Technische Universität Ilmenau
Fakultät für Mathematik
und Naturwissenschaften
Institut für Mathematik

Postfach 10 0565
98684 Ilmenau
Germany
Tel.: 03677/693267
Fax: 03677/693272
Telex: 33 84 23 tuil d.
email: werner.neundorf@mathematik.tu-ilmenau.de
brigitte.walther@mathematik.tu-ilmenau.de

Preprint No. M 12/00

Grafik, Animation und Ausgabeformate in Maple V Release 5

Werner Neundorf, Brigitte Walther

Juli 2000

[‡]MSC (1991): 68-01, 68-04, 68Q40, 68Q25

Abstract

This is a tutorial on programming with Maple graphics version V Release 5. It is a continuation of the preprint [12] and based on scripts and exercises in the course of mathematics and numerical mathematics for students of the Ilmenau Technical University. The aim is to show how you can do in Maple calculations for prepering plots and write graphic plot commands for plotting functions, for doing image presentation, animation, and export of graphics in different formats of the output. It is assumed that the reader is familiar with using Maple interactively and plot commands.

Preface

It is an introduction to special aspects of programming with Maple graphics with examples and exercises.
This gives some insight and examples for making plots, the export and storage of graphics, for animation and last not least for producing animated transparent or non-transparent image files. There are added tutorial aspects, too.

A few words to those who are familiar with other programming languages [1].

- Maple is a procedural programming language. It also includes a number of functional programming constructs. If you have written programs in Basic, Pascal, Algol, C, Lisp, or Fortran, you should be able to write numerical programs in Maple very quickly.
- Maple is not strongly typed like C and Pascal. No declarations are required. Maple is more like Basic and Lisp in this respect. However types exist. Type checking is done at run time and must be programmed explicitly.
- Maple is interactive and an interpreting language. Maple is not suitable for running numerically intensive programs because of the interpreter overhead. Though it is suitable for high-precision numerical calculations and as a tool for generating numerical codes.

This document is based on Maple V Release 5. Maple development continues. New releases of Maple are published almoust every year, which contain not only changes to the mathematical capabilities, but also changes to the programming language and user interface. We are pleased and somewhat surprised to see how quickly this movement is already happening. For this reason, we have applied constructs in the language that will be probable in the language in future versions of Maple. Some of the important things being stated in [1] are not mentioned elsewhere, or they are buried in this documentation. By itself or when coupled with other computing environments and Computer Algebra Systems, Maple can be incorporated effectively into the curriculum to enhance the understanding of both fundamental and advanced topics, while enabling the student actively to put theory into practice.
Use the Maple online help facility by typing `?command` at the Maple input, where `command` is the name of the function about which you would like information. Even better, access help from the menu.

Inhaltsverzeichnis

1 Ausgewählte Maple-Befehle zur Grafik

Zusammenstellung von einfachen Maple-Befehlen zur Vorbereitung und Nutzung der Grafik.

Zu Beginn sind im Maple-Arbeitsblatt (Worksheet, Datei $*.mws$) die entsprechenden Programmpakete mit ihren Grafikkomponenten zu laden, insbesondere die Pakete *plots* und *plottools*.
Bei einigen Kommandos sind weitere Notationen mit gleicher Wirkung angegeben.
Man beachte, dass bei der Darstellung mehrerer Anteile in einer Abbildung diese z.B. als Folge, Liste oder Menge zusammengefasst werden können.

`->`
`f:=x->x^2+sin(x);`
`g:=(u,v,w)->u+v*w+1;`
	: Definiert eine ein- bzw. mehrparametrige Funktion mit dem Pfeiloperator.

`unapply(expr(x,y,...),x,y,...)`
`unapply(expr(x,y,...),(x,y,...))`
`f:=unapply(sin(x*y),x,y);`
	: Wandelt den Ausdruck expr der Variablen $x, y, ...$ in einen funktionalen Operator in Pfeilschreibweise um.
	Aus `unapply(sin(x*y),x,y)` wird $(x,y)\text{->}\sin(x*y)$

`z=x^2;`
`f:=unapply(rhs(%),x);`
`plot(f(x),x=-2..2);`
`plot(f,-2..2);`
	: Funktionsdefinition
	: Zeichnet die Kurve $f(x)=x^2,\ x\in[-2,2]$

`plot(sin);`
`plot(sin,-5..5);`
`plot(sin(x),x=-5..5);`
`plot(sin(x),x=-5..5,y=-2..2);`
	: Zeichnet die Kurve $\sin(x),\ x\in[-10,10]$
	: $\sin(x),\ x\in[-5,5]$
	zusätzlich x-Achse beschriften
	: Ordinatenbereich explizit angegeben

`plot(x/(x^2-1),x=-2..2,-7..7);`
`plot(x/(x^2-1),x=-2..2,-7..7,`
`discont=true);`
	: Funktion mit Vertikalen an Polstellen,
	: ohne Vertikalen

`f:=exp(-x^2);`
`plot(f,x=-3..3);`
`plot(f(x),x=-3..3);`
`smartplot(f);`
	: Formeldefinition, "Scheinfunktion"
	: Zeichnet die Kurve $f(x)=e^{-x^2},\ \ x\in[-3,3]$
	Kurve $f(x)=e^{-x^2},\ \ x\in[-10,10]$

`g:=x->exp(-x^4);`
`g:=unapply(exp(-x^4),x);`
`plot(g(x),x=-3..3);`
	: Funktionsdefinition
	: Zeichnet die Kurve $g(x)=e^{-x^4},\ x\in[-3,3]$

`plot([f1(x),f2(x)],x=a..b,`
`color=[red,blue]);`
`plot({f1(x),f2(x)},x=a..b,`
`color=[red,blue]);`
	: Zeichnet die Kurven $y=f_1(x)$ und $y=f_2(x)$ für x aus $[a,b]$ in den Farben rot und blau in einem Koord.system
	Achtung: Menge {..} nicht geordnet

`plot(sin(x),x=-2*Pi..2*Pi,` `scaling=constrained);`	: Gleicher Maßstab der Koordinaten

```
plot(sin(x),x=-100..100,
   numpoints=1000);
```
: Mehr Punkte für Auswertung der Funktion

```
plot({seq(sin(n*x)/n,n=1..6)},
   x=0..4);
```
: Sequenz von Funktionen in einem Koord.system

```
plot([x,y,t=a..b]);
```
: Zeichnet die Parameterkurve $(x = x(t), y = y(t))$ mit t aus $[a, b]$

```
plot([t*sin(3*t),t*cos(3*t),
   t=-2*Pi..2*Pi]);
```
: Parameterkurve

```
implicitplot(f(x,y)=0,
   x=a..b,y=c..d);
```
: Zeichnet die Lösungsmenge der Gleichung $f(x, y) = 0$, Bereich der x-Achse bzw. der y-Achse ist das Intervall $[a, b]$ bzw. $[c, d]$

```
implicitplot(f,a..b,c..d);
```
: Zeichnet die Lösungsmenge der Gleichung $f(x, y) = 0$, Bereich $[a, b] \times [c, d]$

```
f:=(x,y)->x^2+y^2-1
implicitplot(f,-2..2,-2..2,axes=
   boxed,scaling=constrained);
```
: Zeichnet die Lösungsmenge der Gleichung $f(x, y) = 0$, Bereich $[-2, 2] \times [-2, 2]$

```
implicitplot(f(x,y)=0,x=-2..2,
   y=-2..2,axes=boxed,
   scaling=constrained);
```

```
plot([[x1,y1],[x2,y2],
   [x3,y3],...]);
```
: Verbinden der Punkte $P_1(x_1, y_1), P_2(x_2, y_2), P_3(x_3, y_3), ...$ durch einen Polygonzug. Bei einem geschlossenen Polygonzug muss der erste Punkt mit dem letzten übereinstimmen.

```
plot([[1,1],[2,2],[3,2],[4,1]]);
```
: Verbinden der Punkte P_1, P_2, P_3, P_4 durch einen Polygonzug

```
plot([[1,0],[0,-1],[-1,0],
   [0,1],[1,0]],
   scaling=constrained);
```
: Rhombus

```
plot(...,style=point);
```
: Zeichnet eine Punktmenge

```
plot(sin(x),x=0..2*Pi,
   style=point);
```
: $\sin(x)$ als Menge von ca 50 Punkten, Standardsymbol ist Rhombus

```
plot(x,x=0..1,style=point,
   symbol=circle);
```
: x als Punktmenge, Symbol ist Kreis

```
polarplot(1,scaling=constrained);
```
: Kurve in Polarkoordinaten, Kreis

```
polarplot([sin(t),exp(t),
   t=-Pi..Pi],color=gold);
```
: Kurve in Polarkoordinaten

```
display([p1,p2],options);
```
: Vereint Grafiken u.a. in einer Abbildung

```
f:=x^4-8*x^3+22*x^2-22*x+8;          : Funktion
sf:=convert(f,string);               : Konvertierung in Text (string)
p1:=plot(f(x),x=-0.25..4 ):          : Funktionsplot
p2:=textplot([2.3,8,'Polynom']):     : Textbeschriftung
p3:=textplot([2.9,7,'f(x)='.sf]):
display([p1,p2,p3],                  : Gemeinsamer Graph mit Font
   font=[TIMES,ITALIC,12],
   scaling=constrained);

g1:=x->piecewise(x<0,1-x^3,          : Stückweise definierte Funktion
x<Pi/2,1,x>=Pi/2,sin(x));
plot(g1(x),x=-1..4);                 : Zeichnen der Funktion g1(x)

g2:=proc(x)                          : Funktion mit verzögerter Auswertung
  if x<0 then 1-x^3
   elif x<evalf(Pi/2) then 1
   elif x>=evalf(Pi/2) then sin(x)
  fi
end:
plot('g2(x)',x=-1..4);               : Zeichnen der Funktion g2(x)
plot('g2'(x),x=-1..4);

g3:=proc(x)                          : Funktion als "robuste" Prozedur
if type(x,numeric) then
  if x<0 then 1-x^3
   elif x<evalf(Pi/2) then 1
   elif x>=evalf(Pi/2) then sin(x)
  fi
  else 'g3'(x)       # oder 'g3(x)'
fi
end:
plot(g3(x),x=-1..4);                 : Zeichnen der Funktion g3(x)
plot(g3,-1..4);

f:=x^2-y^2;                          : Funktion z = f(x,y)
plot3d(f,x=-1..1,y=-1..1);           : 3D-Plot
plot3d(f,x=-1..1,y=-1..1,            : 3D-Plot als Contour mit anderen Optionen
   style=contour,contours=15,
   color=red,orientation=[0,0],
   scaling=constrained,axes=normal,
   axesfont=[HELVETICA,12]);

contourplot(x^2-y^2,x=-1..1,y=-1..1); : Höhenlinien der Funktion

plot3d([sin(s)*cos(t),cos(s)*cos(t),  : Parametrische 3D-Grafik, Kugel
   sin(t)],s=0..Pi,t=0..2*Pi,
   scaling=constrained);
```

```
spacecurve([t/10,(1+t/10)*sin(t),
   (1+t/10)*cos(t)],t=0..30*Pi,
   numpoints=500,
   axes=framed,shading=none,
   orientation=[25,62]);
```
: Darstellung einer Raumkurve, Trichtergestalt

```
tubeplot([2*sin(t),2*cos(t),t/3],
   t=-1.5*Pi..2*Pi,
   orientation=[45,68]);
```
: Torusähnliche Gebilde, Schlauch

```
sphereplot(1,theta=0..2*Pi,phi=0..Pi,
   scaling=constrained);
```
: 3D-Darstellung in sphärischen Koord., Kugel

```
cylinderplot(0.5,theta=0..2*Pi,z=-2..2);
```
: 3D-Darstellung in Zylinderkoordinaten, Zylinder

```
f:=(x,n)-> sin(n*x)*exp(-x);
```
: Typische Funktion für Animation

```
animate(f(x,n),x=-3..3,n=1..20,
   frames=20,numpoints=100);
```
: Gedämpfte Sinus-Oszillation

```
plots[animate](f(x,n),x=-3..3,n=1..20,
   frames=20, numpoints=100);
```

```
animate(m*x,x=-1..1,m=-1..1);
```
: "Kippende" Strecke

```
kreis:=plot([sin(t),cos(t),t=0..2*Pi],
   -1..1,-1..1,scaling=constrained,
   axes=none):
```
: Kreis

```
zeiger:=t->plot([[0,0],[sin(t),cos(t)]],
   thickness=3,tickmarks=[0,0]):
```
: "Uhrzeiger"

```
n:=64:
drehz:=[seq(zeiger(2*Pi*i/n),i=0..n)]:
```
: Sequenz von Stellungen des Uhrzeigers

```
pdreh:=plots[display](drehz,
   insequence=true,
   scaling=constrained):
```
: Vorbereitung der Animation

```
plots[display]([kreis,pdreh]);
plots[display](kreis,pdreh);
plots[display]({kreis,pdreh});
```
: "Ziffernblatt mit drehendem Uhrzeiger"

```
b:=polarplot([sin(t),exp(t),t=-Pi..Pi]):
c:=animate(m*x,x=-1..1,m=-1..1):
d:=plot(sin(4*Pi*x),x=-1..1,
   color=blue):
plots[display]([b,c,d]);
```
: 3 Kurven

: Abbildung mit 3 Kurven und Animation

Neben diesem Arbeitsblatt *graph22.mws* findet der Leser grafische Strukturen auch in den zu Maple mitgelieferten Beispielen und Demonstrationen sowie in Maple-Unterverzeichnissen EXAMPLES, SHARE oder PLOTS.

2 Ausgabemöglichkeiten unter Maple

2.1 Arbeitsblätter mit Grafik und ihr Export

Was passiert mit den Grafiken aus einem Maple Arbeitsblatt beim Export nach LaTeX? Zunächst bringt der Druck des Arbeitsblatts den darin enthaltenen Bildschirmplot ungefähr in der Größe $10\times10cm$ horizontal zentriert aufs Papier.
Der Export des Arbeitsblatts *name.mws* nach LaTeX erzeugt neben dem LaTeX-File *name.tex* für die darin enthaltenen Plots zusätzlich die entsprechenden Postscript-Dateien (*eps*-Format, Encapsulated PostScript) *name01.eps*, *name02.eps*, usw. Jede einzelne Grafik befindet sich in einer Box mit den Grenzen (72,72,719,540), also der Dimension $647\times468pt$ $= 227\times165mm$, diese ist um 90 Grad gedreht und noch etwas verschoben.

Die Ausgabe des Grafikfiles kann erfolgen

- im Rahmen von Maple-Gruppen.

  ```
  \begin{maplegroup}
   ...
   \mapleresult
   \begin{maplelatex} \mapleplot{exam1a03.eps} \end{maplelatex}
  \end{maplegroup}
  ```

  Die Grafik im File *name01.eps* erscheint in der Ausgabe in einem berandeten Rechteck der Dimension von ca $157\times63mm$ in einer Box von ca $160\times90mm$, also auf der ganzen Breite der Seite mit etwas Platz darunter. Damit wird die Druckgrafik auch im Vergleich zum Original in der Breite gestreckt, siehe Verhältnis 157:63≈2 und 227:165≈1.4 . Skalierungsangaben `width=...` bzw. `height=...` sind im Befehl `\mapleplot{exam1a01.eps}` ohne Wirkung.

  Will man dieselbe Darstellung der Größe $157\times63mm$ mit dem Stil *psfig.sty* erreichen, so muss man nehmen

  ```
  \psfig{figure=exam1a01.eps,width=8.7cm,height=11.3cm,angle=-90}
  ```

  und die folgenden Ausgaben etwas "heranziehen".
  Die folgenden zwei Anweisungen

  ```
  \psfig{figure=exam1a08.eps,width=15.7cm,angle=-90}
  \psfig{figure=exam1a08.eps,width=15.7cm,height=11.3cm,angle=-90}
  ```

  führen auf die gleiche Grafik der Dimension $157\times113mm$.

- außerhalb von Maple-Text mit dem Stil *psfig.sty* oder *epsf.sty*.

  Beispielvarianten

  ```
  % im Original ist Grafik von der Groesse 22.7 x 16.5cm und gedreht
  \psfig{figure=exam1a03.eps}
  % normale Ansicht (Portrait) etwas verkleinert auf 13.5 x 9.75cm
  \psfig{figure=exam1a03.eps,width=13.5cm,angle=-90}

  \epsfbox{exam1a03.eps}
  \epsfbox[0 0 w h]{exam1a03.eps}
  ```

  Weitere Hinweise zum Stil findet man in [12, 13].

2.2 Grafikausgabe

Leider gibt es in Maple keine Möglichkeit, eine auf dem Bildschirm erzeugte Grafik durch eine implementierte Menüfunktion zu exportieren.

Auf eine Variante des Grafik-Exports wurde jedoch schon verwiesen.
Beim Export des Arbeitsblatts in LaTeX erzeugt man neben dem LaTeX-File für die darin enthaltenen Plots zusätzlich die entsprechenden Postscript-Dateien (*eps*-Format). Die Grafikfiles haben das Format A4, *landscape* und sind *monochrom (schwarz/weiß)*. Je nach Inhalt können diese bis mehrere Megabyte groß sein. In einem LaTeX-Dokument werden sie einbezogen, nachdem man sie um -90 Grad dreht und entsprechend verkleinert.

Als zweite und übliche Variante empfiehlt sich der Maple-Befehl

```
interface(plotdevice=..., plotoutput=..., plotoptions=..., );
```

Damit kann man die erzeugten Grafiken wahlweise z.B. als *ps (PostScript)-, gif (Graphics Interchange Format)-, jpg (Joint Picture Experts Group (JPEG))-* oder *pcx(PC Paintbrush)*-File abspeichern.

Wir zitieren aus [1].
The function *interface* is provided as a unique mechanism of communication between Maple and the user interface. Specifically, this function is used to set and query all variables which affect the format of the output but do not affect the computation.
Its arguments are specifies, i.e. for plots

plotdevice : The name of the plotting device.

plotoutput : Name of a file where the plot output will be stored.

plotoptions : Contain device specific options to be passed to the device driver.

Using the *interface* command you are able to store pictures.

Vorgehensweise

Ein 3D-Plot soll als Postscript-File im Standardformat *landscape* und *monochrom* mit gegebener Größe erstellt werden.

```
> restart:
  with(plots):
  plot3d(sin(x)*exp(y), x=0..10,y=1..4);

> interface(plotdevice=ps,
      plotoutput=`D:/Neundorf/Maple2/bild1.ps`,
      plotoptions=`width=800,height=600`);
  plot3d(sin(x)*exp(y),x=0..10,y=1..4);
> interface(plotdevice=win); # Standard output
```

Es ist empfehlenswert, nach der obigen Umlenkung der Ausgabe auf das File im angegebenen Verzeichnis anschließend sofort wieder die Standardausgabe auf dem Bildschirm zu aktivieren, da sonst alle nachfolgenden Grafiken ebenso unter dem gewählten Filenamen abgespeichert werden und damit die erste Datei ohne Vorwarnung überschrieben wird.

Analog kann die Umlenkung auch im Zusammenhang mit dem Befehl

```
plotsetup(devicetype, plotoutput=..., plotoptions=...)
```

erfolgen. Dies funktioniert wie eine Parametereinstellung für Plots.
Dann hat man die Kommandofolge

```
> restart:
  with(plots):
  interface(plotdevice=win);
> plotsetup(ps,
      plotoutput=`D:/Neundorf/Maple2/bild21.ps`,
      plotoptions=`portrait,noborder,with=350,height=260`);
  plot(1/4*(2*x-2)/(1+x^2)+1/2*arctan(x),x=-2..2);
> plotsetup(default); # Standard output
  plot(1/4*(2*x-2)/(1+x^2)+1/2*arctan(x),x=-2..2);
```

Noch einige allgemeine Bemerkungen zu den beiden Varianten.

2.2.1 Der Befehl `interface`

Die Angabe des Grafikfiles erfolgt vorzugsweise mit LW:/Pfad/Dateiname.

plotdevice	plotoutput File *.*	plotoptions
ps	name.ps	Standard: *landscape, s/w* möglich auch *portrait, color* u.a.
gif	name.gif	Standard: *color, portrait, transparent=false* möglich auch *transparent=true* (transparent)
jpeg	name.jpg	Standard: *color, portrait*
pcx	name.pcx	Standard: *color, portrait*

Wir vergleichen weiterhin den Speicherbedarf der Grafikfiles.
Dazu führen wir folgendes Maple-Arbeitsblatt aus.

Datei: ani0_.mws

```
> restart:  with(plots):  with(plottools):
  # Arbeitsverzeichnis
  pfad := `D:/Neundorf/Maple2/`;
  # Graphik output files
  name1 := `ani0_01.ps`;
  name2 := `ani0_02.pcx`;
  name3 := `ani0_03.gif`;
  name4 := `ani0_04.jpg`;
  name5 := `ani0_05.ps`;

  datei1 := cat(pfad,name1);
  datei2 := cat(pfad,name2);
  datei3 := cat(pfad,name3);
  datei4 := cat(pfad,name4);
  datei5 := cat(pfad,name5);
```

```
> # Plot von f(x,y)
  pl1:=plot3d(sin(x)*exp(y),x=0..10,y=2..5, style=patch,
    axes=BOXED, thickness=1, labels=[`x`,`y`,`f`],
    title=`f(x,y), x=0..10,y=2..5`):
  Q:=plots[display](pl1):
  plots[display](Q);

> # ps-File:  monochrome and landscape orientation
  interface(plotdevice=ps,
    plotoutput=datei1, plotoptions=`width=640,height=480`);
  plots[display](Q);
  interface(plotdevice=win);

> # Speichern in anderen Bildformaten
  interface(plotdevice=pcx,
    plotoutput=datei2, plotoptions=`width=640,height=480`);
  plots[display](Q);
  interface(plotdevice=win);
> interface(plotdevice=gif,
    plotoutput=datei3, plotoptions=`width=640,height=480`);
  plots[display](Q);
  interface(plotdevice=win);
> interface(plotdevice=jpeg,
    plotoutput=datei4, plotoptions=`width=640,height=480`);
  plots[display](Q);
  interface(plotdevice=win);

> # ps-File:  color and portrait orientation
  interface(plotdevice=ps,
    plotoutput=datei5,
    plotoptions=`color,portrait,width=640,height=480`);
  plots[display](Q);
  interface(plotdevice=win);
```

Dazu nehmen wir noch den ersten Plot Q, der bei Export in LaTeX als Datei *ani0_01.eps* entsteht. Weiterhin transformieren wir mit dem Grafikprogramm **Micrografx Picture Publisher** (MPP) das *pcx*−File (analog sind *gif* und *jpg*) auf das *eps*−Format und nennen diese Datei *ani0_02.eps*. Dieses File ist allgemein sehr groß und ist am Anfang sowie am Ende noch von störenden bzw. überflüssigen Zeichen unbedingt zu "säubern".
Die Dateigrößen weisen erhebliche Unterschiede auf.

File	Größe in Byte
ani0_01.ps	121 950
ani0_02.pcx	105 997
ani0_03.gif	22 233
ani0_04.jpg	163 840
ani0_01.eps	121 980
ani0_02.eps	2 520 580
ani0_05.ps	147 738

Das *ps*– und *eps*–File wird in Graustufen gezeichnet. Nach Drehung erhält man die Ansicht als Portrait.

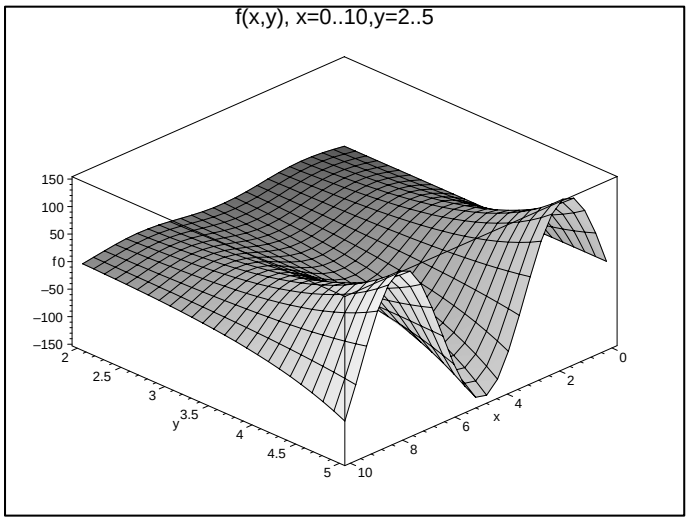


Abb. ani0_01.ps, ani0_01.eps, landscape orientation, s/w → Drehung
$f(x,y) = \sin(x)e^y,\ x = 0..10,\ y = 2..5$

Die in anderen Formaten erzeugten Files haben keinen Rahmen und Titel, so dass auch das mit MPP daraus erzeugte sehr große *eps*–File ohne Rahmen und Titel ausgegeben wird.

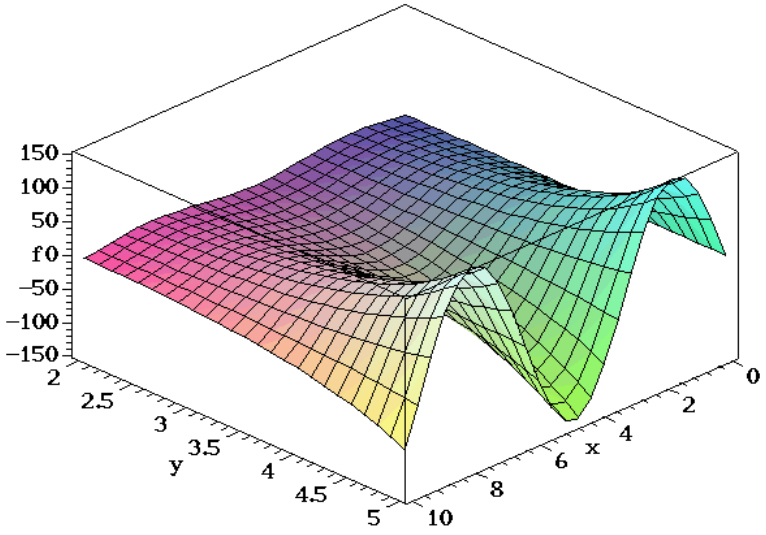


Abb. ani0_02.eps mittels MPP erzeugt aus ani0_02.pcx, *.gif, *.jpg
$f(x,y) = \sin(x)e^y,\ x = 0..10,\ y = 2..5$

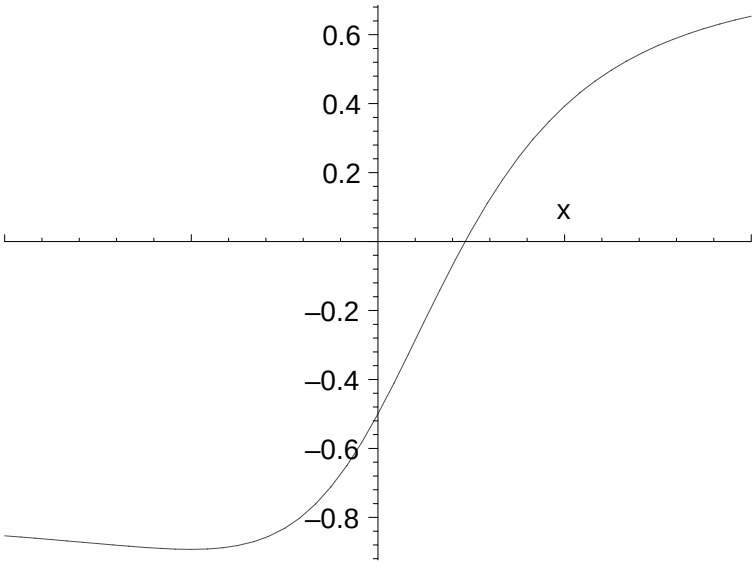
2.2.2 Der Befehl `plotsetup`

Als *devicetype* können z.B. *ps, jpeg, gif, pcx, default, char, tek, x11* genommen werden, vorausgesetzt, die zugehörigen DDL-Files (Dynamic Link Library) sind vorhanden.
Es gilt für Maple V4 *default=char*, und dabei erfolgt die Ausgabe in Pseudografik mit Zeichen wie auf einem Textbildschirm. Für Maple V5 bedeutet *default* soviel wie `interface(plotdevice=win)` und damit die übliche Grafikausgabe.
Die Dateinamen wähle man vollständig, um zu vermeiden, dass ohne diese Angaben die Files automatisch in Unterverzeichnissen von Maple, z.B. in BIN.WIN, BIN.W95 (falls dort natürlich Zugriffsrechte bestehen), und mit Standardnamen wie *postscript.out, plot.pcx, plot.gif, plot.jpg* abgelegt werden. Dazu braucht Maple eben die entsprechenden DLL.

