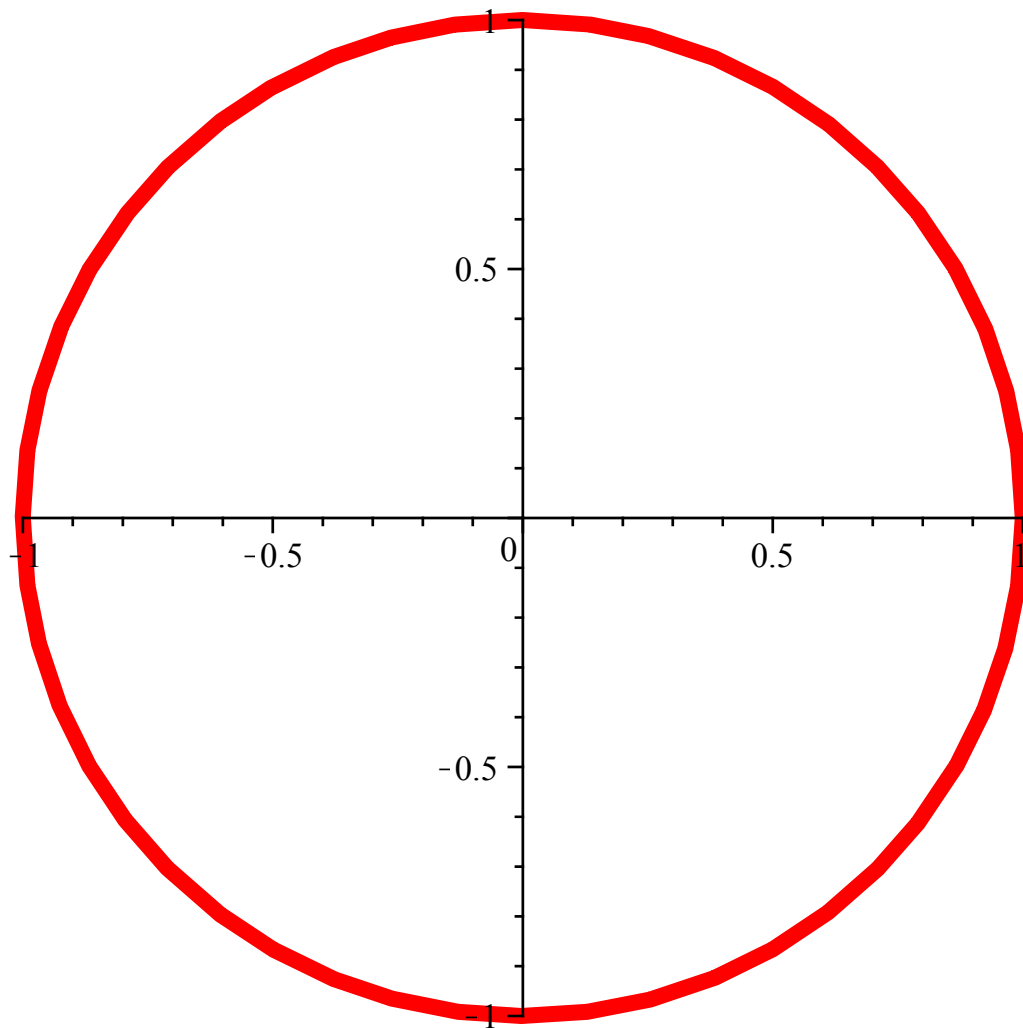
```
> plot3d(x*y,x=-1..1,y=-1..1,axes=boxed);
```



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



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



* **Datenstrukturen/typen:**

- exprseq, set, list (siehe oben)
- Array, table, rtable (Details später)
- Vector, Matrix (für Lineare Algebra)

* **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]]);
```

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

```
> b:=Vector([donald,duck]);
```

$$b := \begin{bmatrix} donald \\ duck \end{bmatrix}$$

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

Error, (in rtable/Power) singular matrix

* Funktionen und Prozeduren:

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

$$x \rightarrow x^3$$

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

8

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

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

8

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

$$f := (x, y) \rightarrow x^3 + y$$

8 + mickimaus

* Grundlegende Programmierkonstrukte:

```
> for i from 1 to 5 do i end do; # Schleife
```

1

2

3

4

5

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

x := 1

```
> if x=1 then # if-konstrukt;
    "eins"
else
    "nicht eins"
end if;

"eins"
```

```
> whattype("eins");

string
```

Oft auch: 'implizite' Schleifenkonstrukte

```
> a:='a';

a := a
```

```
> p:=z->sum(a[k]*z^k,k=1..5); # ein Polynom
```

$$p := z \rightarrow \sum_{k=1}^5 a_k z^k$$

```
> p(8);

8 a1 + 64 a2 + 512 a3 + 4096 a4 + 32768 a5
```

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

```
> a;

a
```

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

a1
indexed
```

```
> # usw. usw.
```

6. Ein Anwendungsbeispiel

Herleitung einer 'Quadraturformel', die Polynome vom Grad $\leq n$ exakt integriert über dem Intervall $[0,1]$; 'Abtastung' des Integranden an den $n+1$ Punkten $0, h, 2h, \dots, 1$ 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
node0 := 0
node1 :=  $\frac{1}{3}$ 
```


$$node_2 := \frac{2}{3}$$

$$node_3 := 1$$

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

Ansatz für die Quadraturformel:

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

$$Q := f \rightarrow \text{add}(\alpha_i f(\text{node}_i), i=0..n)$$

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);
```

$$p := x \rightarrow \text{add}(c_k x^k, k=0..n)$$

```
> 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
```

$$0$$

ANMERKUNGEN:

- Die $x[i]$, $\alpha[i]$ sind indizierte Variablen (steht für x_i , y_i)
- `seq(...)` ist Konstruktor für `exprseq` mittels impliziter Schleife
- `{ ... }` macht daraus eine *Menge* (Menge oder Liste wird erwartet vom `solve`-Befehl)
- Oben haben wir ein lineares Gleichungssystem mittels `solve` gelöst;
für große Gleichungssysteme verwendet man besser Vektor- und Matrixsyntax
aus dem Paket `LinearAlgebra`.
- 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);
                                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 -> h add( f( node_i + 1/2 h ), i = 0 .. n - 1 )
```

```
> 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 -> add( c_k x^k, k = 0 .. n )
```

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;

eq0 := c0 = 0
eq1 := c0 + 0.3333333333 c1 + 0.1111111111 c2 + 0.0370370370 c3 - 0.4566369843 = 0
eq2 := c0 + 0.6666666667 c1 + 0.4444444444 c2 + 0.2962962963 c3 - 1.2044199154 = 0
eq3 := c0 + c1 + c2 + c3 - 2.2873552872 = 0

> fsolve({seq(eq[i],i=0..n)},{seq(c[k],k=0..n)}); # numerischer solver
{c0 = -0.0000000000, c1 = 0.9771985263, c2 = 1.1121275388, c3 = 0.1980292221}

> map(assign,%);

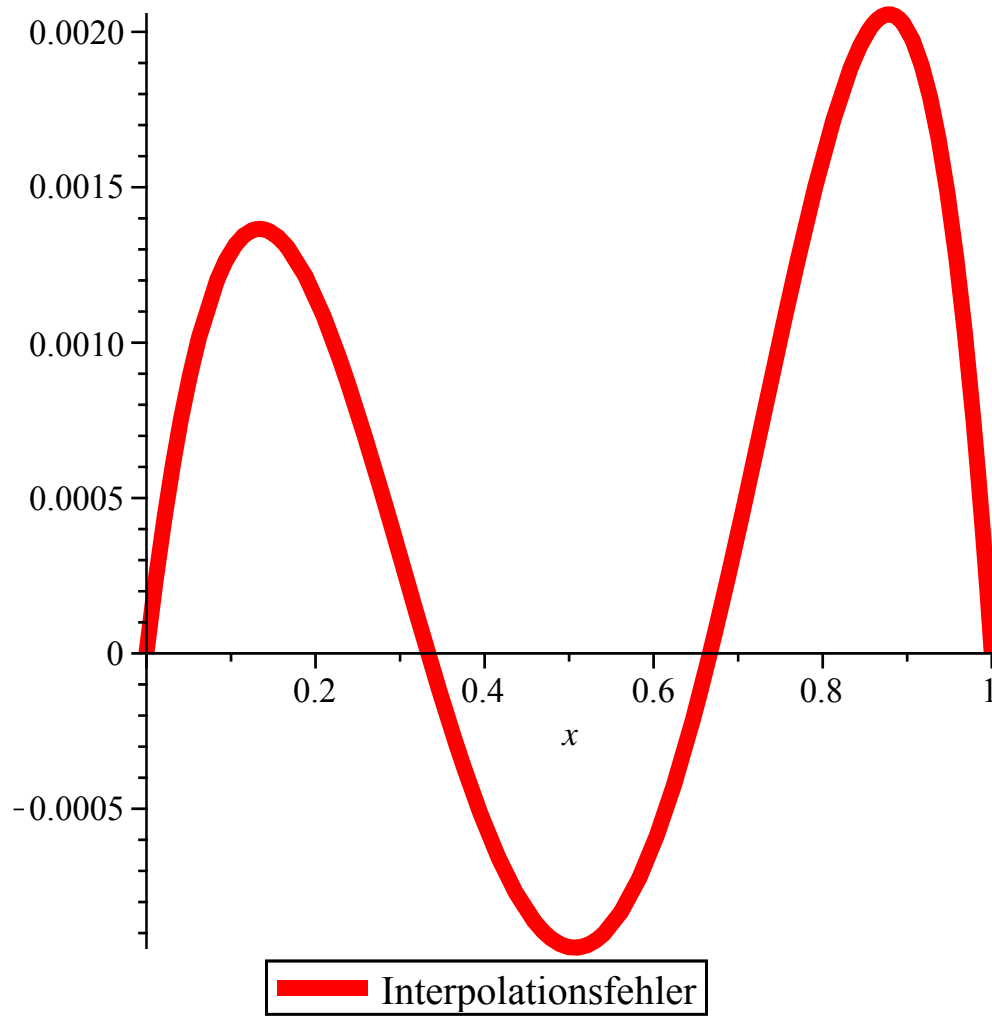
```

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 oder '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 unabdingbares 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 vom Benutzer kommen - schließlich geht es um ComputerMATHEMATIK.**
 - Wir werden auch auf typische 'Fallen' und seltsame Features (bugs?) eingehen.
- === *Ende Maple / Teil I* ===