Anwendungsbeispiel

Datei: expo2.mws

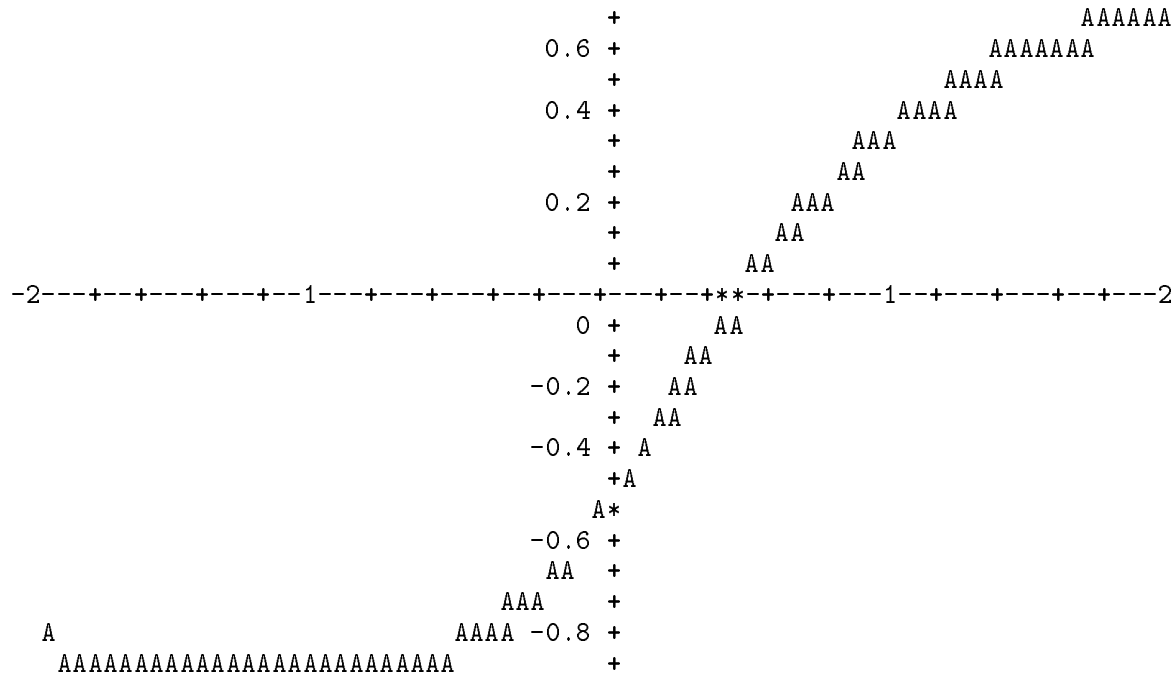
```
> restart: with(plots):
> plot(1/4*(2*x-2)/(1+x^2)+1/2*arctan(x),x=-2..2);
> plotsetup(ps,
  plotoutput='D:/Neundorf/Maple2/bild21.ps',
    plotoptions='portrait,noborder,width=350,height=260');
```



Das Grafikfile *bild21.ps*, ausgegeben mit dem Stil *psfig*, mit den Größenangaben $350 \times 260pt = 123 \times 91mm$ enthält eine Bounding Box [291 165 500 446] der Dimension $209 \times 281pt = 74 \times 99mm$. Dies ist die Höhe-Breite-Ausdehnung des eigentlichen Plots. Wegen dem kleineren Plot im Bereich sind über, links von und unter der Ausgabe des Bildes im LaTeX-Dokument die Abstände zu regulieren.

```
> plot(1/4*(2*x-2)/(1+x^2)+1/2*arctan(x),x=-2..2);
> plotsetup(default); # MapleV4:default -> char
                      # MapleV5:default -> interface(plotdevice=win);
> plot(1/4*(2*x-2)/(1+x^2)+1/2*arctan(x),x=-2..2);
```

```
 # Maple V Release 4 mit  default=char
```




```
> interface(plotdevice=win); # Standard output
> plot(1/4*(2*x-2)/(1+x^2)+1/2*arctan(x),x=-2..2);
> plotsetup(tek);
> plot(1/4*(2*x-2)/(1+x^2)+1/2*arctan(x),x=-2..2);

Plotting error, no plot device driver for plotdevice=tek
> plotsetup(x11);
> plot(1/4*(2*x-2)/(1+x^2)+1/2*arctan(x),x=-2..2);

Plotting error, no plot device driver for plotdevice=x11

> # plotsetup(ps); # Datei postscript.out in MAPLEV5S -> BIN.W95,
                   # falls dort Schreibrechte vorliegen
  plotsetup(ps,plotoutput=`D:/Neundorf/Maple2/bild22.ps`);
> plot(1/4*(2*x-2)/(1+x^2)+1/2*arctan(x),x=-2..2);
> # plotsetup(pcx); # Datei plot.pcx
  plotsetup(pcx,plotoutput=`D:/Neundorf/Maple2/bild23.pcx`);
> plot(1/4*(2*x-2)/(1+x^2)+1/2*arctan(x),x=-2..2);
> # plotsetup(gif); # Datei plot.gif
  plotsetup(gif,plotoutput=`D:/Neundorf/Maple2/bild24.gif`);
> plot(1/4*(2*x-2)/(1+x^2)+1/2*arctan(x),x=-2..2);
> # plotsetup(jpeg); # Datei plot.jpg
  plotsetup(jpeg,plotoutput=`D:/Neundorf/Maple2/bild25.jpg`);
> plot(1/4*(2*x-2)/(1+x^2)+1/2*arctan(x),x=-2..2);

> interface(plotdevice=win); # Standard output
```

Zu den Dateigrößen.

File	Größe in Byte
bild21.ps	5 995
bild22.ps	6 004
bild23.pcx	13 063
bild24.gif	2 784
bild25.jpg	32 768

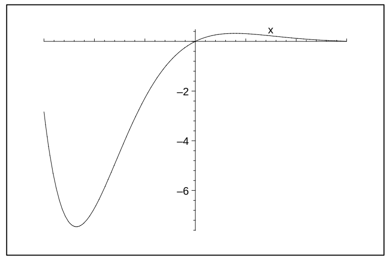
2.3 Export einer Sequenz von Abbildungen

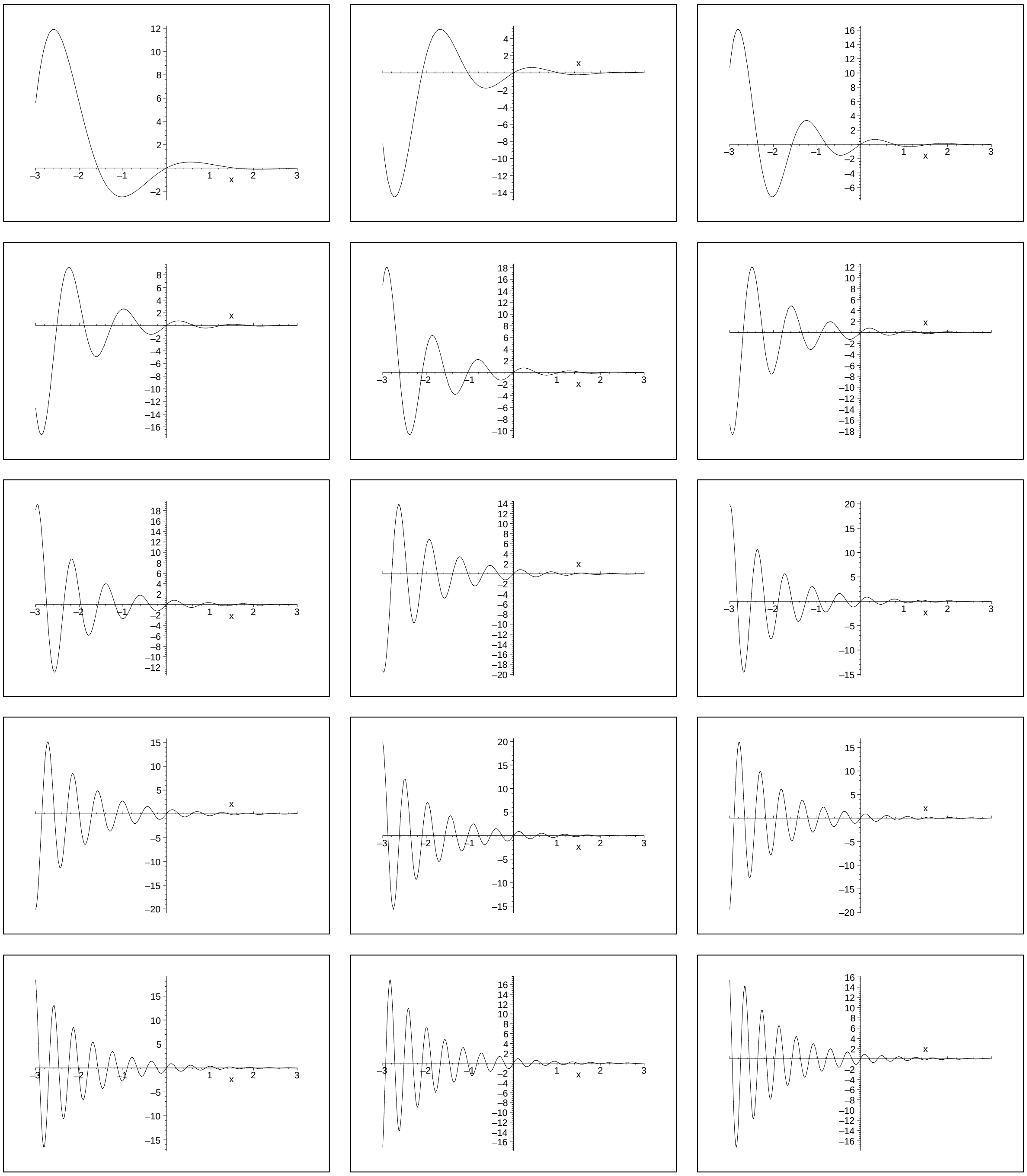
Wir erzeugen eine Sequenz $\{f(x,n):\ n=1,2,...,N,\ N\le 999\}$ von Darstellungen der Funktion $f(x,n)=\sin(nx)\,e^{-x},\ \ x\in[-3,3]$.
Die Dateinamen der N Funktionsgraphen sollen sich nur in der laufenden Nummerierung unterscheiden. Der allgemeine Bezeichner sei 'D:/Neundorf/Maple2/frame$***$.ps', wobei auf den Positionen $***$ (Joker) die Nummerierung fortlaufend gemäß 001, 002, ..., 009, 010, ..., 016 einzutragen ist. Dieser Dateiname wird durch eine Verknüpfung von Zeichenketten gebildet: `cat(pfad,'frame',null,n,'.ps')`.

```
> restart:
  with(plots):

> f:=(x,n)->sin(n*x)*exp(-x):
> frames:=16:  # 1..999
> null:=NULL:
> pfad:='D:/Neundorf/Maple2/':
  filename:='cat(pfad,'frame',null,n,'.ps')':  # null+n = Joker
> for n from 1 to frames do
    for i to 1 do nnull:=2-floor(log[10](evalf(n))):  od:
                        # Anzahl der Nullen
                        # Unterdrueckung der Ausgabe
    for m to nnull do
      null:=cat(null,'0'):
    od:
    interface(plotdevice=ps,plotoutput=filename,
      plotoptions='width=300,height=200'):
    plot(f(x,n),x=-3..3,numpoints=100,color=black):
    interface(plotdevice=win);
    lprint('Creating ',filename);
    for i to 1 do null := NULL: od:  # Unterdrueckung der Ausgabe
  od;

Creating    D:/Neundorf/Maple2/frame001.ps
Creating    D:/Neundorf/Maple2/frame002.ps
Creating    D:/Neundorf/Maple2/frame003.ps
Creating    D:/Neundorf/Maple2/frame004.ps
.......................................
Creating    D:/Neundorf/Maple2/frame015.ps
Creating    D:/Neundorf/Maple2/frame016.ps
```

Die 16 Frames der Funktion $f(x,n) = \sin(nx)\ e^{-x}$, $x \in [-3,3],\ n = 1,2,...,16$.

Bei Animationen ist die Standardanzahl der Frames gleich 16.





3 Beispiele mit Dateiarbeit und Grafik

3.1 1. Beispiel: Newtoniteration

Erinnern wir uns an das Newtonsche Näherungsverfahren zur Lösung der skalaren Gleichung $f(x) = 0$.

```
> restart;
  x[k+1] = x[k] - f(x[k])/fs(x[k]); # fs=f'
```

$$x_{k+1} = x_k - \frac{\mathrm{f}(x_k)}{\mathrm{fs}(x_k)}$$

Ziel ist es, das Iterationsverfahren mit einer vorgegebenen Iterationsanzahl n zu rechnen, und dabei sowohl die Zwischenwerte in einer Datei *newton1.res* zu speichern als auch den Verlauf der Annäherung an die Nullstelle grafisch zu illustrieren.

Also nehmen wir die grafische Darstellung der Funktion und des Iterationsverlaufes als letzten Befehl in der Prozedur auf, da dort nur die letzte Anweisung zurückgegeben wird. Die numerischen Werte werden "vorher" über die *print* oder *fprintf*-Anweisung ausgegeben.

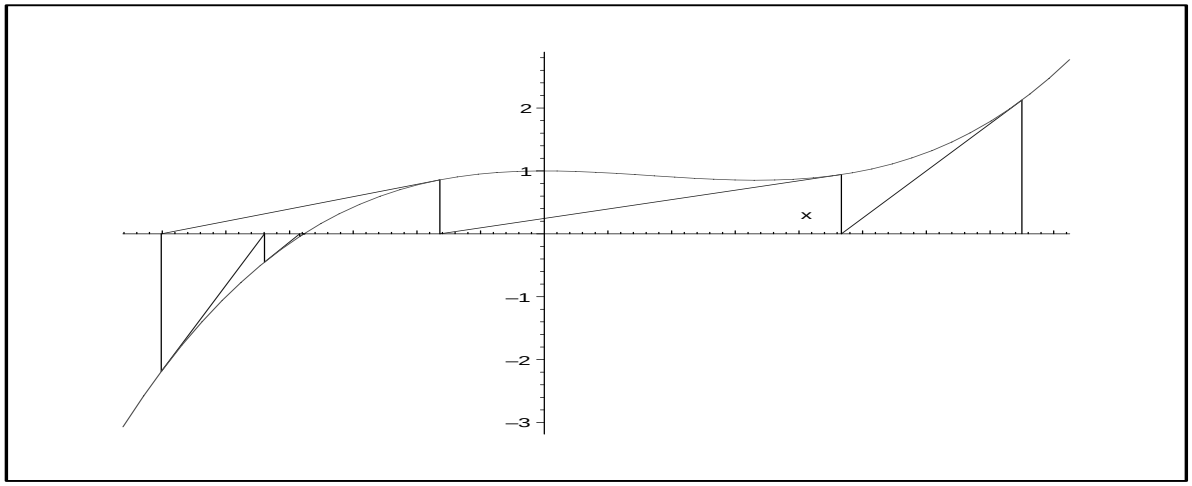
```
> restart;
  with(plots):

  Newton1 := proc(f::procedure,xstart::numeric,n::posint)
    local i,x,xold,xnew,xiter,df,fxold,fxnew,delta,file1,
          curve,s,t,ps,pt,plts,pltt,xleft,xright,Digits_old;
  Digits_old:= Digits;
  Digits := 22;
  file1 := fopen(`D:/Neundorf/Maple2/newton1.res`,WRITE);
  df := D(f); # Ableitung von f
  xold := xstart;
  xiter := xold;
  fxold := evalf(f(xold));
  s := [[xold,0],[xold,fxold]];
  ps[0] := plot(s,color=black); # Anfangsordinate
  fprintf(default,` x = %.20e, f(x)= %.20e\n`,xold,fxold);
  fprintf(file1, `x = %.20e, f(x)= %.20e\n`,xold,fxold);
  for i from 1 to n do
    xnew := evalf(xold-fxold/df(xold));
    delta := abs(xold-xnew);
    fxnew := evalf(f(xnew));
    xiter := xiter,xnew;
    s := [[xnew,0],[xnew,fxnew]];
    t := [[xold,fxold],[xnew,0]];
    ps[i] := plot(s,color=black); # Ordinate
    pt[i] := plot(t,color=blue);  # Tangente
    xold := xnew;
    fxold := fxnew;
    fprintf(default,` x = %.20e, f(x)= %.20e\n`,xold,fxold);
    fprintf(file1, `x = %.20e, f(x)= %.20e\n`,xold,fxold);
  od;
```

```
   fprintf(default,` delta = %.20e\n`,delta);
   fprintf(file1, `delta = %.20e\n`,delta);
   fclose(file1);
   print(`xiter=`,xiter);
   plts := seq(ps[i],i=0..n);
   pltt := seq(pt[i],i=1..n);
   xleft := min(xiter)-0.1*abs(min(xiter));
   xright:= max(xiter)+0.1*abs(max(xiter));
   curve := plot(f(x), x=xleft..xright);
   Digits := Digits_old;
   display(curve,plts,pltt);
   end:
>  f := x->x^3-x^2+1;
   Newton1(f,1.5,8);
```

$$f := x \to x^3 - x^2 + 1$$

```
x =   1.50000000000000000000e+00,  f(x)=   2.12500000000000000000e+00
x =   9.33333333333333333333e-01,  f(x)=   9.41925925925925925926e-01
x =  -3.28174603174603174605e-01,  f(x)=   8.56957494303570928664e-01
x =  -1.20311660303173787304e+00,  f(x)=  -2.18898828343201634623e+00
x =  -8.78759664481079919828e-01,  f(x)=  -4.50813059994179589027e-01
x =  -7.68108290150601923652e-01,  f(x)=  -4.31668208071645282770e-02
x =  -7.55051916522017205589e-01,  f(x)=  -5.61058895011453755000e-04
x =  -7.54877697030048531745e-01,  f(x)=  -9.91001278544930000000e-08
x =  -7.54877666246693721015e-01,  f(x)=  -3.09361500000000000000e-15
delta =  3.07833548107297000000e-08
```

$$xiter = , 1.5, .93333333333333333333, -.32817460317460317460 47, \\ -1.20311660303173787304 0, -.87875966448107991982 78, \\ -.76810829015060192365 20, -.75505191652201720558 89, \\ -.75487769703004853174 51, -.75487766624669372101 54$$

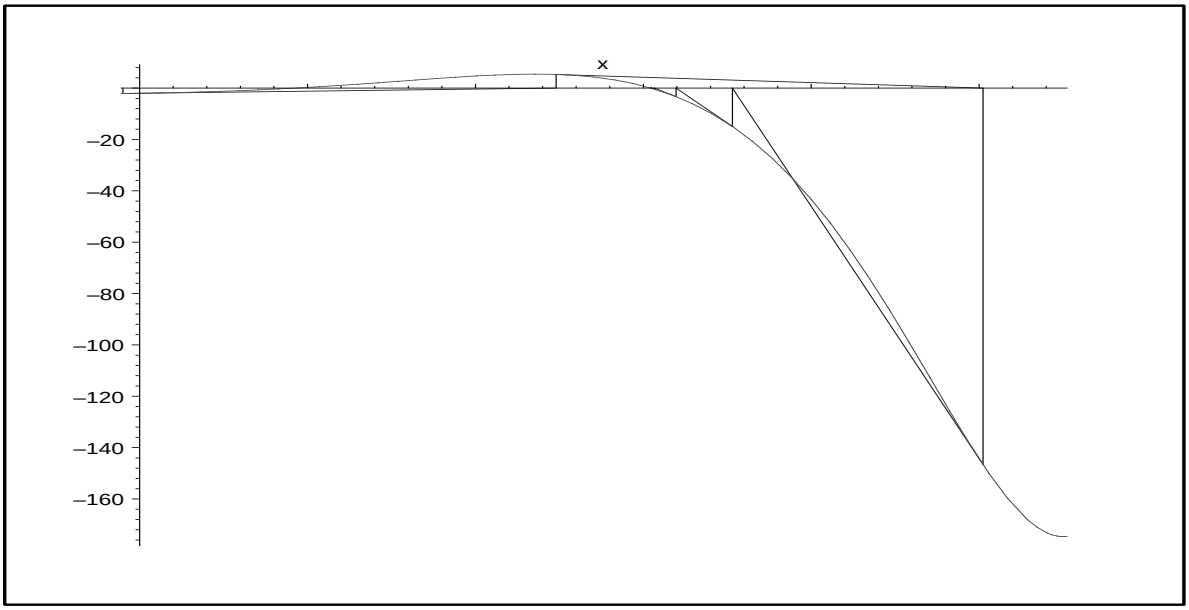


```
> g := x->sin(x)*exp(x)-2;
  Newton1(g,-0.1,8);
```

$$g := x \to \sin(x)\, e^{x} - 2$$

```
x = -1.00000000000000000000e-01,  f(x)=  -2.09033301095242417027e+00
x =  2.48070905057007739414e+00,  f(x)=   5.33492205103551089744e+00
x =  5.02260329350068651526e+00,  f(x)=  -1.46560007875805699853e+02
x =  3.53043210455090763438e+00,  f(x)=  -1.49424909916662889010e+01
x =  3.19489265899230936928e+00,  f(x)=  -3.30030686411928154972e+00
x =  3.06634199214024290438e+00,  f(x)=  -3.86400413238618369132e-01
x =  3.04681587892407230247e+00,  f(x)=  -8.10303880055346420900e-03
x =  3.04638854531580740555e+00,  f(x)=  -3.82585224598028400000e-06
x =  3.04638834335942100605e+00,  f(x)=  -8.54227834000000000000e-13
delta =  2.01956386399502000000e-07
```

$$xiter =, -.1, 2.480709050570077394137, 5.022603293500686515262,$$
$$3.530432104550907634380, 3.194892658992309369279,$$
$$3.066341992140242904379, 3.046815878924072302469,$$
$$3.046388545315807405547, 3.046388343359421006045$$



3.2 2. Beispiel: Schneeflockenkurve

Zur Entstehung der Schneeflockenkurve

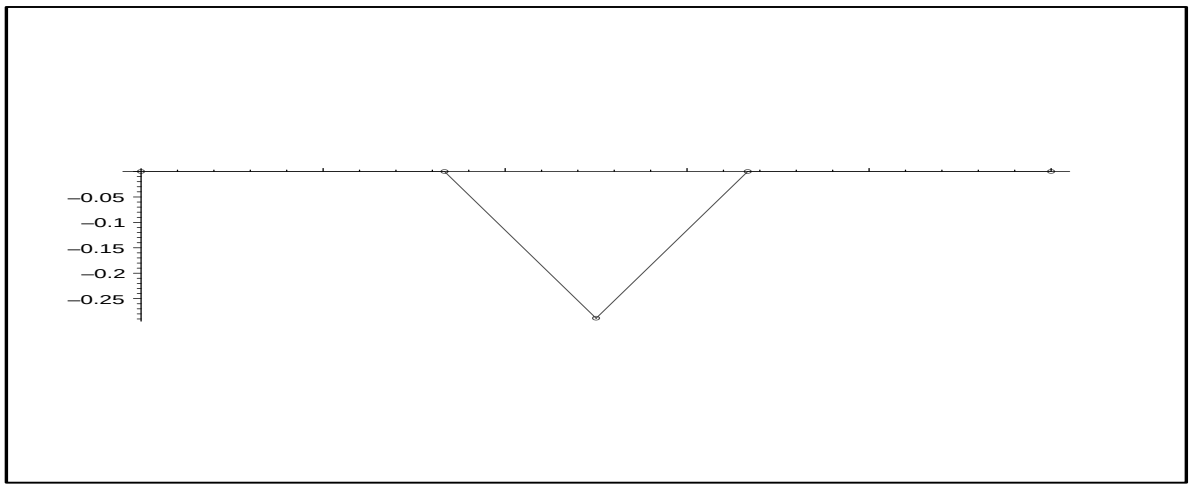
Der Rand der Kurven ist ein Polygonzug. Er entsteht, indem ausgehend von einem gleichseitigen Dreieck jede Seite, definiert durch ihre beiden Eckpunkte, jeweils gedrittelt wird und ein Dreieck auf der "Mitte" dieser Seite errichtet wird.

Der benutzte Datentyp ist die Liste, denn aus einer Liste mit zwei Punkten (Punkt ist selbst Liste seiner Koordinaten) kann man einfach einen Polygonzug durch fünf Punkte erzeugen.

```
> restart:
  with(plots): with(plottools):
> u := [0,0]; v := [1,0];
  x0 := op(1,u): y0 := op(2,u):
  x1 := op(1,v): y1 := op(2,v):
  x2 := x0+1/3*(x1-x0): y2 := y0+1/3*(y1-y0):
  x3 := x0+2/3*(x1-x0): y3 := y0+2/3*(y1-y0):
  x4 := 1/2*(x2+x3+(y3-y2)*sqrt(3)): y4 := 1/2*(y2+y3+(x2-x3)*sqrt(3)):

  pz := [u,[x2,y2],[x4,y4],[x3,y3],v];
  p1 := plot(pz,style=point,symbol=CIRCLE):
  p2 := plot(pz,thickness=2):
  plots[display](p1,p2,scaling=constrained);
```

$$u := [0, 0]$$
$$v := [1, 0]$$

$$pz := [[0, 0], [\frac{1}{3}, 0], [\frac{1}{2}, -\frac{1}{6}\sqrt{3}], [\frac{2}{3}, 0], [1, 0]]$$



Diesen Prozess realisiert die Prozedur *auft*: zu zwei gegebenen Punkten erzeugt sie die fünf neuen Punkte einer Seite (Verfeinerung).
Wenn man dem zusätzlichen reellen Parameter *vz* andere Werte als 1 gibt, entstehen mehr oder weniger bizarre Kurven.

```
> auft := proc(u,v::list)
    local x,y,d,n,x0,y0,x1,y1,x2,y2,x3,y3,x4,y4,p,vz;
    vz := 1; # Parameter
    x0 := op(1,u); y0 := op(2,u);
    x1 := op(1,v); y1 := op(2,v);
    x2 := x0+vz*1/3*(x1-x0); y2 := y0+vz*1/3*(y1-y0);
    x3 := x0+vz*2/3*(x1-x0); y3 := y0+vz*2/3*(y1-y0);
    x4 := 1/2*(x2+vz*x3+(y3-y2)*sqrt(3));
    y4 := 1/2*(y2+vz*y3+(x2-x3)*sqrt(3));
    u,[x2,y2],[x4,y4],[x3,y3],v;
  end:
```

Ecken des Ausgangsdreiecks und seine Darstellung

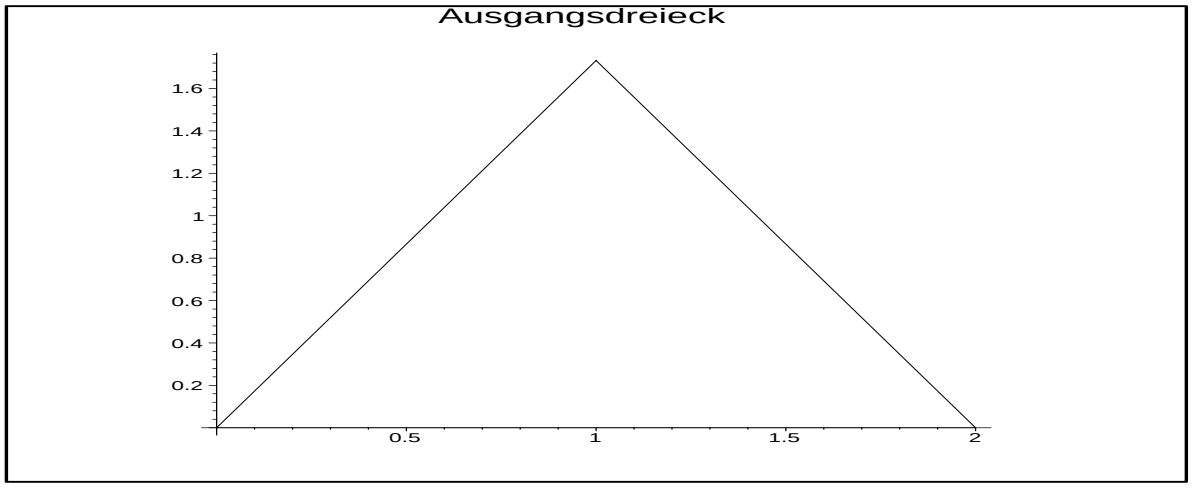
```
> Liste := [[0,0],[2,0],[1,3^(1/2)],[0,0]];
  n := nops(Liste)-1;
  pmax := 5; # Anzahl der Schneeflocken
  p := array(1..pmax,[]):
  p[1] := polygon(Liste); # Ausgangsdreieck = 1.Schneeflocke
  plots[display](p[1],scaling=constrained,title=`Ausgangsdreieck`);
```

$$Liste := [[0, 0], [2, 0], [1, \sqrt{3}], [0, 0]]$$

$$n := 3$$

$$pmax := 5$$

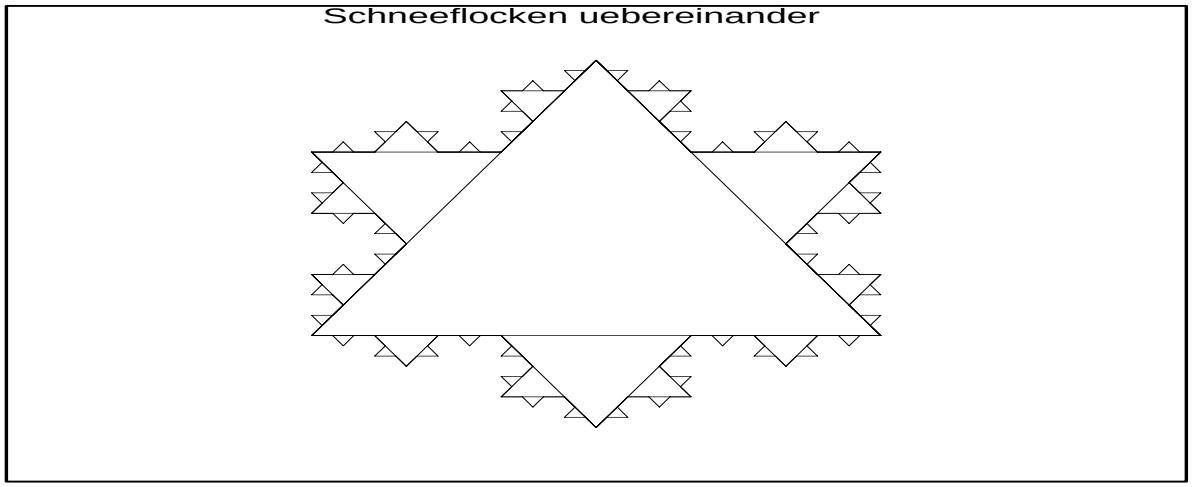
$$p_1 := \mathrm{POLYGONS}([[0, 0], [2., 0], [1., 1.732050808], [0, 0]])$$

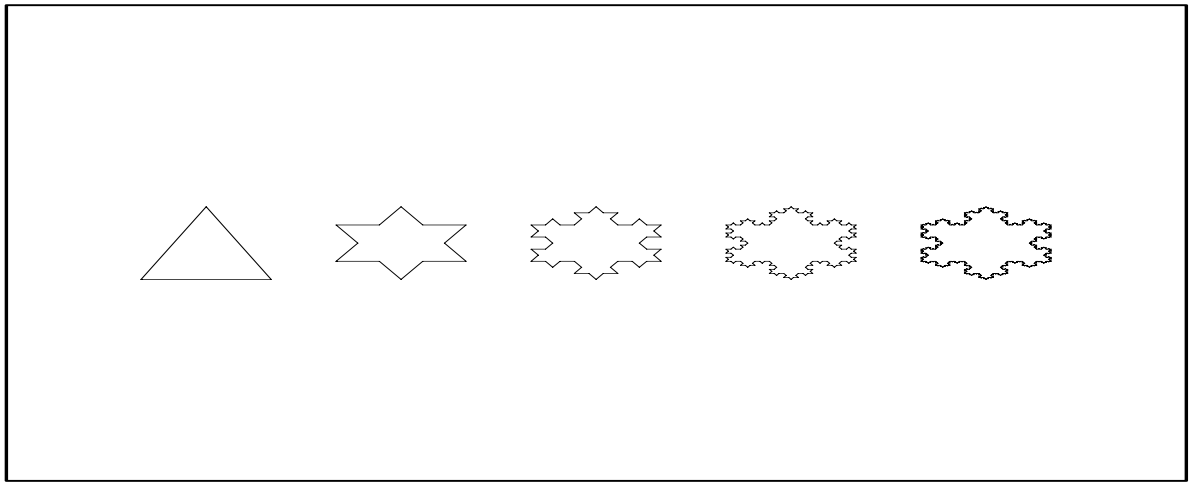


Jede Seite der Schneeflocke wird aus der Liste in eine Subliste übertragen und mittels *auft* verfeinert.
Die Zusammenfassung aller Sublisten ergibt die nächste Schneeflocke.

```
> for k from 2 to pmax do
    n := nops(Liste)-1:
    for i from 1 to n do SubListe.i := [op(i,Liste),op(i+1,Liste)] od:
    for i from 1 to n do SubListe.i := [auft(op(SubListe.i))] od:
    Liste:=[]:
    for i from 1 to n do
      Liste := [op(Liste),op(1..nops(SubListe.i)-1,SubListe.i)] od:
    Liste := [op(Liste),op(nops(SubListe.n),SubListe.n)]:
    p[k] := polygon(Liste):
  od:
```

```
> plots[display](seq(p[j],j=1..pmax-1),axes=none,scaling=constrained,
    title=`Schneeflocken uebereinander`);
  plots[display](p,axes=none,scaling=constrained); # SF nebeneinander
```





Umfang der letzten Schneeflocke
Liste enthält alle Ecken des Polygons.

```
> n := nops(Liste)-1:
  for i from 1 to n do SL[i] := [op(i,Liste),op(i+1,Liste)]; od:
  i := 'i':
  sum(((SL[i][1][1]-SL[i][2][1])^2+
      (SL[i][1][2]-SL[i][2][2])^2)^(1/2),i=1..n);
```

$$\frac{256}{2187}\sqrt{4}\sqrt{6561}$$

```
> radnormal(%); # etwas besser verstaendlich
```

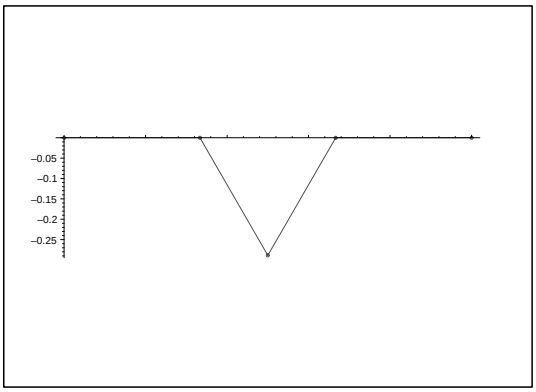
$$\frac{512}{27}$$

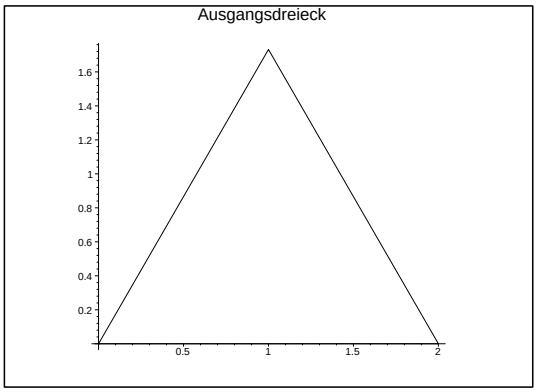
Der Umfang der Kurven nimmt in jedem Schritt um 4/3 zu.

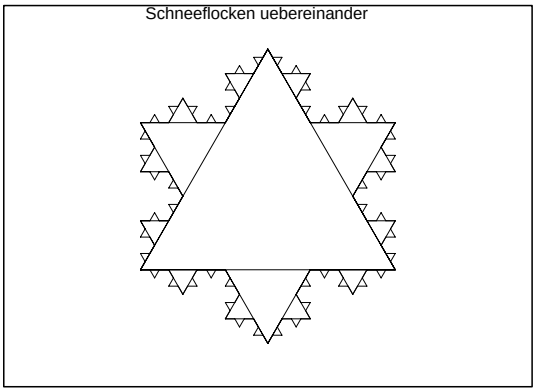
```
> L:=6;
  for i from 2 to pmax do L:=4/3*L; od;
```

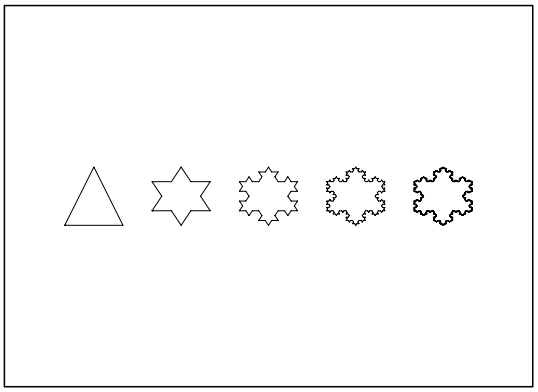
$$L := 6$$
$$L := 8$$
$$L := \frac{32}{3}$$
$$L := \frac{128}{9}$$
$$L := \frac{512}{27}$$

Die nicht verzerrten aber verkleinerten Bilder zu den Schneeflocken in einer Übersicht









3.3 3. Beispiel: Visualisierung der Konvergenz einer Reihe

Wir betrachten die Funktionenreihe $s(x) = \sum_{k=0}^{\infty} a_k(x)$, ihre Partialsummen $s_n(x)$ und untersuchen die Genauigkeit für verschiedene Indizes n.

```
> restart:
  with(plots):
  with(plottools):
```

```
> a:=(4/5*sin(x))^k;
```

$$a := \left(\frac{4}{5}\sin(x)\right)^k$$

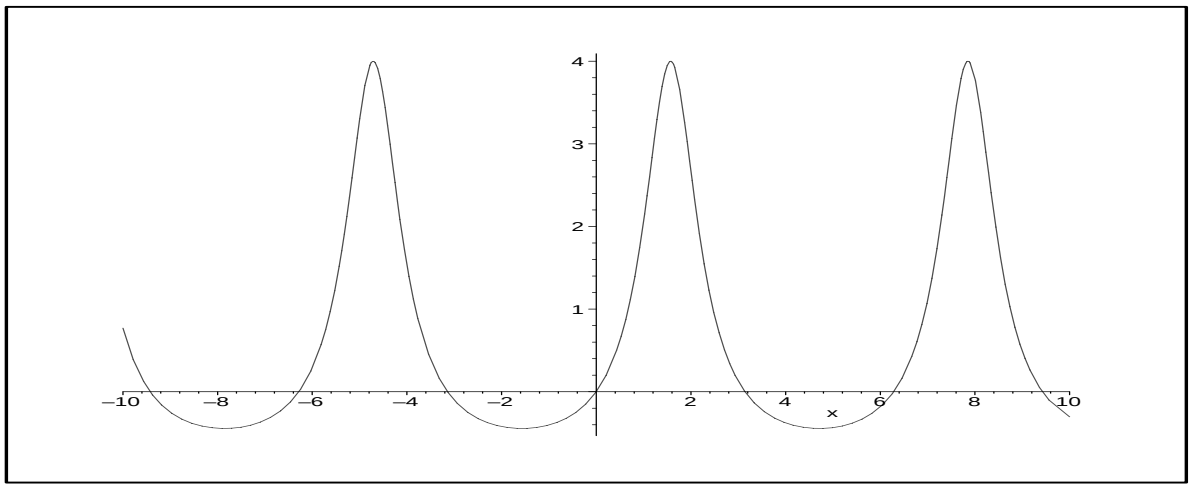
```
> Sum(a,k=1..infinity):
  %=value(%);
```

$$\sum_{k=1}^{\infty} \left(\frac{4}{5}\sin(x)\right)^k = \frac{4}{5}\,\frac{\sin(x)}{1-\frac{4}{5}\sin(x)}$$

```
> summe:=s=rhs(%);
```

$$summe := s = \frac{4}{5}\,\frac{\sin(x)}{1-\frac{4}{5}\sin(x)}$$

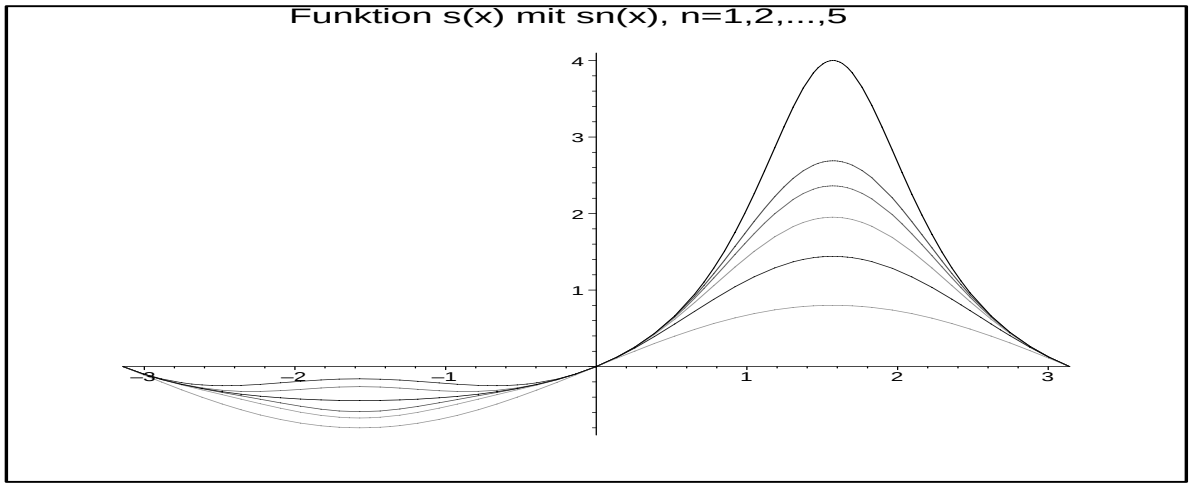
```
> smartplot(4/5*sin(x)/(1-4/5*sin(x)));
```



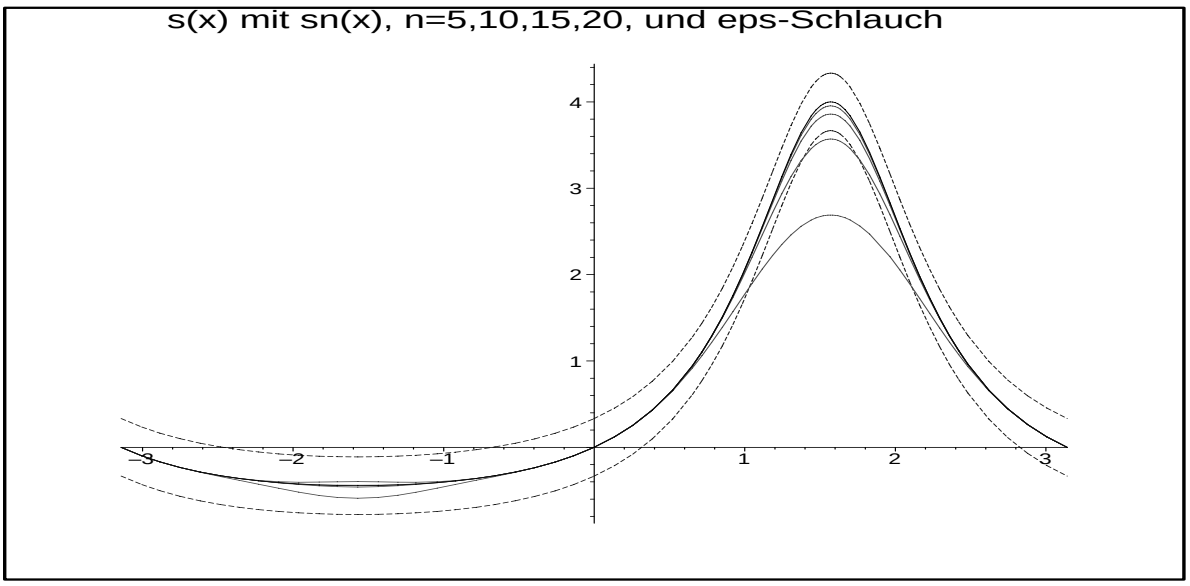

```
> f:=unapply(rhs(summe),x);
```

$$f := x \to \frac{4}{5}\,\frac{\sin(x)}{1-\frac{4}{5}\sin(x)}$$

```
> farbe:=[green,blue,coral,magenta,red]:
> p:=plot(f,-Pi..Pi,color=black,thickness=2):
> display([p,seq(plot(sum(a,k=1..n),x=-Pi..Pi,color=farbe[n]),n=1..5)],
  title=‘Funktion s(x) mit sn(x), n=1,2,...,5‘);
```




```
> p0:=plot({f-1/3,f+1/3},-Pi..Pi,color=blue,linestyle=3,thickness=2):
> n:='n':
  q:=plot({seq(sum(a,k=1..5*n),n=1..4)},x=-Pi..Pi,color=red):
> display([p,q,p0],
  title=‘s(x) mit sn(x), n=5,10,15,20, und eps-Schlauch‘);
```



Man erkennt, dass die Partialsummen $s_{15}(x)$ und $s_{20}(x)$ sich im ε-Schlauch um $s(x)$ befinden, d.h. wir erhalten gleichmäßige Konvergenz der Funktionenreihe sowie

$$\|s(x) - s_n(x)\|_\infty = \max_{x\in[-3,3]} |s(x) - s_n(x)| \le \varepsilon = \frac{1}{3}, \quad n = 15, 20.$$

3.4 4. Beispiel: Das Gibbs'sche Phänomen

Die Funktion $f(x)$ besitze die Periode $T = 2\pi$. Dann setzt man das trigonometrische Polynom oder Fourierpolynom $\Phi_n(x)$ für $f(x)$ folgendermaßen an.

$$\Phi_n(x) = \frac{a_0}{2} + \sum_{k=1}^{n} [a_k \cos(kx) + b_k \sin(kx)].$$

Die Fourierkoeffizienten lauten

$$a_k = \frac{1}{\pi} \int_0^{2\pi} f(x) \cos(kx)\, dx, \quad k = 0, 1, 2, ..., n,$$

$$b_k = \frac{1}{\pi} \int_0^{2\pi} f(x) \sin(kx)\, dx, \quad k = 1, 2, ..., n.$$

Sei $f(x)$ periodisch auf $[0, 2\pi]$ und stückweise gegeben. Wir entwickeln diese Funktion nun in eine Fourierreihe, berechnen Fourierkoeffizienten und -polynome.

```
> restart:
  with(plots):
  with(plottools):

> f:=piecewise(x=0,0,0<x and x<Pi,Pi/2,x=Pi,0,Pi<x and x<2*Pi,-Pi/2);
```

$$f := \begin{cases} 0, & x = 0 \\ \frac{1}{2}\pi, & -x < 0 \ \ and \ \ x - \pi < 0 \\ 0, & x = \pi \\ -\frac{1}{2}\pi, & \pi - x < 0 \ \ and \ \ x - 2\pi < 0 \end{cases}$$

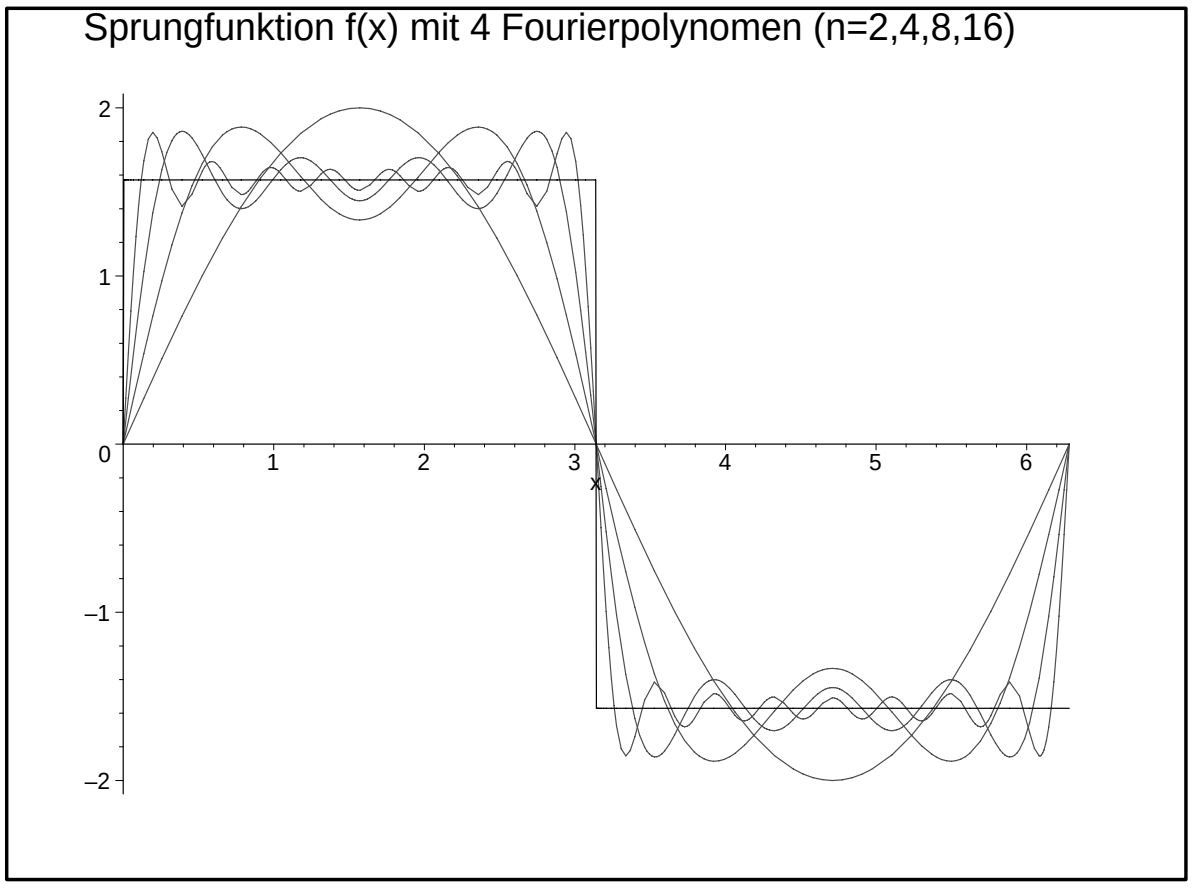
```
> f:=unapply(f,x):
> a:=proc(k,f)
    local t; options remember;
    evalf(int(f(t)*cos(k*t),t=0..2*Pi)/Pi);
  end:
> b:=proc(k,f)
    local t; options remember;
    evalf(int(f(t)*sin(k*t),t=0..2*Pi)/Pi);
  end:

> fourierpolynom:=proc(n,f,x)
    local k;
    a(0,f)*0.5+sum('a(k,f)*cos(k*x)+b(k,f)*sin(k*x)',k=1..n);
  end:
```

```
> seq(a(k,f),k=0..16);
  seq(b(k,f),k=1..16);
```

$$0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0$$

$$2., 0, .6666666667, 0, .4000000000, 0, .2857142857, 0, .2222222222, 0, .1818181818, 0, .1538461538, 0, .1333333333, 0$$

```
> plot([f(x),seq(fourierpolynom(2^n,f,x),n=1..4)],x=0..2*Pi);
  title='Sprungfunktion f(x) mit 4 Fourierpolynomen (n=2,4,8,16)',
  color=[black,red,red,red,red]);
```



Das stark oszillierende Verhalten der Fourierpolynome an den Sprungstellen der zu entwickelnden Funktion wird Gibbs'sches Phänomen oder Effekt genannt.

Die ersten beiden Beispiele sind implementiert im Arbeitsblatt *exam2.mws*, die anderen in *reihe2.mws*.

4 Erste einfache Animationen

Aus dem breiten Spektrum der Möglichkeiten von Animationen werden wir zunächst einige einfache demonstrieren.
Der typische Aufruf für die Animation mit einer Funktion ist

```
animate(f(x,t),x=a..b,t=c..d);
```

wobei $f(x,t)$ eine reelle Funktion ist. Diese wird als Funktion von x im Intervall $[a,b]$ dargestellt, während die Koordinate t wie ein Parameter wirkt. Diesen bezeichnet man auch als Bildkoordinate, denn für die gleichabständigen Stützstellen

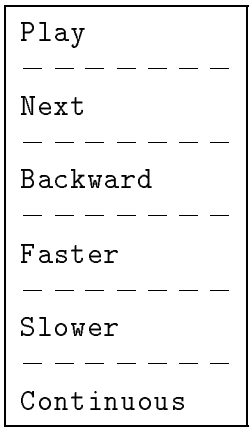
$$t_j = c + (j-1)\frac{d-c}{N-1}, \quad j = 1,2...,N, \quad N > 1,$$

werden fortlaufend die Bilder oder Frames $f(x,t_j)$ erzeugt und für das Animationsmenü gespeichert. Der Standardwert für die Bildanzahl ist $N = 16$. Mittels der Option `frames=...` kann er verändert werden. Weitere Optionen wie

`numpoints, coords, color, view, thickness, axes, scaling, linestyle, ...`

können analog zum Kommando des Funktionsplots verwendet werden.

Auch 2D- und 3D-Plots, Prozeduren, parametrische Funktionen, Listen, Punkte und ähnliches können in Animationen einbezogen werden.
Der Aufruf des Animationskommandos erzeugt eine PLOT Datenstruktur, die anschließend ausgegeben werden kann. Dazu ist dann das Erstbild anzuklicken sowie der Menüpunkt *Animation* oder mittels rechter Maustaste im neu geöffneten Untermenü mit *Copy, Style, Axes, Projection, Animation* der letzte Balken zu betätigen, um eine der folgenden Aktionen auszulösen:



Die Menüpunkte sind selbsterklärend.
Es ist auch möglich, eine Animation in Form einer Bildfolge unter Verwendung der Option `insequence` und der Funktion `display` durchzuführen. Darauf kommen wir später zurück.

Es gibt natürlich die Möglichkeit der Darstellung der einzelnen Bilder oder ihrer Folge.

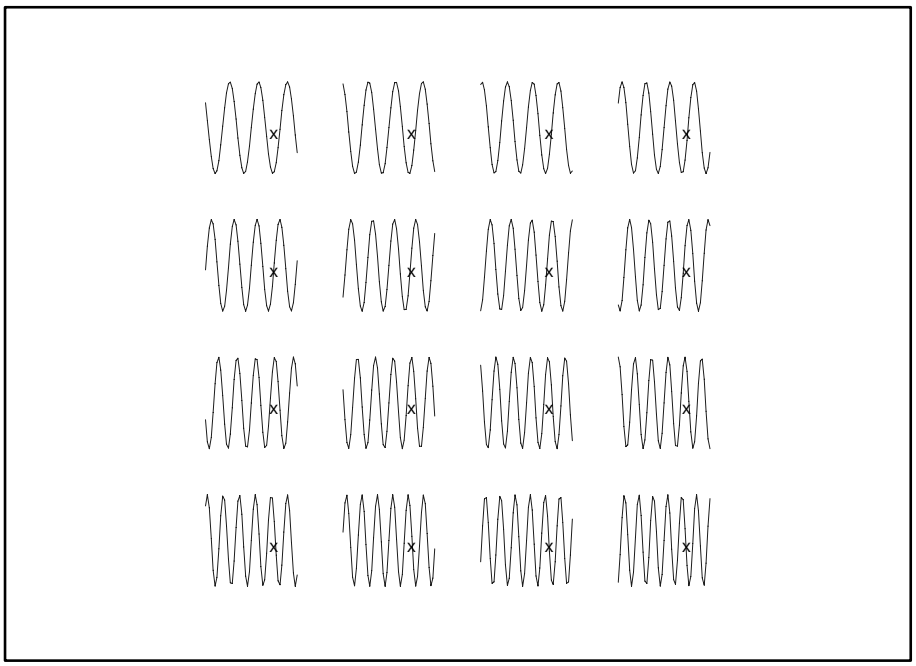
Die zu ladenden Maple-Pakete sind `with(plots)` oder `with(plots,animate)` . Man kann auch das Kommando `plots[animate]` verwenden.

Einige Beispiele

Zunächst wollen wir uns die 16 Frames einer Animation als Tableau betrachten.

Datei: ani1_.mws

```
> restart:
  with(plots):  # with(plot,animate):
  with(plottools):
> setoptions(scaling=constrained,axes=none):

Sinus mit wachsender Frequenz
> animate(sin(x*t),x=-10..10,t=1..2,color=blue,thickness=2):
  plots[display](%);
```



Dann definieren wir 12 Funktionen mit Animation, wobei wir später nebeneinander nur das erste und letzte Frame ausgeben werden.
Aus der PLOT Datenstruktur für die Animation kann man mittels des Kommandos `op` beliebige Unterstrukturen herauslesen, so auch das letzte Frame, und dann plotten.

```
Sinus mit wachsender Frequenz
> animate(sin(x*t),x=-10..10,t=1..2,color=blue,thickness=2);
  plots[display](op(16,op(1,%)),thickness=2);

Positiver Teil vom Sinus
> animate(sin(5*x*t),x=-4..4,t=0..1,view=0..1,
    numpoints=200,frames=30,
    color=magenta,thickness=2);
  plots[display](op(30,op(1,%)),view=0..1,thickness=2);
```

Horizontale Strecke nach oben bewegen

```
> animate([x,t,x=-4..4],t=1..4,color=red,thickness=2);
  plots[display](op(16,op(1,%)),thickness=2);
```

Vertikale Strecke nach rechts bewegen

```
> animate([t,x,x=-4..4],t=1..4,color=green,thickness=2);
  plots[display](op(16,op(1,%)),thickness=2);
```

Um 90 Grad gedrehter Sinus

```
> animate([sin(x*t),x,x=-4..4],t=1..4,frames=20,
    numpoints=100,color=violet,thickness=2);
  plots[display](op(20,op(1,%)),thickness=2);
```

Rosette

```
> animate([sin(x*t),x,x=-4..4],t=1..4,coords=polar,
    frames=20,numpoints=200,color=navy,thickness=2);
  plots[display](op(16,op(1,%)),thickness=2);
```

Bild mit Verkleinerungen

```
> animate([sin(x*t),x,x=-4..4],t=1..5,coords=rose,
    frames=20,numpoints=100,color=maroon,thickness=2);
  plots[display](op(20,op(1,%)),thickness=2);
```

Zunehmende konzentrische Kreise

```
> animate([u*sin(t),u*cos(t),t=-Pi..Pi],u=1..8,
    view=[-8..8,-8..8],color=brown,thickness=2);
  plots[display](op(16,op(1,%)),thickness=2);
```

Wachsende Spirale

```
> animate([u*t,t,t=1..8*Pi],u=1..4,coords=polar,
    frames=60,numpoints=100,color=red,thickness=2);
  plots[display](op(60,op(1,%)),thickness=2);
```

Bewegung von 2 Funktionen

```
> animate({x-x^3/u,sin(u*x)},x=0..Pi/2,u=1..16,
    color=blue,thickness=2);
  plots[display](op(16,op(1,%)),thickness=2);
```

Binomialkoeffizient

```
> animate(binomial,1..4,1..6,color=black,thickness=2);
  plots[display](op(16,op(1,%)),thickness=2);
```

Ahornblatt, Maple-Signum

```
> s := t -> 100/(100+(t-Pi/2)^8):
  r := t -> s(t)*(2-sin(7*t)-cos(30*t)/2):
  animate([u*r(t)/2,t,t=-Pi/2..3/2*Pi],u=1..2,
    numpoints=200,coords=polar,axes=none,color=green,thickness=2);
  plots[display](op(16,op(1,%)),thickness=2);
```

Bei *.mws –> Export As –> LaTeX werden die Plots vor ihrer Animation als *.eps Files im Format *landscape* und *s/w* ausgegeben.
Die Ausgabe nach Animation der Plots erzeugt dieselben *.eps Files, also mit dem ersten Frame und nicht etwa den "animierten".

12 animierte Funktionen mit dem ersten und letzten Frame

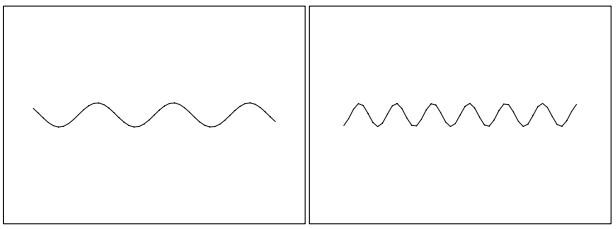


Abb. 1

ani1_02.eps, ani1_03.eps
Sinus mit wachsender Frequenz

```
animate(sin(x*t),x=-10..10,t=1..2,
   color=blue,thickness=2);
```

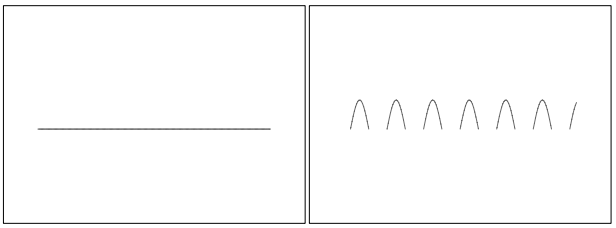


Abb. 2

ani1_04.eps, ani1_05.eps
Positiver Teil vom Sinus

```
animate(sin(5*x*t),x=-4..4,t=0..1,
   view=0..1,numpoints=200,frames=30,
   color=magenta,thickness=2);
```

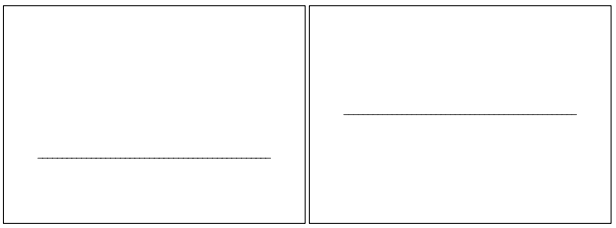


Abb. 3

ani1_06.eps, ani1_07.eps
Horizontale Strecke nach oben bewegen

```
animate([x,t,x=-4..4],t=1..4,
   color=red,thickness=2);
```

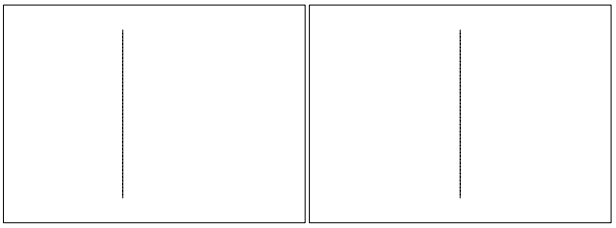


Abb. 4

ani1_08.eps, ani1_09.eps
Vertikale Strecke nach rechts bewegen

```
animate([t,x,x=-4..4],t=1..4,
   color=green,thickness=2);
```

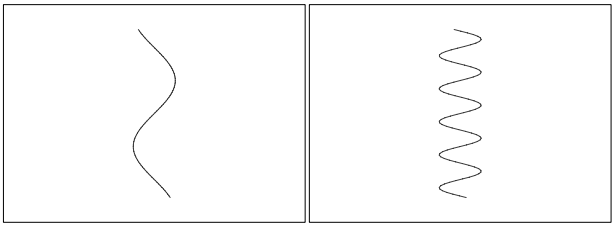


Abb. 5

ani1_10.eps, ani1_11.eps
Um 90 Grad gedrehte Sinus-Kurve

```
animate([sin(x*t),x,x=-4..4],t=1..4,
   frames=20,numpoints=100,
   color=violet,thickness=2);
```

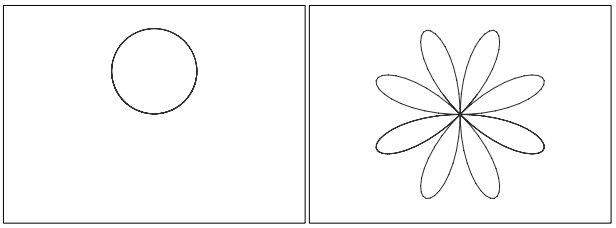


Abb. 6

ani1_12.eps, ani1_13.eps
Rosette

```
animate([sin(x*t),x,x=-4..4],t=1..4,
   coords=polar,frames=20,numpoints=200,
   color=navy,thickness=2);
```

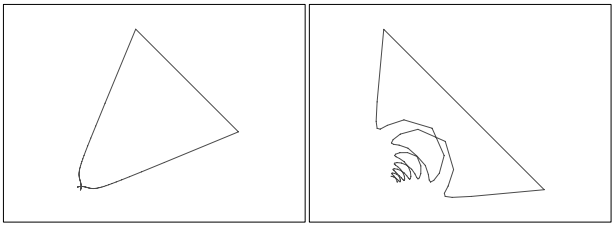


Abb. 7

ani1_14.eps, ani1_15.eps
Bild mit Verkleinerungen

```
animate([sin(x*t),x,x=-4..4],t=1..5,
  coords=rose,frames=20,numpoints=100,
  color=maroon,thickness=2);
```

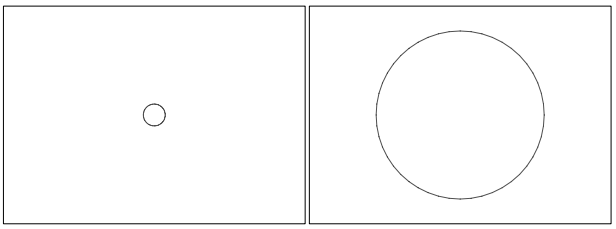


Abb. 8

ani1_16.eps, ani1_17.eps
Zunehmende konzentrische Kreise

```
animate([u*sin(t),u*cos(t),t=-Pi..Pi],
  u=1..8,view=[-8..8,-8..8],
  color=brown,thickness=2);
```

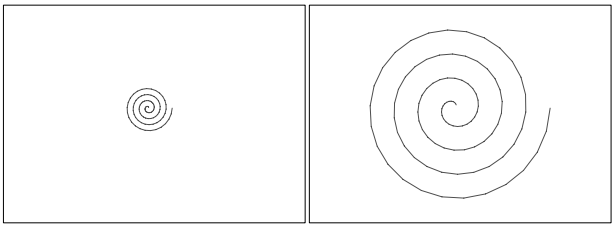


Abb. 9

ani1_18.eps, ani1_19.eps
Wachsende Spirale

```
animate([u*t,t,t=1..8*Pi],u=1..4,
  coords=polar,frames=60,numpoints=100,
  color=red,thickness=2);
```

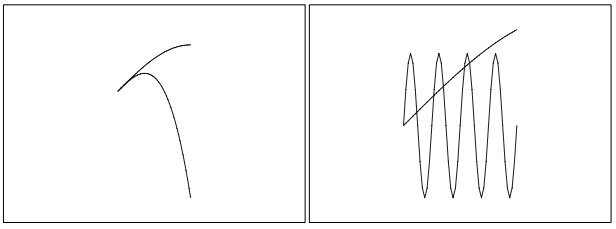


Abb. 10

ani1_20.eps, ani1_21.eps
Bewegung von 2 Funktionen

```
animate({x-x^ 3/u,sin(u*x)},x=0..Pi/2,
  u=1..16,color=blue,thickness=2);
```

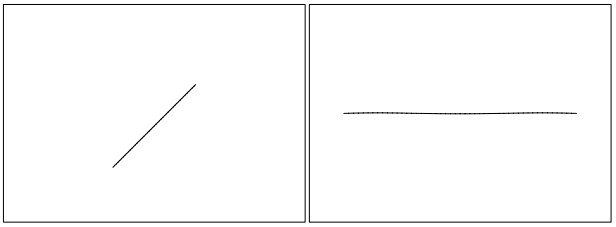


Abb. 11

ani1_22.eps, ani1_23.eps
Binomialkoeffizient

```
animate(binomial,1..4,1..6,
  color=black,thickness=2);
```

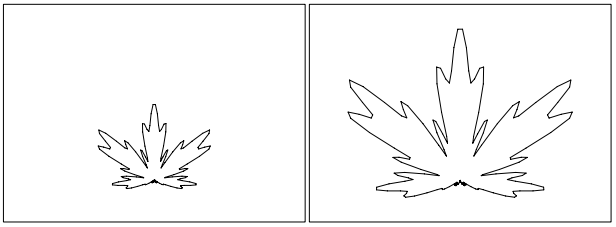


Abb. 12

ani1_24.eps, ani1_25.eps
Ahornblatt, Maple-Signum

```
s := t -> 100/(100+(t-Pi/2)^ 8):
r := t -> s(t)*(2-sin(7*t)-cos(30*t)/2):
animate([u*r(t)/2,t,t=-Pi/2..3/2*Pi],
  u=1..2,numpoints=200,coords=polar,
  axes=none,color=green,thickness=2);
```

5 Export von GIF-Files

Den Export von Grafiken in den Formaten *ps* und *gif* haben wir bereits erläutert, und er soll mit dem Ziel weiterer Betrachtungen durch folgendes Beispiel noch einmal demonstriert werden.

Datei: ani2_.mws

```
> restart:
  with(plots):
  with(plottools):

  # Arbeitsverzeichnis
  pfad := `D:/Neundorf/Maple2/`;
  # Graphik output files
  name1 := `ani2_01.ps`;
  name2 := `ani2_02.gif`;
  name3 := `ani2_03.gif`;
  name4 := `ani2_04.gif`;

  datei1 := cat(pfad,name1);
  datei2 := cat(pfad,name2);
  datei3 := cat(pfad,name3);
  datei4 := cat(pfad,name4);
```

1. Plot

```
> pl1:=plot3d(sin(x)*exp(y), x=0..10,y=2..5,
    style=patchnogrid, grid=[40,40]):
  plots[display](pl1);
```

PS File: color and portrait

```
> pl11:=plot3d(sin(x)*exp(y), x=0..10,y=2..5,
    style=patch, grid=[40,40], axes=boxed,
    title=`f(x,y)=sin(x)*exp(y), x=0..10,y=2..5`):
  plots[display](pl11);
  interface(plotdevice=ps,
      plotoutput=datei1,
      plotoptions=`color,portrait,width=640,height=480`);
  plots[display](pl11);
  interface(plotdevice=win);
```

GIF File

```
> interface(plotdevice=gif,
      plotoutput=datei2,
      plotoptions=`width=640,height=480`);
  plots[display](pl1);
  interface(plotdevice=win);
```

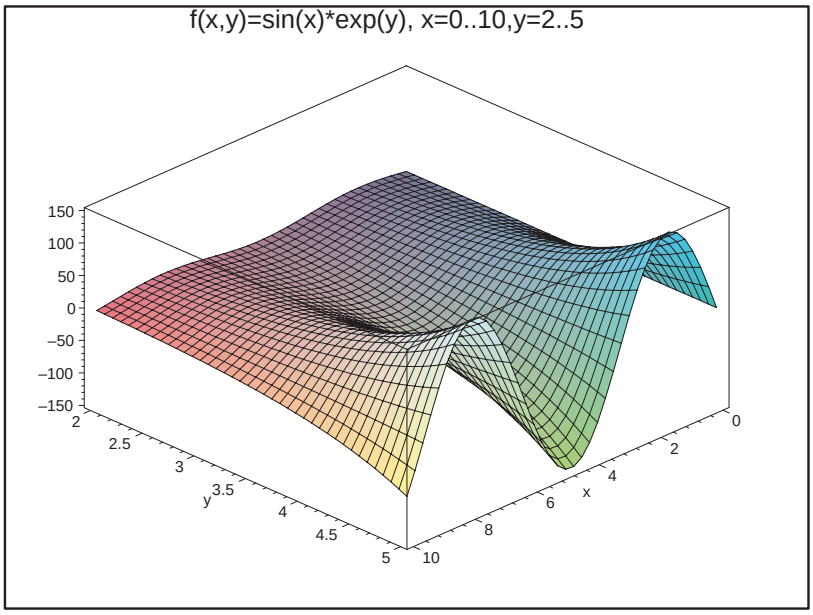


Abb. ani2_01.ps, $f(x,y) = \sin(x)e^y$, $x = 0..10$, $y = 2..5$,
style=patch, grid=[40,40], axed=boxed,
color, portrait, width=640, height=480

Beim *gif*-File wird allein die farbige Fläche (Plot *pl1*) ohne jegliche Zusätze ausgegeben, wobei auch noch der Boxrahmen fehlt.
Die eigentliche Grafik erscheint dabei auf dem weißen Hintergrund der Box. Damit ist das Bild nicht transparent.

5.1 Transparente GIF-Files

Nur für das *gif*-Format können beim Export von Grafik die Varianten eines nicht transparenten oder transparenten Bildes ausgewählt werden. Nichttransparenz ist Standard, und man kann dies in den Plotoptionen von `interface` durch `transparent=false` notieren. Wünscht man ein transparentes Bild, so muss man dort `transparent=true` angeben.
Am besten ist die Situation zu erkennen, wenn man beide Varianten als *gif*-Files auf eine Seite seiner Homepage stellt, die einen farbigen Hintergrund hat. Das nicht transparente Bild liegt in einer weißen Box, während das transparente ganz einfach vor dem Hintergrund steht. Beim Druck der beiden Bilder auf farbigem Papier mittels MPP ist jedoch kein Unterschied, d.h. das nicht transparente Bild erhält keine weiße Box.

Eine Kontrolle des Zeichenfonts durch den Treiber ist nicht möglich.
In den Plotoptionen wird die Zahl der Farben in der Palette eingestellt durch die Angabe `color=...` (*default*=256). Dort kann man ebenfalls die Höhe und Breite des Bildbereichs als `height=...,width=...` in Pixel angeben. Die Standardwerte des Bildbereichs sind 512×512 Pixel. Jedoch sollte man überprüfen, welcher Maßstab wirklich zutrifft. Erfahrungen im Umgang mit den Dimensionsangaben und die Ansichten der Bilder lassen eher die Vermutung zu, dass es sich bei die Anzahl um *points per inch (pt)* handelt.

Bei der Umrechnung zwischen den Maßstäben gemäß

$$1'' = 1\,in = 25.4\,mm = 72.27\,pt = 83\,pix$$

bedeutet das folgende Größenordnungen für die *gif*-Bilder

Pixel	mm, falls pix	mm, falls pt
512×512	157×157	180×180
640×480	196×147	225×165
200×200	61×61	70×70

GIF File: Transparent, verschiedene Größen

```
> interface(plotdevice=gif,
      plotoutput=datei3,
      plotoptions=`width=640,height=480,transparent=true`);
  plots[display](pl1);
  interface(plotdevice=win);

  interface(plotdevice=gif,
      plotoutput=datei4,
      plotoptions=`width=200,height=200,transparent=true`);
  plots[display](pl1);
  interface(plotdevice=win);
```

Abb. ani2_04.eps generiert aus ani2_04.gif mittels MPP,
$f(x,y) = \sin(x)e^y$, $x = 0..10$, $y = 2..5$, style=patchnogrid,
width=200, height=200, transparent=true

Man erkennt durch die Kennzeichnung der Ecken die Dimension des Bildes (Box) von $200\times200pt$=$70\times70mm$ mit dem kleinen darin liegenden Graphen.
Schiebt man die Box zu weit hoch, dann weicht der darüber stehende Text nach rechts aus. Darunter befindlichen Text kann man nach Bedarf ein kleines Stück in die Box hineinschieben.

Weitere Plots

```
> datei5 := cat(pfad,‘ani2_05.ps‘);
  datei6 := cat(pfad,‘ani2_06.gif‘);
  datei7 := cat(pfad,‘ani2_07.ps‘);
  datei8 := cat(pfad,‘ani2_08.gif‘);
  datei9 := cat(pfad,‘ani2_09.ps‘);
  datei10 := cat(pfad,‘ani2_10.ps‘);
  datei11 := cat(pfad,‘ani2_11.gif‘);
  datei12 := cat(pfad,‘ani2_12.gif‘);
```

2.Plot

```
> f:= (x,y)-> -Re(arctan(x+I*y)):
  pl2:=plot3d(f(x,y), x=-3..3,y=-3..3,
    color=x*y, style=patchnogrid, grid=[40,40]):
  plots[display](pl2);
```

PS File: color and portrait

```
> pl21:=plot3d(f(x,y), x=-3..3,y=-3..3,
    color=x*y, style=patch, grid=[40,40], axes=boxed,
    title=‘f(x,y)=-Re(arctan(x+I*y)), x=-3..3,y=-3..3‘):
  plots[display](pl21);
  interface(plotdevice=ps,
      plotoutput=datei5,
      plotoptions=‘color,portrait,width=640,height=480‘);
  plots[display](pl21);
  interface(plotdevice=win);
```

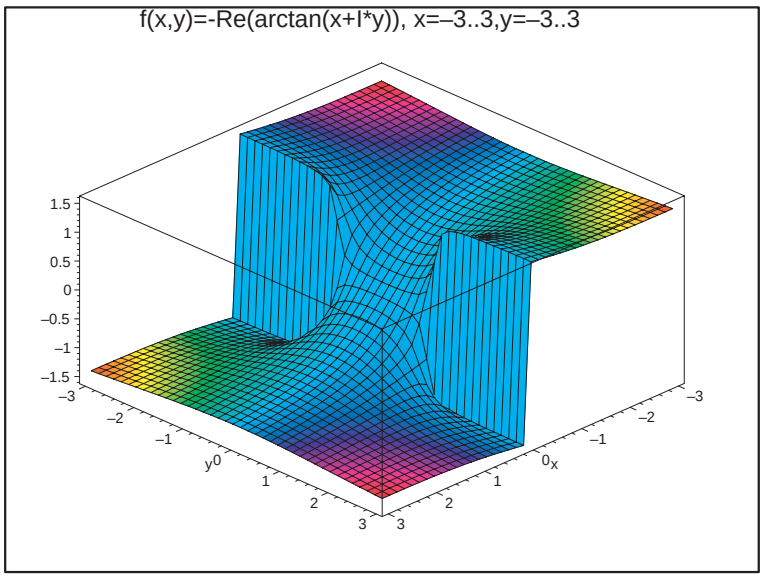


Abb. ani2_05.ps
$f(x,y) = -\Re(\arctan(x + i * y))$,
$x = -3..3$, $y = -3..3$,
color=x∗y, style=patch,
grid=[40,40], axes=boxed,
color, portrait,
width=640, height=480

Neben den konstanten Farben kann man eigene Farben mittels der RGB- oder HUE-Farbpalette definieren.

```
> macro(skyblue = COLOR(RGB, 0.1960, 0.6000,0.8000)):
  plot3d(f(x,y), x=-3..3,y=-3..3,
    color=skyblue, style=patch, grid=[40,40], axes=boxed,
    title=‘f(x,y)=-Re(arctan(x+I*y)), x=-3..3,y=-3..3‘);
```

GIF Transparent File

```
> interface(plotdevice=gif,
      plotoutput=datei6,
      plotoptions=‘width=200,height=200,
      transparent=true‘);
  plots[display](pl2);
  interface(plotdevice=win);
```

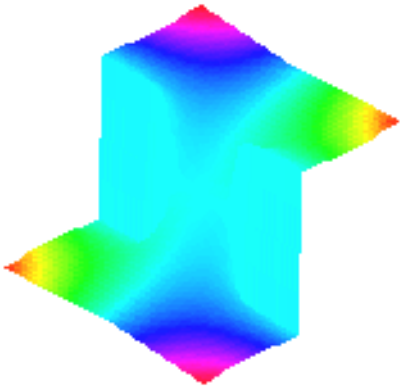


Abb. ani2_06.eps
generiert aus ani2_06.gif mittels MPP,
$f(x,y) = \sin(x)e^y$, $x = 0..10$, $y = 2..5$,
color=x*y, style=patchnogrid, grid=[40,40],
width=200, height=200, transparent=true

3. Plot

```
> f:= (x,y)-> min(3,abs(exp(1.0/(x+I*y)))):
  pl3:=plot3d(f(x,y), x=-2..2,y=-2..2,
    # color=...,
    style=patchnogrid, grid=[40,40],
    orientation=[225,45]):
  plots[display](pl3);
```

PS File: color and portrait

```
> pl31:=plot3d(f(x,y), x=-2..2,y=-2..2,
    # color=...,
    style=patch, grid=[40,40], axes=boxed,
    orientation=[225,45],
    title=‘f(x,y)=min(3,abs(exp(1.0/(x+I*y)))), x=-2..2,y=-2..2‘):
  plots[display](pl31);

  interface(plotdevice=ps,
      plotoutput=datei7,
      plotoptions=‘color,portrait,width=640,height=480‘);
  plots[display](pl31);
  interface(plotdevice=win);
```

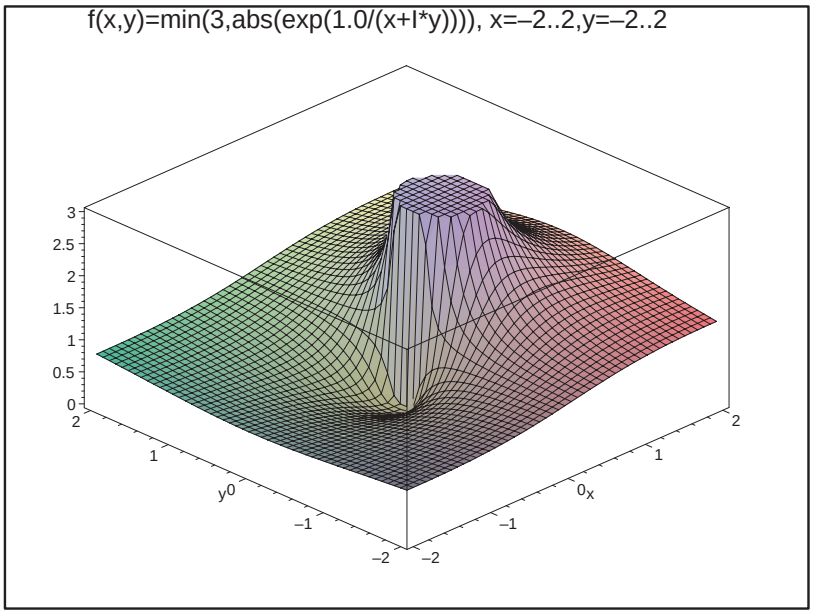


Abb. ani2_07.ps

$f(x,y) = \min(3, \mathrm{abs}(\exp(1/(x + i * y)))),\ x = -2..2,\ y = -2..2,$
style=patch, grid=[40,40], axes=boxed, orientation=[225,45],
color, portrait, width=640, height=480

GIF Transparent File

```
> interface(plotdevice=gif,
      plotoutput=datei8,
      plotoptions=‘width=200,height=200,
      transparent=true‘);
  plots[display](pl3);
  interface(plotdevice=win);
```

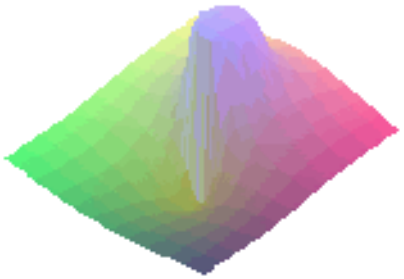


Abb. ani2_08.eps
generiert aus ani2_08.gif mittels MPP,
$f(x,y) = \min(3, \mathrm{abs}(\exp(1/(x + i * y)))),$
$x = -2..2,\ y = -2..2,$
style=patchnogrid, grid=[40,40],
orientation=[225,45],
width=200, height=200,
transparent=true

5.2 Animationssequenzen als transparente GIF-Files

Neben der gewöhnlichen Animation weisen wir die Folge der Frames einer Variablen zu, um dann einzelne Frames oder die Framefolge in verschiedenen Darstellungen zu plotten bzw. zu speichern.

4. Plot Animation: Ahornblatt

```
> setoptions(scaling=constrained,axes=none):
  X:=cos(x)*(1+k*sin(x))*(1+k*0.3*cos(8*x))*(1+k*0.1*cos(24*x)):
  Y:=sin(x)*(1+k*sin(x))*(1+k*0.3*cos(8*x))*(1+k*0.1*cos(24*x)):
> animate([X,Y,x=0..2*Pi], k=0..1, color=green,thickness=3);
```

Animation als Variable

```
> signe:=animate([X,Y,x=0..2*Pi], k=0..1, color=red,thickness=3):
```

Kontrolle einzelner Blätter

```
> plots[display](op(1,op(1,signe)));  # 1.Frame
  plots[display](op(16,op(1,signe))); # 16.Frame
```

Darstellung der Framesequenzen

Alle Blätter im Bild übereinander

```
> plots[display](seq(op(k,op(1,signe)),k=1..16));
```

PS File: color and portrait

```
> interface(plotdevice=ps,
      plotoutput=datei9,
      plotoptions=‘color,portrait,width=640,height=480‘);
  plots[display](seq(op(k,op(1,signe)),k=1..16));
  interface(plotdevice=win);
```

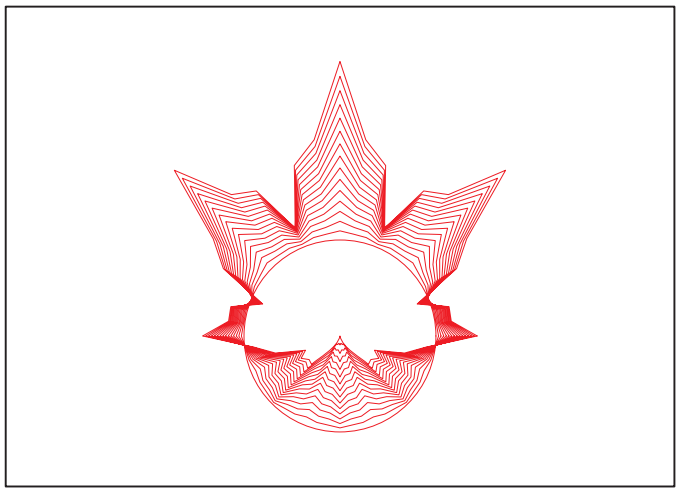


Abb. ani2_09.ps

16 Frames der Animation zur Entstehung des Ahornblatts übereinander, color=red, color, portrait, width=640, height=480

Alle Blätter im Tableau nebeneinander, PS File: color and portrait

```
> plots[display](signe);
> interface(plotdevice=ps,
      plotoutput=datei10,
      plotoptions='color,portrait,width=640,height=480');
  plots[display](signe);
```

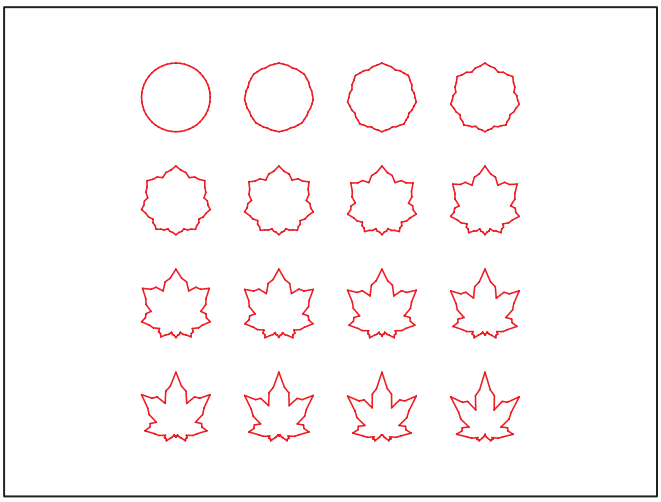


Abb. ani2_10.ps

16 Frames der Animation zur Entstehung des Ahornblatts nebeneinander, color=red, color, portrait, width=640, height=480.
Dabei gehen die Unterschiede zwischen den Größen der Bilder verloren.

GIF bzw. GIF Transparent File

```
> interface(plotdevice=gif,
      # plotoutput=datei11, mit transparent=false
      plotoutput=datei12,
      plotoptions='width=200,height=200,transparent=true');
  plots[display](signe);
  interface(plotdevice=win);
```

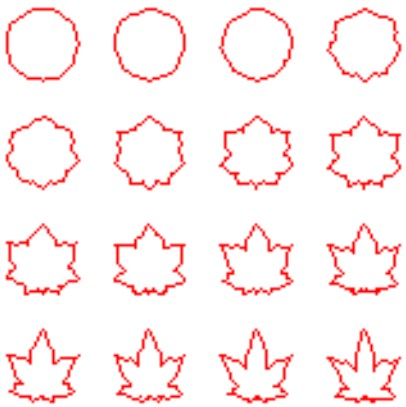


Abb. ani2_12.eps
generiert aus ani2_12.gif mittels MPP,
16 Frames der Animation
zur Entstehung des Ahornblatts,
color=red, width=200, height=200,
transparent=true

6 Export animierter GIF-Files

Unser Ziel soll sein, dass auf den Seiten einer Hompage im Internet in Erkennungszeichen, Symbolen, Ikonen oder Labels einfache Animationen ablaufen sollen, d.h. dass dort sich periodisch wiederholende Bildfolgen dargestellt werden. Diese sollen nicht nur ein Blickfang für den Betrachter sein, sondern auch in irgendeiner Art Hinweise auf die an dieser Stelle behandelte Themen oder Querverweise (Links) beinhalten.

Dazu dienen animierte transparente Files. Maple unterstützt beim Export der Animation nur den Filetyp *gif*.

Allgemeine Hinweise zu Implementierung

- Die Transparenz der Figur macht sich bei Auswahl entsprechender Farben vor einem beliebigen Hintergrund günstiger und eindrucksvoller als die Animation bei Nichttransparenz in einer weißen Box. Dazu erfolgen noch Größenangaben. Titel über die Figur sind nicht sinnvoll, da bei Export als *gif*-File diese nicht übertragen werden. Ähnlich werden auch andere Optionen, die bei Vorabbetrachtung auf dem Bildschirm wirken, nur mit der *default*-Einstellung "transformiert" (z.B. `thickness`).

- Also werden wir vorzugsweise **animierte transparente GIF-Files** in den Standardeinstellungen *color, portrait* produzieren und exportieren. Sie werden entweder neu angelegt, oder falls schon eine unter dem Namen besteht, überschrieben.

- Natürlich können wir in diesem Skript von den Animationen oft nur ausgewählte Bilder (Frames), Übersichten/Tableaus mit einigen oder auch allen Bildern oder die "Überlagerung" solcher Bilder demonstrieren, um einen ersten (bescheidenen) Eindruck vom Ablauf der Animation zu erhalten. Gerade bei zu viel Überlagerung besteht die Gefahr, dass man zum Schluss die Übersicht verliert. Solche Bildsequenzen haben wir in den bisherigen Kapiteln an Beispielen schon gezeigt.

- Den angenehmen Teil der Animation überlassen wir somit dem Leser durch die beigefügten Beispiele und Tests in den Arbeitsblättern. Auch wenn dabei so manche besonderen Effekte einen noch überraschen, so haben diese meist einen plausiblen Grund und sind nach einiger Überlegung auch zu erklären.

- Im weiteren können wir nicht jedes Detail der Erzeugung einer Animation erläutern. Dabei werden u.a. die Kommandos `plod3d, plots[tubeplot], animat3d` benutzt, oder z.B. Systeme von gewöhnlichen Differentialgleichungen (Anfangswertaufgaben) numerisch und grafisch gelöst (z.B. mit dem Kommando `DEplot3d`).

- Es ist also auch eine Frage der Rechengeschwindigkeit und Speicherkapazität, kompliziertere Animationen vorzubereiten und effektvoll mit dem Animationsmenü oder auf der Homepage zu demonstrieren.

- Anwendungen findet der Leser auf der Homepage

  http://imath.mathematik.tu-ilmenau.de/~neundorf/index_de.html

  unter **Navigator → Publications → Computeralgebra → ...**

6.1 Die Display-Option insequence

Das *display*-Kommando geht von der Definition von Plots oder Animationen aus, um sie dann einzeln oder gemeinsam darzustellen.
Die Notation erfolgt beispielhaft i.a. in folgender Form.

```
A:=plot( ): # plot3d( ):  animate( ):  animat3d( ):
B:=plot( ): # plot3d( ):  animate( ):  animat3d( ):

display(A,B,options);
display([A,B],options);
display({A,B},options);

display3d(A,B,options);
display3d([A,B],options);
display3d({A,B},options);
```

Sobald eine der Plot-Variablen eine Animation darstellt, kann man mit dem *display*-Kommando das Animationsmenü aufrufen.

Etwas aufwendiger, aber damit auch reicher an Möglichkeiten einer Verarbeitung, ist die Definition einer Liste mit einer gegebenen Anzahl von Elementen. Um diese einer Animation zu unterwerfen, braucht man die Option `insequence`. Ihre Standardeinstellung ist `insequence=false`. Aber im Fall `insequence=true` kann die Folge der Listenelemente=Bilder als Framesequenz einer Animation dienen. Weiterhin zeigt sich, dass anstelle von `display` das allgemeinere Kommando `plots[display]` bevorzugt wird.
Auch hier einige Beipiele für die Notation.

```
# Definition der Liste
A:=[L]:  # [seq(d[k],k=1..n)]:

# keine Animation, Bilder einzeln oder uebereinander
display(A[1],options);
display(A,options);
plots[display](A,options);

# 3 Varianten der Animation
display(A,insequence=true,options);
plots[display](A,insequence=true,options);
p:=display(A,insequence=true,options):
p;
# Nicht moeglich:
#     plots[display](p);
#     display(p);
```

Informationen in der Maple-Hilfe zu plots[display]

```
Function         : plots[display] - display a list of plot structures
Calling Sequence : display([L], insequence=true, options)

Parameters       :
 L               - a set, a list, or an array of PLOT structures
                   to be displayed
 insequence=true - (optional) display the list of plots in sequence,
                   i.e. animate them frame by frame., default is false.
 options         - (optional) standard options accompanying PLOT or PLOT3D
                   structures

Description      :
If L is a list or a set and insequence is false, the display function
creates a union of the given plot structures.
If insequence is true, then the list of plots are animated in the sequence
given. If plots are given in a set then the order cannot be guaranteed.
If L is an array, then display arranges the plots corresponding to the
array. Individual orientation and projection are preserved.

Display is not smart about things like collisions with label names.
If two plot structures have different labeling, display chooses one of them.
Similarly, in the case of multiple titles, only one of them is used.
The display function does not handle mixed infinit y and non-infinity plots.
To see what an infinity plot is see plot[infinity].

Animations produced by animate or animate3d can also be displayed.
If insequence is true, then the animations are displayed in sequence,
otherwise the corresponding frames in each animation are merged together
and then displayed in sequence.

Animation can be displayed over background plots by mixing them
in a set or list and setting insequence to false.
Remaining arguments are interpreted as plot options which are specified
in help for plot,options and plot3d,options. These global specifications
override the ones defined locally in the individual plot structures.

-------------------------------------------------------------------------

Function         : plots[display3d] - display a set of 3D plot structures
Calling Sequence : display3d(L)
                   display3d(L, options)
Parameters       : L - a set or list of PLOT3D structures to be displayed
Description      : The function display3d is an alias for plots[display].
```

6.2 Export als animierte GIF-Files

Sei *pictures* eine Liste mit einer Bildfolge. Dann notiert man im Zusammenhang mit ihrem Export meist folgende Kommandofolge.

```
> restart:
  with(plots):
  with(plottools):  # with(plots,animate):
  with(DEtools):    # fuer DGL-Solver

  # Arbeitsverzeichnis
  pfad := `D:/Neundorf/Maple2/`;

  # Graphik output files
  name1 := `bild1.ps`;
  name2 := `bild2.gif`;
  datei1 := cat(pfad,name1);
  datei2 := cat(pfad,name2);
```

Definition der Framesequenz

```
> pictures:= ...
```

Kontrollanzeige der Animation

Meist erfolgt die Abbildung mit einer Überlagerung aller Bilder.
Dabei werden aber manchmal einige Optionen nicht berücksichtigt.

```
> plots[display](pictures);
```

Animation

```
> plots[display](pictures,insequence=true); # ev.  weitere Optionen
```

PS File: color, portrait

Keine Unterstützung der Animation, Transparenz ohne Effekt

```
> interface(plotdevice=ps,
      plotoutput=datei1,
      plotoptions=`color,portrait,width=200,height=200`);
  plots[display](pictures);
      # plots[display](pictures,insequence=true);
      # fuehrt zu Error
  interface(plotdevice=win);
```

Animiertes transparentes GIF File: color, portrait

```
> interface(plotdevice=gif,
      plotoutput=datei2,
      plotoptions=`width=200,height=200`);
  plots[display](pictures,insequence=true);
  interface(plotdevice=win);
```

7 Animierte transparente GIF-Files in Beispielen

Bei den folgenden Beispielen, die teilweise aus der Literatur oder der Maple-Hilfe stammen, werden die Kommandos zum Laden der Maple-Pakete nicht mehr notiert.
In der Datei *ani31_.mws* sind der Orbit und das ausgemalte Ahornblatt implementiert. Diese Beispiele werden ausführlich beschrieben. Die nächsten sind in den Arbeitsblättern *ani32_.mws, ani33_.mws, ani34_.mws, ani41_.mws, ani42_.mws, ani43_.mws* zu finden.

7.1 Orbit

Der Orbit ist ein Raumkurve, deren Parameterdarstellung

$$P(t) = (x(t), y(t), z(t)),\ t \in [0, T],$$

als Lösung eines Systems von drei linearen Differentialgleichungen 1.Ordnung mit Anfangsbedingungen $P(0) = (x_0, y_0, z_0)$ erhalten wird. Die Lösung selbst nimmt etwas Rechenzeit in Anspruch (sec...min).
Verändert man die Orientierung dieser Kurve sukzessiv, so erhält man eine gewünschte Bildsequenz für die Animation.

```
> pfad := `D:/Neundorf/Maple2/`;
  name1 := `ani31_01.ps`;
  name2 := `ani31_02.ps`;
  name3 := `ani31_03.ps`;
  name4 := `ani31_04.gif`;
  datei1 := cat(pfad,name1);
  datei2 := cat(pfad,name2);
  datei3 := cat(pfad,name3);
  datei4 := cat(pfad,name4);
```

Bild: Orbit in 12 Lagen

```
> ti:=time():  # Zeitmessung fuer DGL-Solver

> GRD:=0,30,60,90,120,150,180,210,240,270,300,330:
          # 12 Orientierungen

  curve:=[D(x)(t)=10*(y(t)-x(t)),
        D(y)(t)=28*x(t)-y(t)-x(t)*z(t),
        D(z)(t)=-8*z(t)/3+x(t)*y(t)]:

  picts:=[seq(DEplot3d(curve,[x,y,z], t=0..15,
        {[0,10,0,25]}, x=-20..20,y=-25..25,z=5..45,
        thickness=3, stepsize=0.01,
        scene=[x,y,z], linecolor=t-12,
        axes=none, orientation=[180,grd]),
        grd=GRD)]:

> Rechenzeit:=time()-ti, `sec`;
```

12 Orbits übereinander (3D-Ansicht, Orientierung nicht berücksichtigt)

```
> plots[display](picts);

> interface(plotdevice=ps,
      plotoutput=datei1,
      plotoptions='color,portrait,width=640,height=480');
  plots[display](picts);
  interface(plotdevice=win);
```

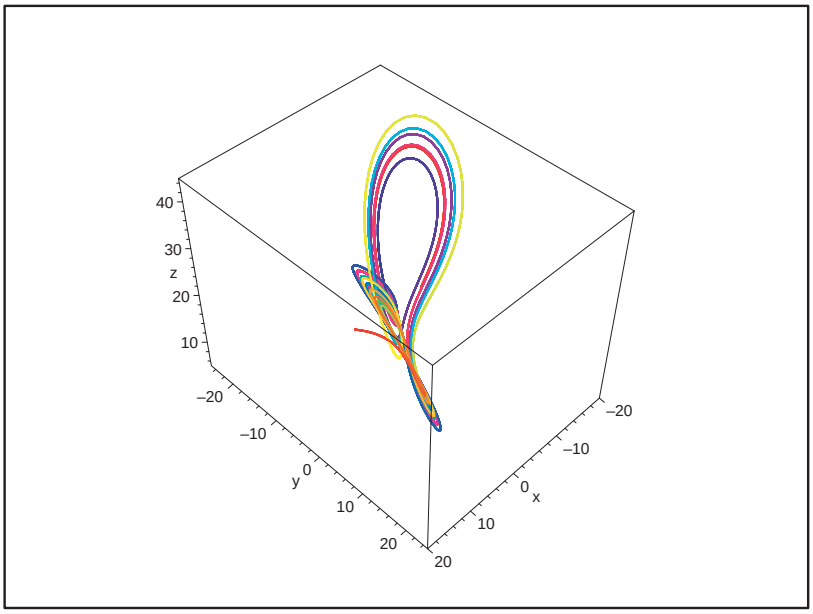


Abb. ani31_01.ps, Orbitlagen übereinander
(3D-Ansicht, Orientierung nicht berücksichtigt)

Einzelne Orbits zur Kontrolle

```
> plots[display](picts[1],axes=none);
> plots[display](picts[2],axes=none);
> plots[display](picts[3],axes=none);
> plots[display](picts[10],axes=none);
```

Ansicht aller 12 Orbitlagen als Tableau

```
> qq:=array(1..3,1..4,[]):
  for k from 1 to 4 do
    qq[1,k]:=plots[display](picts[k],axes=none):
    qq[2,k]:=plots[display](picts[4+k],axes=none):
    qq[3,k]:=plots[display](picts[8+k],axes=none):
  od:
  plots[display](qq);
> interface(plotdevice=ps,
      plotoutput=datei2,
      plotoptions='color,portrait,width=640,height=480');
  plots[display](qq);
  interface(plotdevice=win);
```

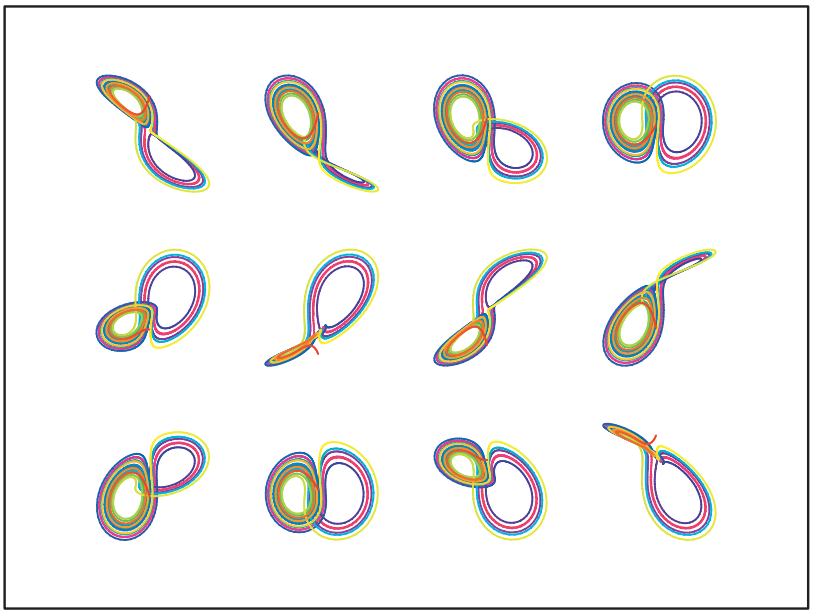


Abb. ani31_02.ps, 12 Orbitlagen als Tableau
$t = 0..15, x = -20..20, y = -25..25, z = 5..45,$
thickness=3, stepsize=0.01, scene=[x,y,z], linecolor=t-12,
axes=none, orientation=[180,grd], grd=0(30)330

PS File

```
> interface(plotdevice=ps,
      plotoutput=datei3,
      plotoptions='color,portrait,width=200,height=200');
  plots[display](picts, # insequence=true,
    style=patch, thickness=3);
  interface(plotdevice=win);
```

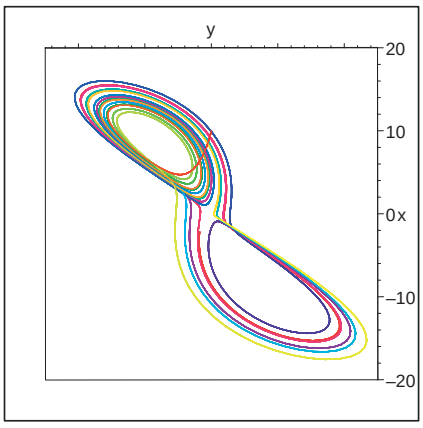


Abb. ani31_03.ps
1.Orbitlage,
style=patch, grd=0

Animation

```
> plots[display](picts,insequence=true,style=patch);
```

GIF Animated Transparent File

```
> interface(plotdevice=gif,
      plotoutput=datei4,
      plotoptions=‘width=200,height=200,
      transparent=true‘);
  plots[display](picts,insequence=true,
    style=patch,
    thickness=3);
  interface(plotdevice=win);
```

7.2 Ausgemaltes Ahornblatt

Das ausgemalte Ahornblatt [3] ist eine flache räumliche Figur, die mittels eines 3D-Plots definiert wird. Eine Folge von 10 verschiedenen Gittern führt auf die Listenstruktur und somit Framesequenz.

```
> pfad := ‘D:/Neundorf/Maple2/‘;
  name5 := ‘ani31_05.ps‘;
  name6 := ‘ani31_06.ps‘;
  name7 := ‘ani31_07.ps‘;
  name8 := ‘ani31_08.gif‘;
  datei5 := cat(pfad,name5);
  datei6 := cat(pfad,name6);
  datei7 := cat(pfad,name7);
  datei8 := cat(pfad,name8);
```

Bild: Ausgemaltes Ahornblatt in 10 Blättern

```
> setoptions3d(style=patchnogrid,scaling=constrained,projection=0.5):

> X:=cos(x)*cos(y)*(1+0.2*sin(y))*(1+sin(x))*(1+0.3*cos(8*x))*
        (1+0.08*cos(24*x)):
> Y:=sin(x)*cos(y)*(1+0.2*sin(y))*(1+sin(x))*(1+0.3*cos(8*x))*
        (1+0.08*cos(24*x)):
> Z:=0.02*sin(y)*(1-sin(y)):

> d:=k->plot3d([X,Y,Z],x=-Pi/2..3*Pi/2,y=-Pi/2..Pi/2,
      color=[sin(y*x),-cos(y)+sin(0.5*x-y),-cos(y-0.3)],
      ambientlight=[0.4,0.4,0.4], light=[75,40,0.8,0.7,0.3],
      orientation=[-90,0], grid=[5*k+1,5*k+1]):
  dk:=[seq(d(k),k=1..10)]:
```

Alle 10 Frames übereinander

```
> # Ahornblatt mit Spitze nach oben
  display3d(dk,orientation=[-90,0]);
> # Ahornblatt mit Spitze nach rechts unten
  plots[display](dk); # orientation=[45,45] ist Standard
> # Ahornblatt mit Spitze nach rechts
  display3d(dk,orientation=[0,0]);
```

```
> interface(plotdevice=ps,
      plotoutput=datei5,
      plotoptions='color,portrait,width=640,height=480');
  plots[display](dk); # Ahornblatt mit Spitze nach rechts unten
  interface(plotdevice=win);
```

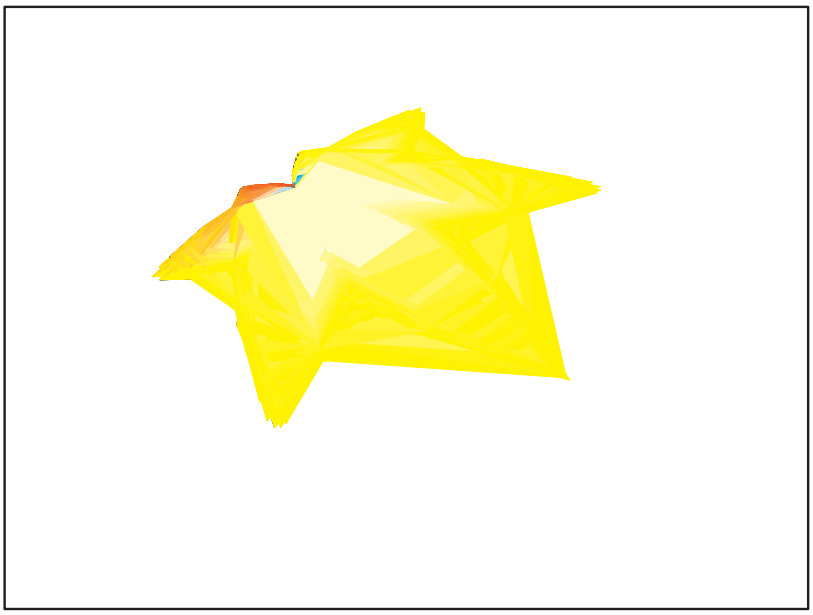


Abb. ani31_05.ps, Ahornblätter übereinander

Kontrolle des "Wachstums" der einzelnen Blätter, Spitze nach oben

```
> plots[display](dk[1],orientation=[-90,0],axes=normal);
  plots[display](dk[2],orientation=[-90,0],axes=normal);
  plots[display](dk[3],orientation=[-90,0],axes=normal);
  plots[display](dk[10],orientation=[-90,0],axes=normal);
```

Ansicht der 10 Blätter als Tableau

```
> pp:=array(1..2,1..5,[]):
  for k from 1 to 5 do
    pp[1,k]:=display3d(dk[k],orientation=[-90,0]):
    pp[2,k]:=display3d(dk[k+5],orientation=[-90,0]):
  od:
  plots[display](pp);
```

```
> interface(plotdevice=ps,
      plotoutput=datei6,
      plotoptions='color,portrait,width=640,height=480');
  plots[display](pp);
  interface(plotdevice=win);
```

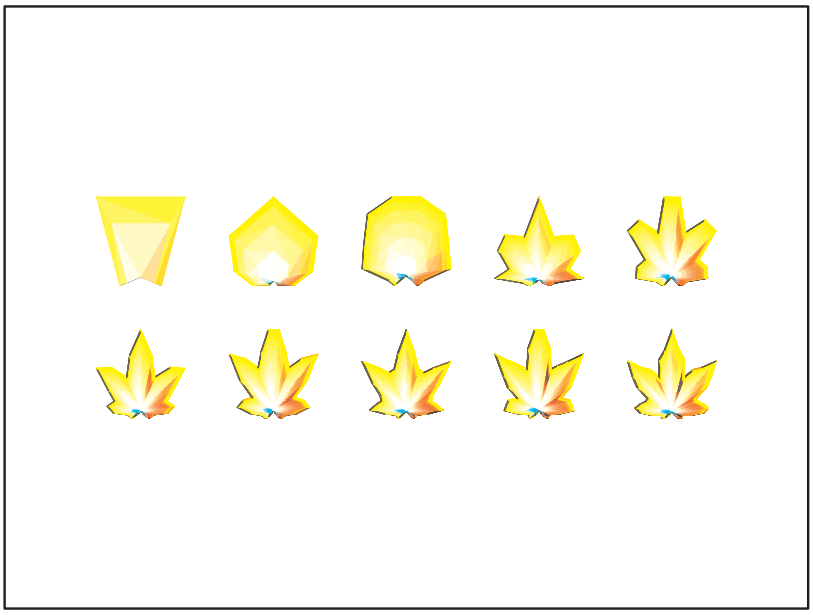


Abb. ani31_06.ps, 10 Ahornblätter als Tableau
color=[sin(y*x),-cos(y)+sin(0.5*x-y),-cos(y-0.3)],
ambientlight=[0.4,0.4,0.4], light=[75,40,0.8,0.7,0.3],
orientation=[-90,0], grid=[5*k+1,5*k+1]

PS File für Blätter übereinander

```
> interface(plotdevice=ps,
      plotoutput=datei7,
      plotoptions=‘color,portrait,width=200,height=200‘);
  plots[display](dk,orientation=[-90,0]);
  interface(plotdevice=win);
```

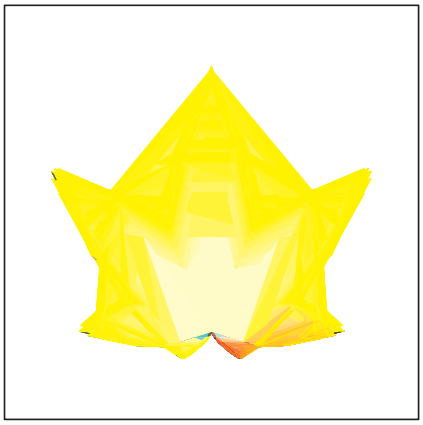


Abb. ani31_07.ps
10 Ahornblätter übereinander

3 Varianten der Animation mit gleichem Verlauf

```
> # Blatt nach oben
  display3d(dk,insequence=true);
> # Blatt nach links unten
  plots[display](dk,insequence=true,orientation=[-90,0]);
> # Blatt nach rechts unten
  pl1:=display3d(dk,insequence=true,orientation=[180,0]):
  pl1;
```

GIF Animated Transparent File

```
> interface(plotdevice=gif,
      plotoutput=datei8,
      plotoptions=`width=200,height=200,
      transparent=true`);
  plots[display](dk,insequence=true,orientation=[-90,0]);
  interface(plotdevice=win);
```

7.3 Das Ahornblatt in 5 Varianten

In den Kapiteln 5.2 und 7.2 haben wir die schon Möglichkeiten der Darstellung des Ahornblatts, seiner Animation und Exports als animiertes transparentes *gif*-File kennengelernt. Die folgenden Versionen stellen eine Fortsetzung dar. Die Idee für diese Programme ist aus [3]. Dabei lassen wir in der Animation das Ahornblatt als Contour unterschiedlich wachsen, sich spiegeln und drehen. Die Überlagerung der Frames des jeweiligen Ahornblatts geben wir als *eps*-File aus, indem wir das Arbeitsblatt *ani32_.mws* nach Ausführung mit den entsprechenden Plots nach LaTeX exportieren. Dabei werden die gewünschten Plots als *eps*-Files im Format *landscape* und s/w generiert.

```
> pfad := `D:/Neundorf/Maple2/`;
  name1 := `ani32_01.gif`;
  name2 := `ani32_02.gif`;
  name3 := `ani32_03.gif`;
  name4 := `ani32_04.gif`;
  name5 := `ani32_05.gif`;

  datei1 := cat(pfad,name1);
  datei2 := cat(pfad,name2);
  datei3 := cat(pfad,name3);
  datei4 := cat(pfad,name4);
  datei5 := cat(pfad,name5);
```

1. Definition des Ahornblatts

```
> setoptions(scaling=constrained,axes=none):
  X:=(x,k)->cos(x)*(1+k*sin(x))*(1+k*0.3*cos(8*x))*
          (1+k*0.1*cos(24*x)):
  Y:=(x,k)->sin(x)*(1+k*sin(x))*(1+k*0.3*cos(8*x))*
          (1+k*0.1*cos(24*x)):
```

1. Bild

```
> animate([X(x,k),Y(x,k),x=0..2*Pi],k=0..1,color=green,thickness=3);
```

Parameterplot

```
> d1:=k->plot([X(x,k),Y(x,k),x=0..2*Pi],thickness=2,color=green):
  dk1:=[seq(d1((j-1.0)/15.0),j=1..16)]:
```

Endzustand des Blatts

```
> d1(1); display(dk1[16]);
```

Alle 16 Frames übereinander

```
> display(dk1); # Ahornblatt mit Spitze nach oben
```

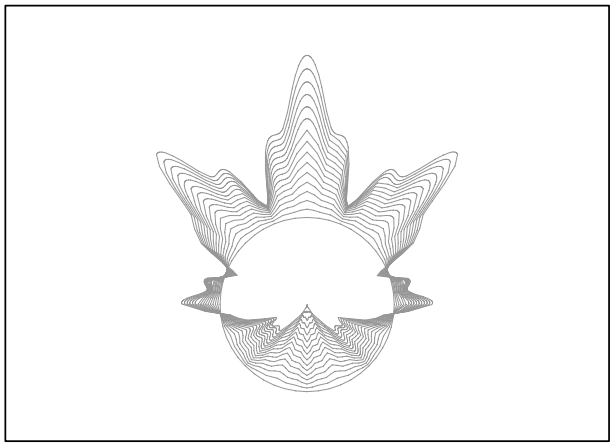


Abb. ani32_01.eps
16 Ahornblätter übereinander, $k = 0..1$

Ansicht aller 16 Entwicklungsstufen als Tableau

```
> pp:=array(1..2,1..8,[]):
  for j from 1 to 8 do
    pp[1,j]:=display(dk1[j]):  pp[2,j]:=display(dk1[j+8]):
  od:
  plots[display](pp);
```

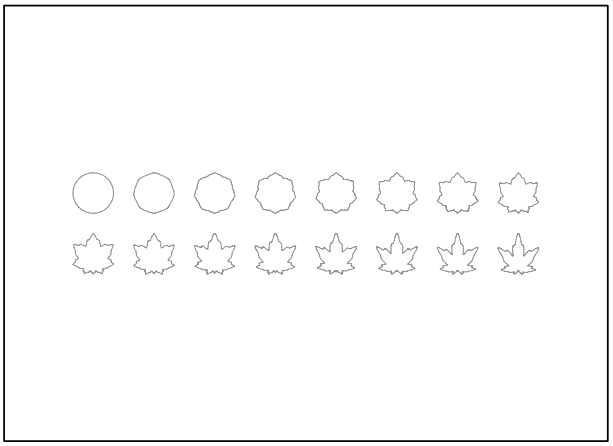


Abb. ani32_02.eps
Ansicht aller 16 Entwicklungsstufen als Tableau

3 Varianten der Animation

```
> display(dk1,insequence=true);
> pl1:=display(dk1,insequence=true):  pl1;
> plots[display](dk1,insequence=true);
> # 16 Ansichten als Tableau
  display(pl1);
```

GIF Animated Transparent File

```
> interface(plotdevice=gif,
      plotoutput=datei1,
      plotoptions=`width=200,height=200,transparent=true`);
  plots[display](dk1,insequence=true);
  interface(plotdevice=win);
```

2. Bild

Anderer Verlauf des Blatts: weiter entwickeln

```
> animate([X(x,k),Y(x,k),x=0..2*Pi],k=0..2,color=gold,thickness=3);
```

Parameterplot

```
> d2:=k->plot([X(x,k),Y(x,k),x=0..2*Pi],thickness=2,color=gold):
  dk2:=[seq(d2(2*(j-1.0)/15.0),j=1..16)]:
```

Endzustand des Blatts

```
> d2(2); display(dk2[16]);
```

Alle 16 Frames übereinander

```
> display(dk2); # Ahornblatt mit Spitze nach oben
```

Abb. ani32_03.eps
Blatt weiter entwickeln, $k = 0..2$

Ansicht aller 16 Entwicklungsstufen als Tableau

```
> pp:=array(1..2,1..8,[]):
  for j from 1 to 8 do
    pp[1,j]:=display(dk2[j]):
    pp[2,j]:=display(dk2[j+8]):
  od:
  plots[display](pp);
```

3 Varianten der Animation

```
> display(dk2,insequence=true);
> pl2:=display(dk2,insequence=true):  pl2;
> plots[display](dk2,insequence=true);

> # 16 Ansichten als Tableau display(pl2);
```

GIF Animated Transparent File

```
> interface(plotdevice=gif,
      plotoutput=datei2,
      plotoptions=`width=200,height=200,transparent=true`);
  plots[display](dk2,insequence=true);
  interface(plotdevice=win);
```

3. Bild

Anderer Verlauf des Blatts: vor und zurück (mit Spiegelung)

```
> animate([X(x,1-k),Y(x,1-k),x=0..2*Pi],k=0..2,color=blue,thickness=3);
```

Parameterplot

```
> d3:=k->plot([X(x,k),Y(x,k),x=0..2*Pi],thickness=2,color=blue):
  dk31:=[seq(d3(1.0-2*(j-1.0)/15.0),j=1..16)]:
  dk32:=[seq(d3(-1.0+2*(j-1.0)/15.0),j=1..16)]:
```

Zustände des Blatts

```
> d3(1); d3(0); d3(-1);
  display(dk31[1]); display(dk31[16]);
  display(dk32[1]); display(dk32[16]);
```

Alle 32 Frames übereinander

```
> # Ahornblatt mit Spiegelung
  dk3:=[op(dk31),op(dk32)]:
  display(dk3);
```

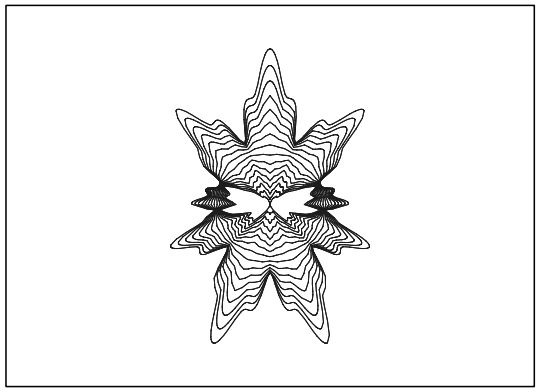


Abb. ani32_04.eps
Blatt nach oben und unten
(mit Spiegelung)

Ansicht aller 32 Entwicklungsstufen als Tableau

```
> pp:=array(1..4,1..8,[]):
  for j from 1 to 8 do
    pp[1,j]:=display(dk31[j]):
    pp[2,j]:=display(dk31[j+8]):
    pp[3,j]:=display(dk32[j]):
    pp[4,j]:=display(dk32[j+8]):
  od:
  plots[display](pp);
```

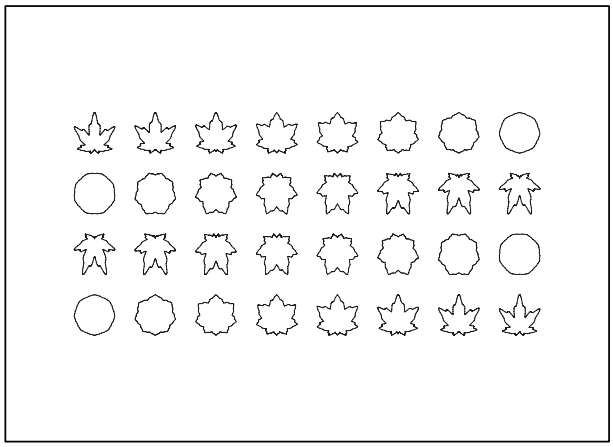


Abb. ani32_05.eps
Ansicht aller 32 Entwicklungsstufen als Tableau

3 Varianten der Animation

```
> display(dk3,insequence=true);
> pl3:=display(dk3,insequence=true):
  pl3;
> plots[display](dk3,insequence=true);

> # 32 Ansichten als Tableau
  display(pl3);
```

GIF Animated Transparent File

```
> interface(plotdevice=gif,
      plotoutput=datei3,
      plotoptions=`width=200,height=200,transparent=true`);
  plots[display](dk3,insequence=true);
  interface(plotdevice=win);
```

2. Definition des Ahornblatts mit mehr Zacken

```
> setoptions(scaling=constrained,axes=none):

  s := t->100/(100+(t-Pi/2)^8):
  r := t->s(t)*(2-sin(7*t)-cos(30*t)/2):
  v := (t,u)->u*r(t)/2:
```

4. Bild

```
> animate([v(t,u),t,t=-Pi/2..3/2*Pi],u=1..2,
      numpoints=200, coords=polar, axes=none,
      color=green, thickness=2);
```

Parameterplot

```
> d4:=u->plot([v(t,u),t,t=-Pi/2..3/2*Pi],
      numpoints=200, coords=polar, axes=none,
      color=green, thickness=2):
  dk4:=[seq(d4(1.0+(j-1.0)/15.0),j=1..16)]:
```

Zustände des Blatts

```
> d4(1); d4(2);
  display(dk4[1]); display(dk4[16]);
```

Alle 16 Frames übereinander

```
> # Ahornblatt mit Spitze nach oben
  display(dk4);
```

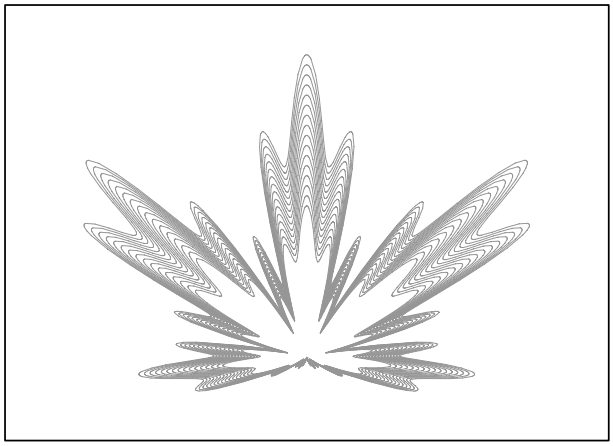


Abb. ani32_06.eps
Alle 16 Frames des Blattes übereinander, $1 \leq u \leq 2$

Ansicht aller 16 Entwicklungsstufen als Tableau, 16 gleiche Bilder

```
> pp:=array(1..2,1..8,[]):
  for j from 1 to 8 do
    pp[1,j]:=display(dk4[j]):
    pp[2,j]:=display(dk4[j+8]):
  od:
  plots[display](pp);
```

3 Varianten der Animation

```
> display(dk4,insequence=true);
> pl4:=display(dk4,insequence=true):
  pl4;
> plots[display](dk4,insequence=true);
```

```
> # 16 gleiche Ansichten als Tablaeu
  display(pl4);
```

GIF Animated Transparent File

```
> interface(plotdevice=gif,
      plotoutput=datei4,
      plotoptions=‘width=200,height=200,transparent=true‘);
  plots[display](dk4,insequence=true);
  interface(plotdevice=win);
```

5. Bild

Parameterplot für Ahornblatt, Drehung

```
> d5:=u->plot([v(t,2-abs(u-2)),t+2*Pi*(u-1),t=-Pi/2..3/2*Pi],
      numpoints=200, coords=polar, axes=none,
      color=COLOR(RGB,rand()/10^12,rand()/10^12,rand()/10^12),
      thickness=2):

  # 2 Umkreisungen, 1<=u<=3
  dk5:=[seq(d5(1.0+(j-1.0)/31.0),j=1..62)]:
```

62 Frames übereinander

```
> display(dk5);
```

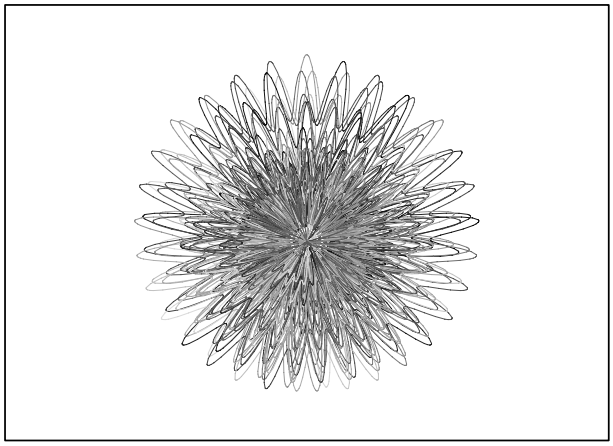


Abb. ani32_07.eps
2 Umkreisungen, $1 \leq u \leq 3$

Variante der Animation

```
> display(dk5,insequence=true);
```

GIF Animated Transparent File

```
> interface(plotdevice=gif,
      plotoutput=datei5,
      plotoptions=‘width=200,height=200,transparent=true‘);
  plots[display](dk5,insequence=true);
  interface(plotdevice=win);
```

7.4 Farbige Quadrate

Im ersten Beispiel färben wir ein Quadrat ein, wobei die Farben ständig wechseln. Dann unterteilen wir ein Quadrat wie ein Schachbrett (Karomuster) und lassen ein kleines Quadrat ("Spielfeld") mit abwechselnden Farben zeilenweise "von Feld zu Feld laufen".

1. Bild: Quadrat mit Farbwechsel

```
> name1 := `ani33_01.ps`;
  name2 := `ani33_02.gif`;
  datei1 := cat(pfad,name1);
  datei2 := cat(pfad,name2);

> setoptions(scaling=constrained,axes=none):

> p8 := [seq(plots[polygonplot]([[0,0],[1,0],[1,1],[0,1]],
        # color=green), alle Quadrate green
        # color=rand()/10^12), alle Quadrate weiss mit Rand
        # ohne Farbe
      color=COLOR(RGB,rand()/10^12,rand()/10^12,rand()/10^12)),
      i=1..20)]:
```

Alle 20 farbigen Quadrate übereinander

```
> display(p8);
```

Animation

```
> display(p8,insequence=true);
```

Ansicht aller 20 farbigen Quadrate als Tableau

```
> pp:=array(1..4,1..5,[]):
  for j from 1 to 5 do
    pp[1,j]:=display(p8[j]):
    pp[2,j]:=display(p8[j+5]):
    pp[3,j]:=display(p8[j+10]):
    pp[4,j]:=display(p8[j+15]):
  od:
  plots[display](pp);
```

PS File für Tableau

```
> interface(plotdevice=ps,
      plotoutput=datei1,
      plotoptions=`color,portrait,width=640,height=480`);
  plots[display](pp);
  interface(plotdevice=win);
```

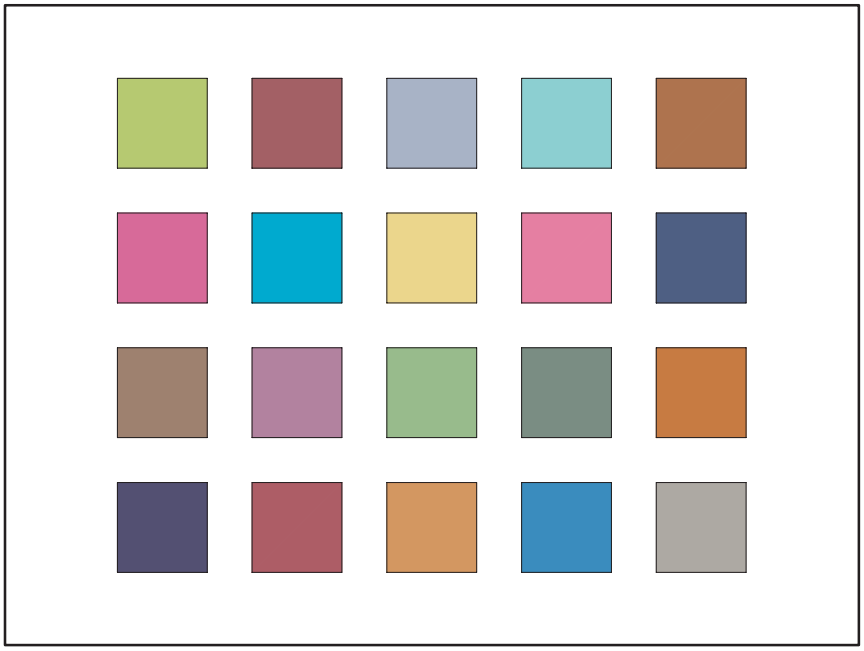


Abb. ani33_01.ps
20 farbige Quadrate
als Tableau

GIF Animated Transparent File

```
> interface(plotdevice=gif,
      plotoutput=datei2,
      plotoptions=‘width=200,height=200,transparent=true‘);
  plots[display](p8,insequence=true);
  interface(plotdevice=win);
```

2. Bild: Karo mit laufendem kleinem farbigem Quadrat

```
> name3 := ‘ani33_03.ps‘;
  name4 := ‘ani33_04.ps‘;
  name5 := ‘ani33_05.gif‘;
  datei3 := cat(pfad,name3);
  datei4 := cat(pfad,name4);
  datei5 := cat(pfad,name5);

> setoptions(scaling=constrained,axes=none):
> p9:=[seq(seq(plots[polygonplot]([[i,j],[i+1,j],[i+1,j+1],[i,j+1]],
      color=COLOR(RGB,rand()/10^12,rand()/10^12,rand()/10^12)),
      i=1..10),j=1..10)]:
```

Alle 100 Ansichten übereinander

```
> plots[display](p9);
```

PS File

```
> interface(plotdevice=ps,
      plotoutput=datei3,
      plotoptions=‘color,portrait,width=640,height=480‘);
  plots[display](p9);
  interface(plotdevice=win);
```

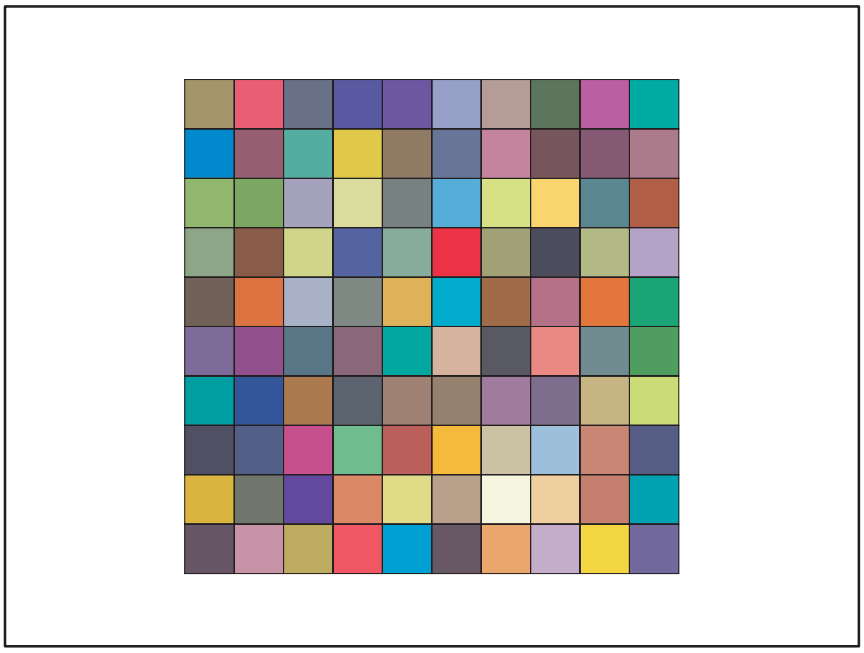


Abb. ani33_03.ps
100 Ansichten
übereinander

Animation

```
> plots[display](p9,insequence=true);
```

Ansicht aller 100 kleinen Quadrate im Karo als Tableau

```
> qq:=array(1..10,1..10,[]):
  for j from 1 to 10 do
    for i from 1 to 10 do
      qq[10+1-i,j]:=display(p9[j+10*(i-1)]):
    od:
  od:
  plots[display](qq);
```

PS File für Tableau

```
> interface(plotdevice=ps,
      plotoutput=datei4,
      plotoptions=‘color,portrait,width=640,height=480‘);
  plots[display](qq);
  interface(plotdevice=win);
```

Die Grafik ani33_04.ps hat dieselbe Farbpalette und ähnliche Struktur (zusätzlicher kleiner Abstand zwischen den Feldern) wie ani33_03.ps.

GIF Animated Transparent File

```
> interface(plotdevice=gif,
      plotoutput=datei5,
      plotoptions=‘width=200,height=200,transparent=true‘);
  plots[display](p9,insequence=true);
  interface(plotdevice=win);
```

7.5 Oberfläche als Gitter

Mittels `plot3d` definieren wir eine Oberfläche als Gitter (Liniengitter, wireframe) und konstruieren daraus eine Folge solcher Flächen, indem wir eine zusätzlichen Parameter einbringen und "laufen" lassen.

Bild: Gitteroberfläche

```
> with(plots,animate3d):
  pfad := `D:/Neundorf/Maple2/`;
  name1 := `ani34_1.gif`;
  datei1 := cat(pfad,name1);
```

Definition

```
> setoptions(scaling=constrained,axes=none):
> r:=(x,y,n)->n*sin(x)*exp(y):
```

Animation

```
> animate3d(r(x,y,n), x=-3.5..4,y=4..10,n=0..20,
          frames=21, orientation=[65,90],
          style=wireframe, color=gray);
```

Framesequenz

```
> d10:=n->plot3d(r(x,y,n), x=-3.5..4,y=4..10,
          orientation=[65,90], style=wireframe, color=gray):
  dk10:=[seq(d10(n),n=0..20)]:
```

Kontrolle einzelner Plots und ihrer Größe

```
> d10(0);
  display(d10(1),axes=normal);
  display(d10(2),axes=normal);
  display(dk10[20],axes=normal);
```

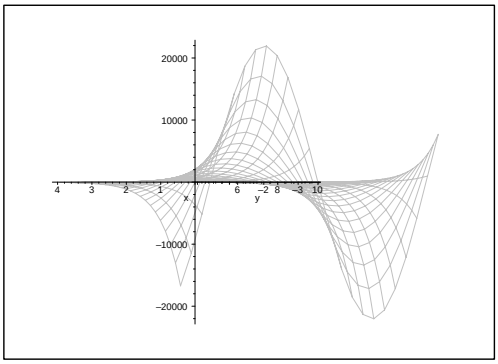


Abb. ani34_01.eps
Gitteroberfläche
1.Frame

Plottet man andere Frames, so ändert sich nur der Maßstab auf der r-Achse. Die Gitterfläche behält die gleiche Form.

Alle 21 Frames übereinander

```
> display(dk10);
```

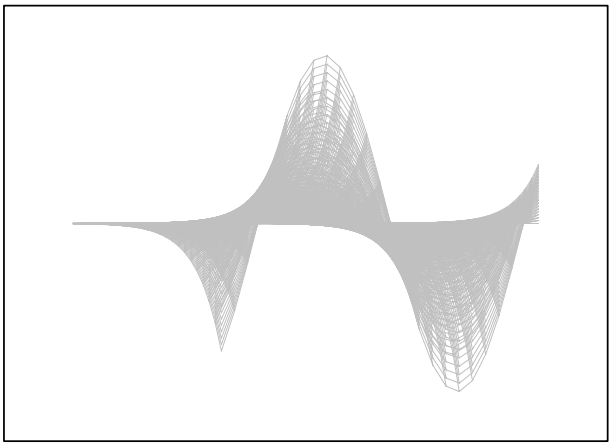


Abb. ani34_04.eps
Gitteroberfläche
Ansicht aller 21 Frames
übereinander

3 Varianten der Animation

```
> display(dk10,insequence=true);
> pl10:=display(dk10,insequence=true):
  pl10;
> plots[display](dk10,insequence=true);

> # 1.Ansicht und weitere 20 gleiche Ansichten als Tablaeu
  display(pl10);
```

GIF Animated Transparent File

```
> interface(plotdevice=gif,
      plotoutput=datei1,
      plotoptions=`width=200,height=200,transparent=true`);
  plots[display](p10,insequence=true);
  interface(plotdevice=win);
```

7.6 Torus in drei Varianten

Wir definieren eine Torusfunktion $c(t) = (x(t), y(t), z(t))$, $t \in [t_0, t_1]$, (Parameterfunktion) und stellen sie grafisch dar mittels `plots[tubeplot]`.

Drei Varianten der Betrachtung des Torus werden uns hier interessieren.
Für $[t_0, t_1] = [0, t_1]$ erhalten wir mit wachsender Intervallgrenze t_1 einen sich stetig verlängernden Torus ("Schlauch wird immer länger"), bis bei $[t_0, t_1] = [0, 2\pi]$ eine in sich geschlossene gewundene Torusfläche entsteht. Natürlich kann man auch Teilstücke des Torus ("Schlauchstücke") betrachten und den geschlossenen Torus aus solchen durch Aneinanderfügen zusammensetzen.
Dann lässt man ein buntes Toruselement laufen, ohne dass der Torus als ganzes zu sehen ist.
Ein besonderer Effekt ist, im transparenten Liniengitter des Torus ein buntes Toruselement kreisen zu lassen, ähnlich zu einer Lichtschlange mit beweglichem Licht.

```
> pfad := `D:/Neundorf/Maple2/`;
  name2 := `ani34_2.ps`;
  name3 := `ani34_3.ps`;
  name4 := `ani34_4.gif`;
  name5 := `ani34_5.ps`;
  name6 := `ani34_6.ps`;
  name7 := `ani34_7.gif`;
  name8 := `ani34_8.ps`;
  name9 := `ani34_9.ps`;
  name10 := `ani34_10.ps`;
  name11 := `ani34_11.gif`;

  datei2 := cat(pfad,name2);
  datei3 := cat(pfad,name3);
  datei4 := cat(pfad,name4);
  datei5 := cat(pfad,name5);
  datei6 := cat(pfad,name6);
  datei7 := cat(pfad,name7);
  datei8 := cat(pfad,name8);
  datei9 := cat(pfad,name9);
  datei10 := cat(pfad,name10);
  datei11 := cat(pfad,name11);
```

1. Bild: Zunehmender Torus, von Beginn an und verlängern
Konstruktion aus 10 Teilstücken

```
> setoptions(scaling=constrained,axes=none):
> curve:=[-10*cos(t)-2*cos(5*t)+15*sin(2*t),
          -15*cos(2*t)+10*sin(t)-2*sin(5*t),
          10*cos(3*t)]:

> ti:=time():  # Zeitmessung
  d11:=i->plots[tubeplot](curve,t=0..2*Pi*i/n, radius=3):
  n:=10:
  dk11:=[seq(d11(i),i=1..n)]:
  Rechenzeit:=time()-ti,`sec`;
```

Kontrolle einzelner Frames

```
> display(d11(1),style=patch); display(d11(2),style=patch);
```

10.Frame = Torus

```
> display(d11(10),style=patch);
```

PS File für 10.Frame = Torus

```
> interface(plotdevice=ps,
      plotoutput=datei2,
      plotoptions=`color,portrait,width=640,height=480`);
  plots[display](d11(10),style=patch);
  interface(plotdevice=win);
```

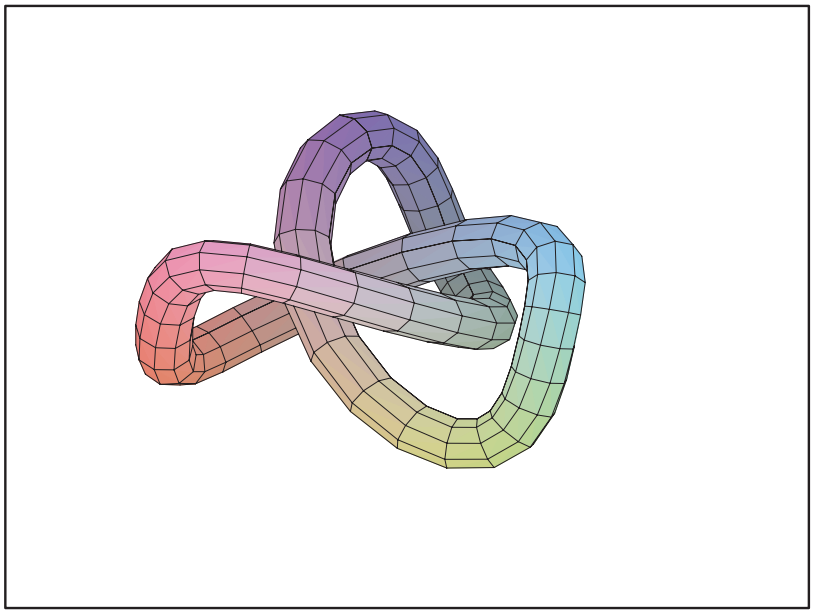


Abb. ani34_2.ps
10.Frame = Torus

Alle Frames übereinander

```
> plots[display](dk11,style=patch);
```

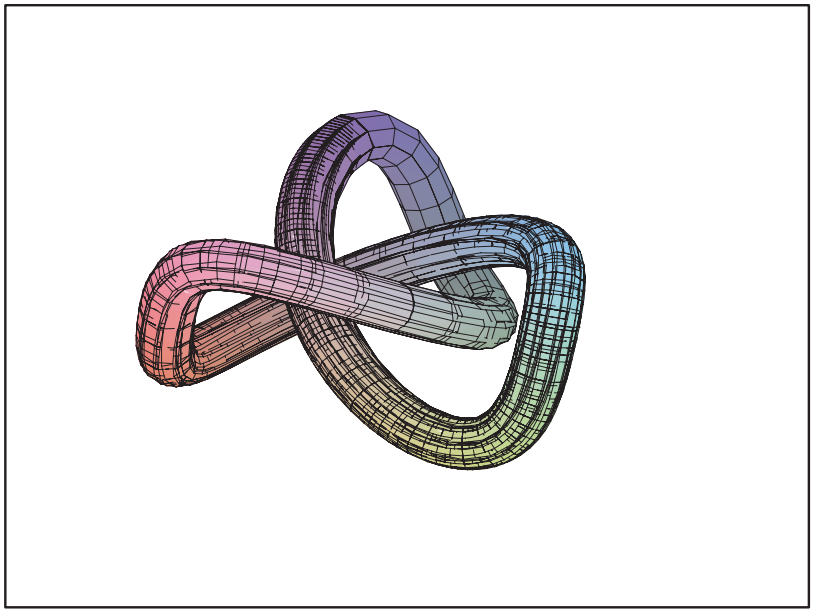


Abb. ani34_3.ps
Alle Frames
übereinander

PS File für alle Frames übereinander

```
> interface(plotdevice=ps,
      plotoutput=datei3,
      plotoptions=‘color,portrait,width=640,height=480‘);
  plots[display](dk11,style=patch);
  interface(plotdevice=win);
```

Animation

```
> plots[display](dk11,insequence=true,style=patch);
```

GIF Animated Transparent File

```
> interface(plotdevice=gif,
      plotoutput=datei4,
      plotoptions=‘width=200,height=200,transparent=true‘);
  plots[display](dk11,insequence=true,style=patch);
  interface(plotdevice=win);
```

2. Bild: Torus, stückweise anzeigen/aneinanderfügen

Unterteilung in 10 Teilstücke

```
> ti:=time():
  d12:=i->plots[tubeplot](curve,t=2*Pi*(i-1)/n..2*Pi*i/n, radius=3):
  n:=10:
  dk12:=[seq(d12(i),i=1..n)]:
  Rechenzeit:=time()-ti,‘sec‘;
```

Kontrolle einzelner Frames

```
> display(d12(1),style=patch);
  display(d12(2),style=patch);
```

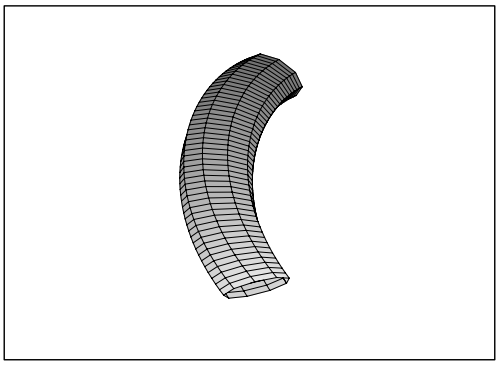


Abb. ani34_09.eps
1.Frame
$t = 0..2 * \pi * i/n,\ i = 1$

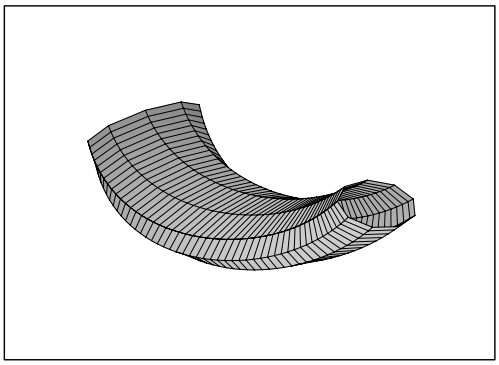


Abb. ani34_10.eps
2.Frame
$t = 2 * \pi * (i-1)/n..2 * \pi * i/n,\ i = 2$

Alle Frames übereinander = aneinander

```
> plots[display](dk12,style=patch);
```

PS File für alle 10 Frames übereinander = aneinander

```
> interface(plotdevice=ps,
      plotoutput=datei5,
      plotoptions=‘color,portrait,width=640,height=480‘);
  plots[display](dk12,style=patch);
  interface(plotdevice=win);
```

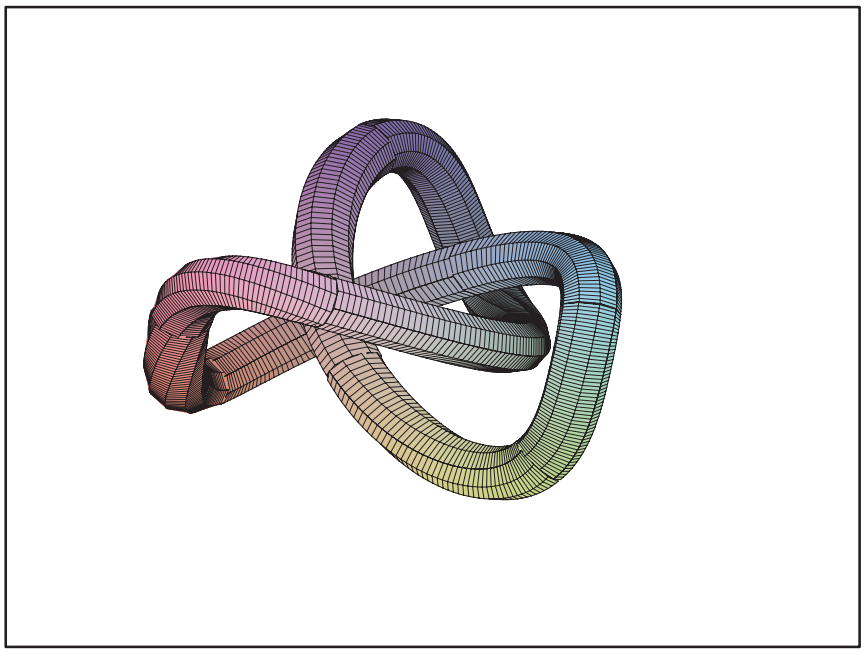


Abb. ani34_5.ps
Alle 10 Frames
übereinander
= aneinander

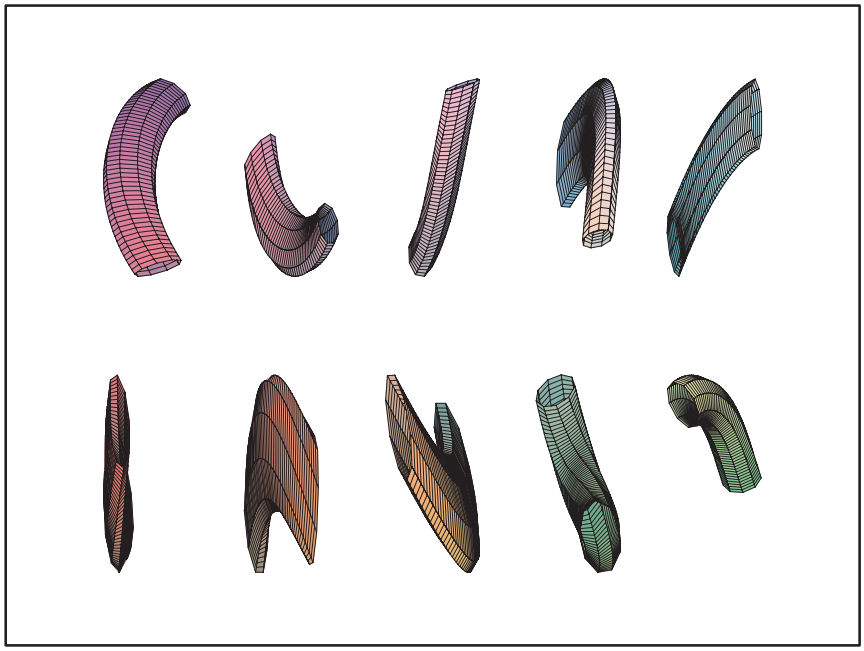


Abb. ani34_6.ps
Alle 10 Frames
als Tableau

Alle 10 Frames als Tableau

```
> pp:=array(1..2,1..5,[]):
  for j from 1 to 5 do
    pp[1,j]:=display(dk12[j],style=patch):
    pp[2,j]:=display(dk12[j+5],style=patch):
  od:
  plots[display](pp);
```

PS File für alle 10 Frames als Tableau

```
> interface(plotdevice=ps,
      plotoutput=datei6,
      plotoptions=‘color,portrait,width=640,height=480‘);
  plots[display](pp); interface(plotdevice=win);
```

Animation

```
> plots[display](dk12,insequence=true,style=patch);
```

GIF Animated Transparent File

```
> interface(plotdevice=gif,
      plotoutput=datei7,
      plotoptions=‘width=200,height=200,transparent=true‘);
  plots[display](dk12,insequence=true,style=patch);
  interface(plotdevice=win);
```

3. Bild: Torus, stückweise anzeigen/aneinanderfügen und Gesamtgitter anzeigen (“Lichtschlange“)

Unterteilung in 10 Teilstücke

Transparentes Torusgitter

```
> d131:=plots[tubeplot](curve,t=0..2*Pi, radius=3,
          style=wireframe, color=gray):
```

Torusstück

```
> d132:=i->plots[tubeplot](curve,t=2*Pi*(i-1)/n..2*Pi*i/n, radius=3):
  n:=10:
  dk13:=[seq(d132(i),i=1..n)]:
```

Kontrolle einzelner Frames mit Gesamtgitter

```
> q1:=display(d131):
  q1;
> q2:=display(d132(1),style=patch):
  q2z:=plots[display](q1,q2):
  q2z;
```

PS File für Gesamtgitter und 1.Teilstück

```
> interface(plotdevice=ps,
      plotoutput=datei8,
      plotoptions=‘color,portrait,width=640,height=480‘);
  plots[display](q2z);
  interface(plotdevice=win);

> q2:=display(d132(2),style=patch):
  q2z:=plots[display](q1,q2):
  q2z;
```

PS File für Gesamtgitter und 2.Teilstück

```
> interface(plotdevice=ps,
      plotoutput=datei9,
      plotoptions=‘color,portrait,width=640,height=480‘);
  plots[display](q2z);
  interface(plotdevice=win);
```

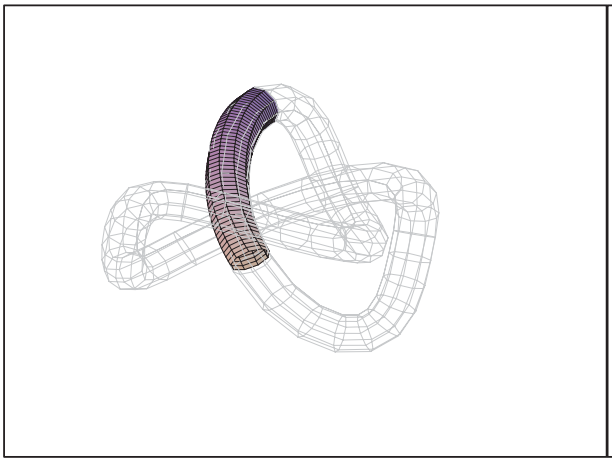


Abb. ani34_8.ps
Gesamtgitter und 1.Teilstück
$t = 0..2 * \pi * i/n,\ i = 1$

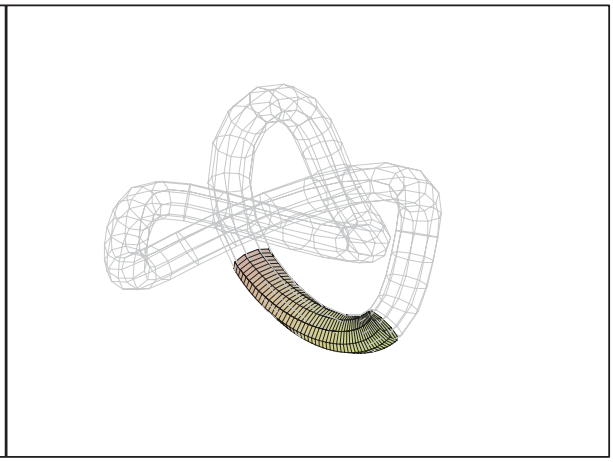


Abb. ani34_9.ps
Gesamtgitter und 2.Teilstück
$t = 2 * \pi * (i-1)/n..2 * \pi * i/n,\ i = 2$

```
> q2:=display(d132(3),style=patch):
  q2z:=plots[display](q1,q2):
  q2z;
```

Alle 10 Frames übereinander

```
> plots[display](dk13,style=patch);
```

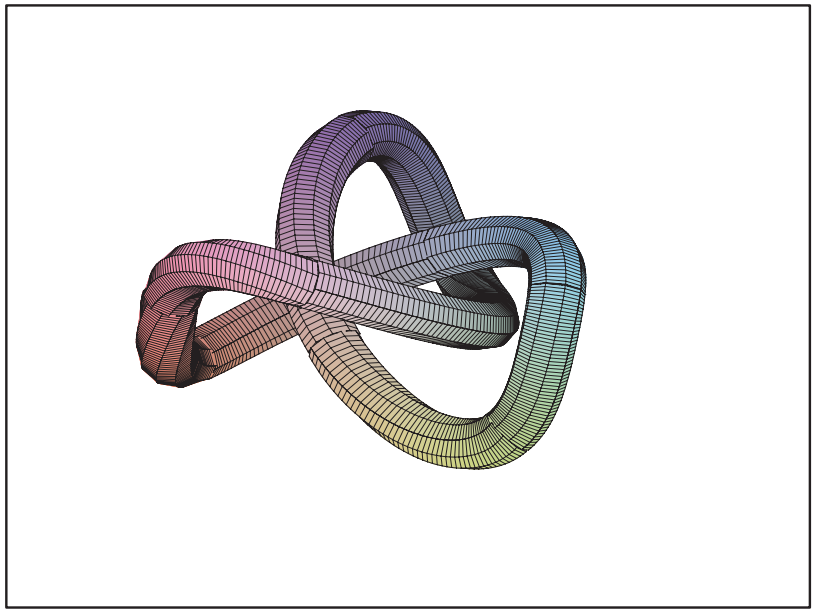


Abb. ani34_10.ps
Alle 10 Frames
übereinander

PS File für alle 10 Frames übereinander

```
> interface(plotdevice=ps,
      plotoutput=datei10,
      plotoptions='color,portrait,width=640,height=480');
  plots[display](dk13,style=patch);
  interface(plotdevice=win);
```

1.Animation ohne Torusgitter

```
> dk13a:=[seq(d132(i),i=1..n)]:
> display(dk13a,insequence=true,style=patch);
```

2.Animation mit Torusgitter

```
> for i from 1 to n do
    q2:=display(d132(i),style=patch):
    q2z[i]:=plots[display](q1,q2):
  od:
  dk13b:=[seq(q2z[i],i=1..n)]:
  display(dk13b,insequence=true,style=patch);
```

GIF Animated Transparent File

```
> interface(plotdevice=gif,
      plotoutput=datei11,
      plotoptions=‘width=200,height=200,transparent=true‘);
  plots[display](dk13b,insequence=true,style=patch);
  interface(plotdevice=win);
```

Lichtschlange

```
> for k from 1 to 10 do
    for j to 1 do qli:=display(d132(k),style=patch):  od:
    plots[display](q1,qli);
  od;
```

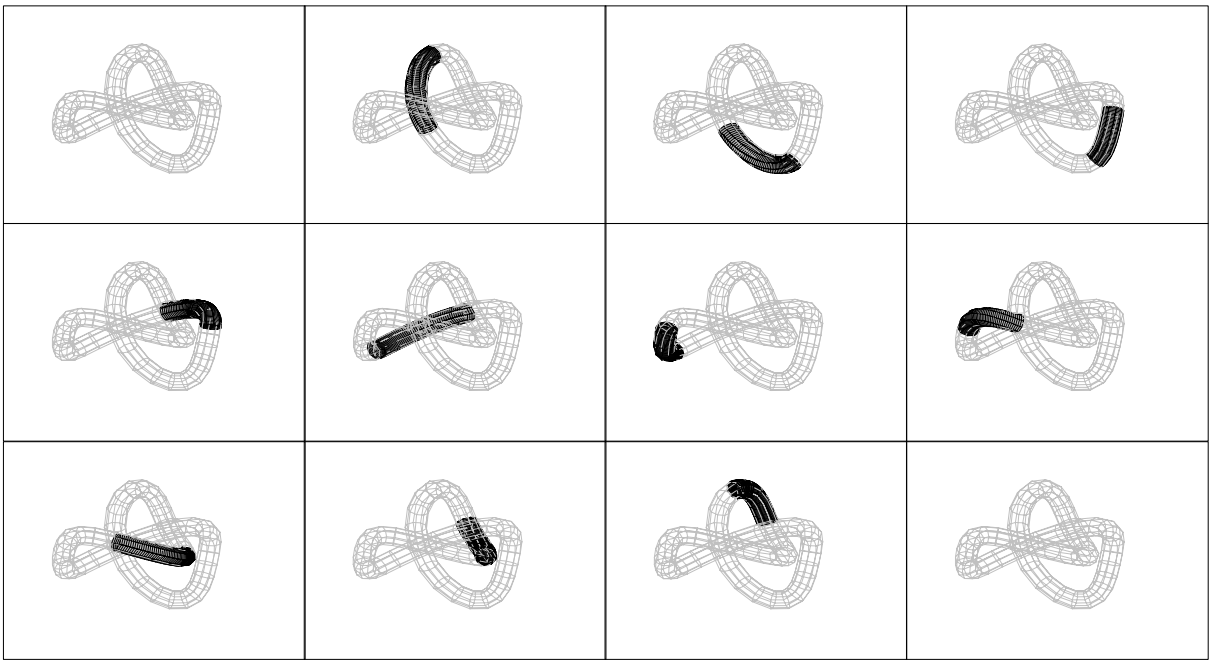


Abb. ani34a01.eps, ani34a02.eps, ..., ani34a11.eps, ani34a01.eps2

7.7 Turtle-Grafik mit Fraktalen, Bäumen und Mengen

Wir kommen zur Erzeugung der Schneeflockenkurve im Kapitel 3.2 zurück und werden diese Idee etwas ausbauen. Dazu verwenden wir eine Idee aus [17].
Wir definieren den Algorithmus in Form von Prozeduren. Wir geben verschiedene Möglichkeiten des Plots der Turtle-Grafik sowie seine Animation an und exportieren letztere als animiertes transparentes *gif*-File. Die grafischen Darstellungen werden beim Export der ausgeführten Worksheets *ani41..3_.mws* nach LaTeX als *eps*-Files im Format *landscape* und *s/w* generiert und ausgegeben. Wir verzichten hier auf eine mögliche Ausgabe als *ps*-Files mit Farboption.

Konstruktion der Fraktale

Wir steuern durch eine Zeichenfolge das Zeichnen von Geradenstücken:

f - bedeutet: Zeichne eine Strecke der Länge 1,
p - bedeutet: Drehe die Zeichenrichtung um einen vorgegenen Winkel nach rechts,
m - bedeutet: Drehe die Zeichenrichtung um einen vorgegenen Winkel nach links.

Der Winkel sei beispielsweise $\pi/3$.
Dann bedeutet die Zeichenfolge f,p,p,f,p,p,f,p,p ein gleichseitiges Dreieck.

Jede Strecke (jedes Zeichen f) dieser so erzeugten Grundstruktur kann nun durch eine Regel weiterverarbeitet werden - und dies kann mehrmals wiederholt werden.
Soll etwa jede Strecke wie bei der Schneeflockenkurve gedrittelt, und die mittlere Strecke durch eine gleichseitiges Dreieck ersetzt werden, so lautet die *Abbildungsregel* für eine Strecke (also für f):

f,m,f,p,p,f,m,f.

Wir benutzen diese Regeln in der folgenden Prozedur, um uns das Ergebnis nach mehrmaligen Anwenden der Abbildungsregel zeichnen zu lassen. Im Prozedurkopf sind die Parameter kommentiert. Für die Umsetzung der Abbildungsregel auf eine Struktur verwendet man das Kommando `map`. Die verschiedenen Ausgabevarianten, die in der Prozedur eingebaut sind, werden durch einen *Indikator/Auswahlparameter* gesteuert.

Das Fraktal Schneeflocke

```
> star:=proc(structure::list, repeatrule::procedure,
             level::integer, nr::integer, ind::integer)

  # list       - Liste fuer Struktur
  # repeatrule - Abbildungsregel
  # level      - Anzahl der Regelanwendungen auf Grundstruktur
  # nr         - Nummer der ausgewaehlten Abbildung, 0..level
  # ind        - Variante der Ausgabe

  local i,j,k,plant,pts,lst,tt,xx,yy,q,qs,qq;

  for j from 0 to level
  do
    lst:=NULL;
```

```
  plant:=structure;
  if j>0 then
    for i from 1 to j do plant:=map(repeatrule,plant) od;
  fi;
  xx:=0; yy:=0; tt:=1/3*Pi; pts:=[xx,yy];
  for i in plant
  do
    if i=p then tt:=tt+1/3*Pi;
      elif i=m then tt:=tt-1/3*Pi
      elif i=f then xx:=xx+cos(tt);
      yy:=yy+sin(tt);
      pts:=pts,[xx,yy]
    fi
  od;
  q[j]:=plot({lst,[pts]},color=blue):  # q[j]:=plot([pts],color=blue):
  # Zufallsfarben
  # color=COLOR(RGB,rand()/10^12,rand()/10^12,rand()/10^12)):
od;
qs:=[seq(q[j],j=0..level)]:

# Framesequenz, Frames gross und neben/untereinander
if ind=0 then
  q:=plots[display](qs,axes=NONE,scaling=constrained,insequence=true):
  plots[display](q)

# Animation
  elif ind=1 then
  plots[display](qs,axes=NONE,scaling=constrained,insequence=true)

# Frames uebereinander
  elif ind=2 then
  plots[display](qs,axes=NONE,scaling=constrained)

# Frames klein und als Tableau neben/untereinander
  elif ind=3 then
  qq:=array(1..nops(qs),[]):  # geeignetes Array
  for k from 1 to nops(qs) do
    qq[k]:=plots[display](qs[k],axes=none,scaling=constrained):
  od:
  plots[display](qq)

# Ausgewaehltes Frame mit Massstab
  elif ind=4 then
      # q:=plots[display](qs,axes=NONE,scaling=constrained,
      #          insequence=true):
      # plots[display](op(nr+1,op(1,q)));
    if nr>=0 and nr<=level then
      plots[display](qs[nr+1],view=[-50..50,0..100],
            axes=boxed,scaling=constrained,tickmarks=[2,2]);
    fi;
fi;
end:
```

Abbildungsregel

```
> repeatrule:=x->subs(f=(f,m,f,p,p,f,m,f),x):
```

Schneeflocke
Grundstruktur = Dreieck mit 4 Fortsetzungen

```
> level:=4:
```

Flocken groß und neben-/untereinander

```
> star([f,p,p,f,p,p,f,p,p],repeatrule,level,0,0);
```

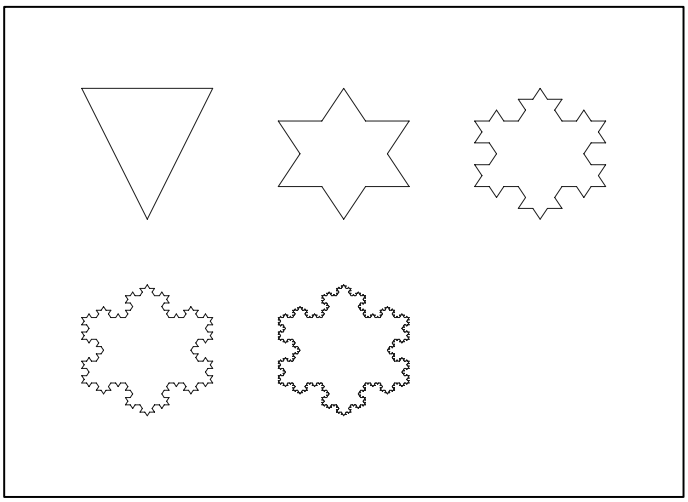


Abb.
ani41_01.eps

Alle Flocken übereinander

```
> star([f,p,p,f,p,p,f,p,p],repeatrule,level,0,2);
```

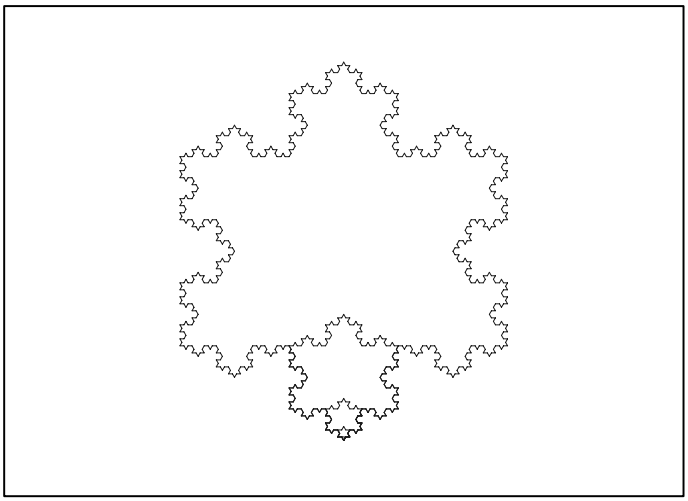


Abb.
ani41_03.eps

Animation

```
> star([f,p,p,f,p,p,f,p,p],repeatrule,level,0,1);
```

Beim Export der Animation wird nur das erste Frame berücksichtigt.

Flocken klein nebeneinander als Tableau

```
> star([f,p,p,f,p,p,f,p,p],repeatrule,level,0,3);
```

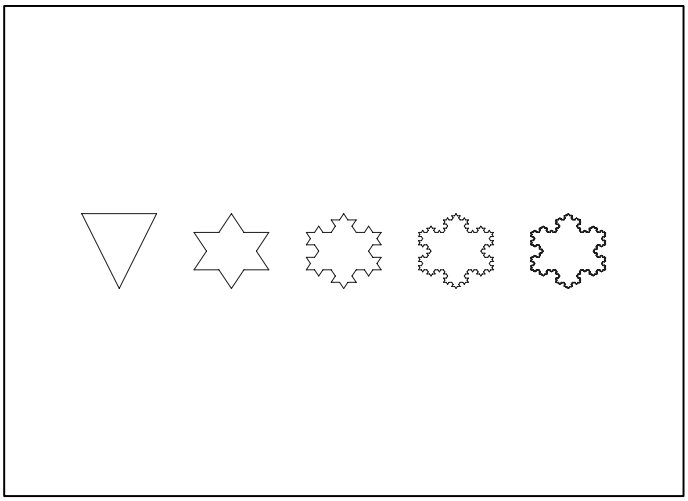


Abb.
ani41_04.eps

Ausgewählte Flocken mit Größenvergleich

```
> nr:=4:
  star([f,p,p,f,p,p,f,p,p],repeatrule,level,0,4);
  star([f,p,p,f,p,p,f,p,p],repeatrule,level,1,4);
  star([f,p,p,f,p,p,f,p,p],repeatrule,level,2,4);
  star([f,p,p,f,p,p,f,p,p],repeatrule,level,3,4);
  star([f,p,p,f,p,p,f,p,p],repeatrule,level,nr,4);
```

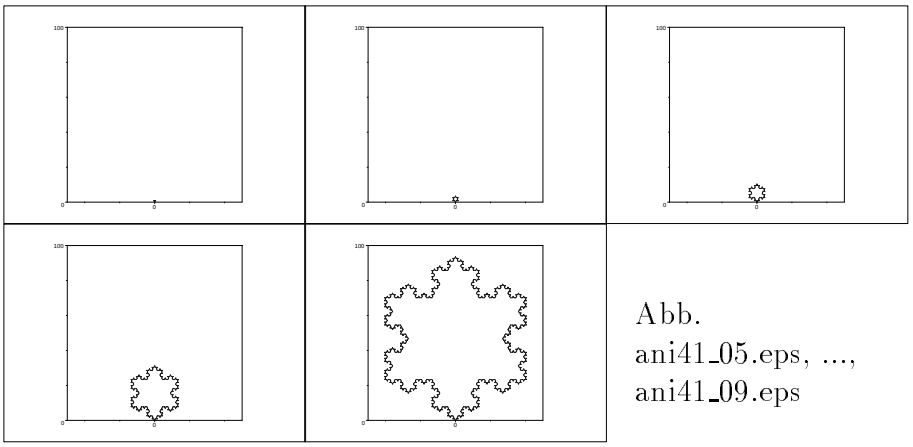


Abb.
ani41_05.eps, ...,
ani41_09.eps

Kontrolle

Was passiert mit den Merkmalen der einzelnen Abbildungen,
wenn diese zu einem Array zusammengefasst werden?
Größe, Maßstab, Achsen, Titel, Beschriftung, ...

```
> arr:=array(0..level,[]):
  for i from 0 to level do
    arr[i]:=star([f,p,p,f,p,p,f,p,p],repeatrule,level,i,4):
  od:
  plots[display](arr);
```

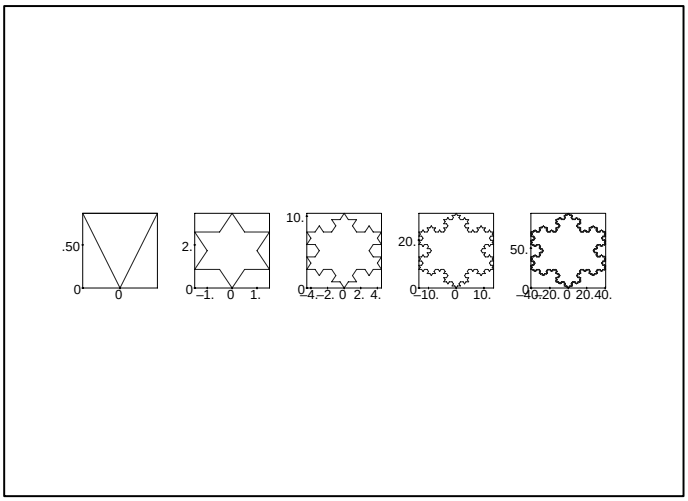


Abb.
ani41_10.eps

Modifikation der Fraktale bez. des Drehwinkels

Mit einer Strecke der Länge 1 haben wir mit dem Winkel $\varphi = \pi/3$ und der Abbildungsregel (Zeichenfolge) f → f,p,p,f,p,p,f,p,p ein gleichseitiges Dreieck erzeugt.
Einfacher wäre, den Winkel $\varphi = 2\pi/3$ zu nehmen und das gleiche Dreieck gemäß der Zeichenfolge f,p,f,p,f,p zu konstruieren.
Analog kann nun jede Strecke (jedes Zeichen f) einer Grundstruktur durch eine Regel weiterverarbeitet werden - und dies kann mehrmals wiederholt werden.
Wir testen einige andere Abbildungsregeln unter Verwendung unterschiedlicher Winkel.
Dazu erstellen wir eine weitere Prozedur.

```
> pfad := `D:/Neundorf/Maple2/`;
  name1 := `ani42_01.gif`;
  name2 := `ani42_02.gif`;
  name3 := `ani42_03.gif`;
  name4 := `ani42_04.gif`;
  name5 := `ani42_05.gif`;
  name6 := `ani42_06.gif`;

  datei1 := cat(pfad,name1);
  datei2 := cat(pfad,name2);
  datei3 := cat(pfad,name3);
  datei4 := cat(pfad,name4);
  datei5 := cat(pfad,name5);
  datei6 := cat(pfad,name6);

> star1:=proc(structure::list, repeatrule::procedure,
              phi::numeric, level::integer, ind::integer)

  # list       - Liste fuer Struktur
  # repeatrule - Abbildungsregel
  # phi        - Drehwinkel
  # level      - Anzahl der Regelanwendungen auf Grundstruktur
  # ind        - Variante der Ausgabe
```

```
local i,j,k,plant,pts,lst,tt,xx,yy,q,qq,q2;
global qs,qs2;

for j from 0 to level
do
  lst:=NULL;
  plant:=structure;
  if j>0 then
  for i from 1 to j do
    plant:=map(repeatrule,plant) od;
  fi;
  xx:=0; yy:=0; tt:=phi;
  pts:=[xx,yy];
  for i in plant
  do
    if i=p then tt:=tt+phi;
      elif i=m then tt:=tt-phi
      elif i=f then xx:=xx+cos(tt);
      yy:=yy+sin(tt);
      pts:=pts,[xx,yy]
    fi
  od;
  q[j]:=plot({lst,[pts]},
        color=COLOR(RGB,rand()/10^12,rand()/10^12,rand()/10^12)):
  # q[j]:=plot([pts],color=blue):  auch moeglich
od;
qs:=[seq(q[j],j=0..level)]:
qs2:=[seq(q[j],j=0..level),seq(q[level-j],j=1..level)]:

if ind=0 then
    q:=plots[display](qs,axes=none,scaling=constrained,
         insequence=true):
    plots[display](q);
    # neu
    q2:=plots[display](qs2,axes=none,scaling=constrained,
         insequence=true):
    plots[display](q2);

  elif ind=1 then
    plots[display](qs2,axes=none,scaling=constrained,insequence=true)

  elif ind=2 then
    plots[display](qs2,axes=none,scaling=constrained)

  elif ind=3 then
    qq:=array(1..nops(qs2),[]):
    for k from 1 to nops(qs2) do
      qq[k]:=plots[display](qs2[k],axes=none,scaling=constrained):
    od:
    plots[display](qq);
fi;
end:
```

1. Figur

```
> repeatrule:=x->subs(f=(f,m,f,p,p,f,m,f),x):
> phi:=evalf(Pi/3):
  star1([f,p,p,f,p,p,f,p,p],repeatrule,phi,3,0);
```

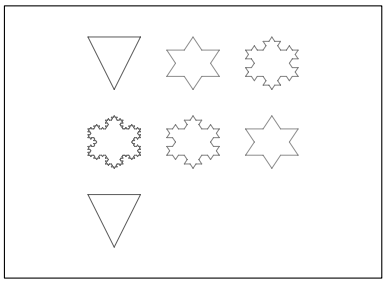


Abb.
ani42_01.eps

```
> star1([f,p,p,f,p,p,f,p,p],repeatrule,phi,3,1);
```

GIF Animated Transparent File

```
> interface(plotdevice=gif,
      plotoutput=datei1,
      plotoptions=‘width=200,height=200,transparent=true‘);
  plots[display](qs2,insequence=true,axes=none,thickness=3);
  interface(plotdevice=win);
```

2. Figur

```
> repeatrule:=x->subs(f=(f,m,m,f,p,p,f,p,p,f,m,m,f),x):
> phi:=evalf(Pi/3):
  star1([f,p,p,f,p,p,f,p,p],repeatrule,phi,3,1):
  star1([f,p,p,f,p,p,f,p,p],repeatrule,phi,3,0);
```

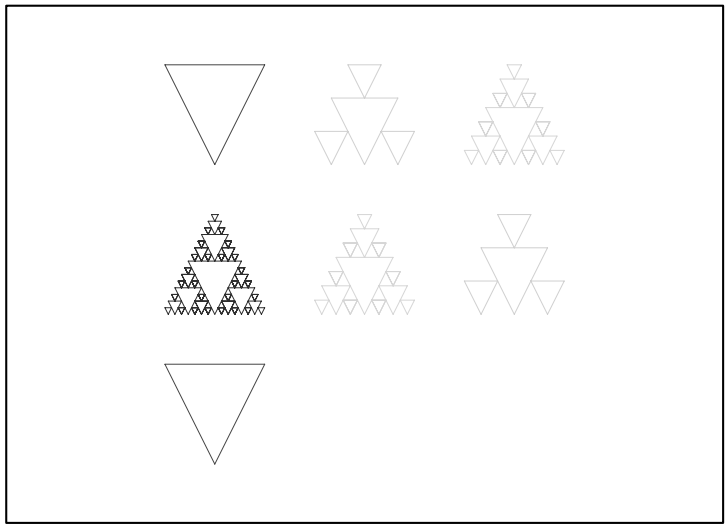


Abb.
ani42_02.eps

Bei dieser und den nächsten Figuren erfolgt die Ausgabe des *gif*-Files analog.

3. Figur

```
> repeatrule:=x->subs(f=(f,m,f,p,f,p,f,m,f),x):
> phi:=evalf(Pi/2):
  star1([f,p,f,p,f,p,f,p],repeatrule,phi,3,1):
  star1([f,p,f,p,f,p,f,p],repeatrule,phi,3,0);
```

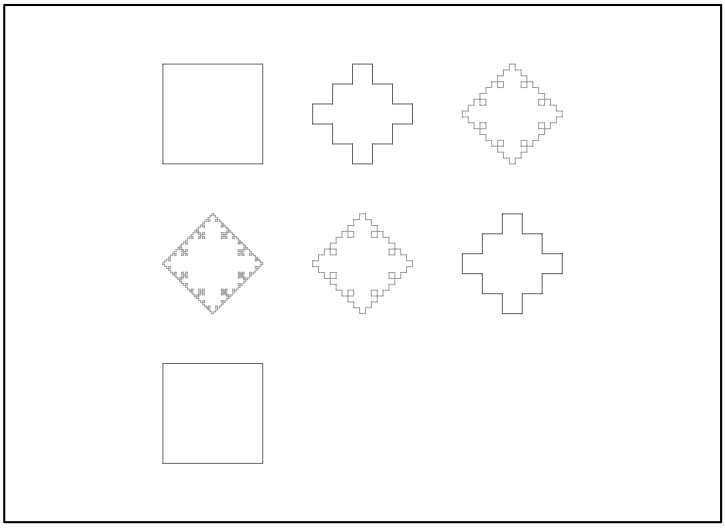


Abb.
ani42_03.eps

4. Figur

```
> repeatrule:=x->subs(f=(f,m,m,m,f,p,p,f,p,p,f,p,p,f,m,m,m,f),x):
> phi:=evalf(Pi/4):
  star1([p,f,p,p,f,p,p,f,p,p,f,p,p],repeatrule,phi,3,1):
  star1([p,f,p,p,f,p,p,f,p,p,f,p,p],repeatrule,phi,3,0);
```

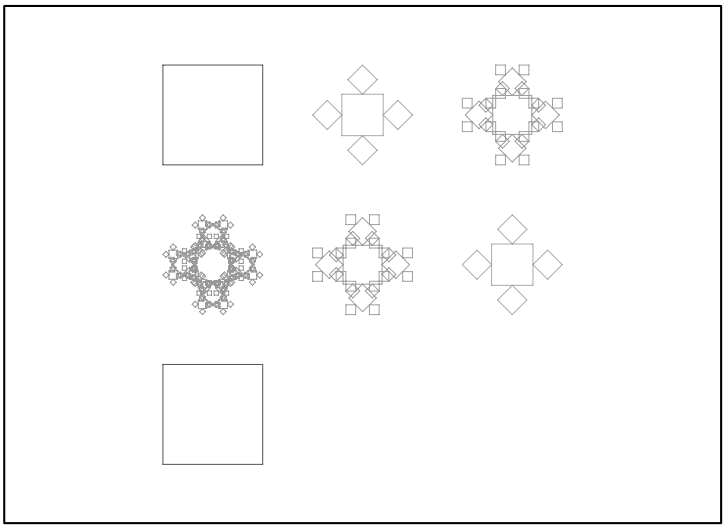


Abb.
ani42_04.eps

5. Figur

```
> repeatrule:=x->subs(f=(f,m,m,m,f,p,p,f,p,f,p,f,p,p,f,m,m,m,f),x):
> phi:=evalf(Pi/4):
  star1([p,f,p,p,f,p,p,f,p,p,f,p,p],repeatrule,phi,3,1):
  star1([p,f,p,p,f,p,p,f,p,p,f,p,p],repeatrule,phi,3,0);
```

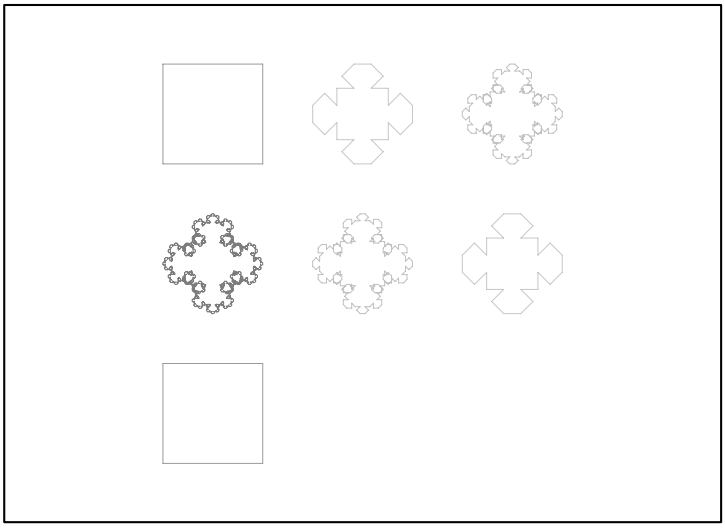


Abb.
ani42_05.eps

6. Figur

```
> repeatrule:=x->subs(f=(f,m,m,f,p,f,p,f,p,f,p,f,m,m,f),x):
> phi:=evalf(Pi/3):
  star1([f,p,f,p,f,p,f,p,f,p,f,p],repeatrule,phi,3,1):
  star1([f,p,f,p,f,p,f,p,f,p,f,p],repeatrule,phi,3,0);
```

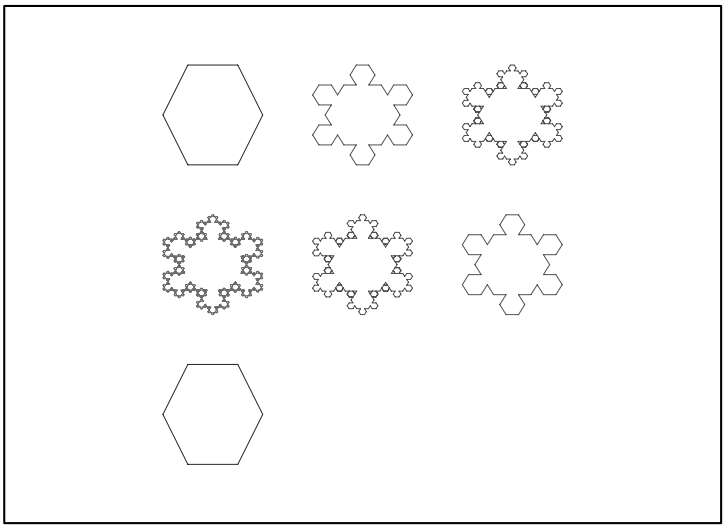


Abb.
ani42_06.eps

Bäume

Zunächst vereinfachen wir das Parameterkonzept für die Ausgabe in der Prozedur *star* und notieren die modifizierte Version *star2*.

```
> pfad := `D:/Neundorf/Maple2/`;
  name1 := `ani43_01.gif`;
  name2 := `ani43_02.gif`;

  datei1 := cat(pfad,name1);
  datei2 := cat(pfad,name2);

> star2:=proc(structure::list,repeatrule::procedure,
              level::integer, nr::integer, ind::integer)
  # list       - Liste fuer Struktur
  # repeatrule - Abbildungsregel
  # level      - Anzahl der Regelanwendungen auf Grundstruktur

  local i,j,k,plant,pts,lst,tt,xx,yy,q,qs,qq;

  for j from 0 to level
  do
    lst:=NULL;
    plant:=structure;
    if j>0 then
      for i from 1 to j do plant:=map(repeatrule,plant) od;
    fi;
    xx:=0; yy:=0; tt:=1/3*Pi; pts:=[xx,yy];

    for i in plant
    do
      if i=p then tt:=tt+1/3*Pi;
        elif i=m then tt:=tt-1/3*Pi
        elif i=f then xx:=xx+cos(tt);
        yy:=yy+sin(tt);
        pts:=pts,[xx,yy]
      fi
    od;
    q[j]:=plot({lst,[pts]},color=blue):
  od;

  qs:=[seq(q[j],j=0..level)]:
  q:=plots[display](qs,axes=none,scaling=constrained,insequence=true):
  plots[display](q)

  end:
```

Nun qualifiziert man die Prozedur *star2* in der Weise, dass man nicht nur Strecken aneinanderfügen kann, sondern auch zu bisher gezeichneten Punkten zurückspringen kann (solche Punkte muss man dann während der Rechnung speichern), so ergeben sich interessante Baumstrukturen. Die notwendige Liste *data* definieren wir als globale Variable.

```
> tree:=proc(structure::list,repeatrule::procedure,
             level::integer, theta::realcons)

  # list       - Liste fuer Struktur
  # repeatrule - Abbildungsregel
  # level      - Anzahl der Regelanwendungen auf Grundstruktur
  # theta      - Drehwinkel fuer Zweige

  local i,j,lst,plant,pts,stack,tt,xx,yy,xyt,q;

  plant:=structure;
  for i to level do plant:=map(repeatrule,plant) od;
  xx:=0; yy:=0; tt:=theta;
  lst:=NULL;
  stack:=[];
  pts:=[xx,yy];

  for i in plant
  do
    if i=p then tt:=tt+theta;
      elif i=m then tt:=tt-theta;
      elif i=l then stack:=[[xx,yy,tt],op(stack)];
      elif i=r then
      if nops(stack) < 1 then ERROR('StackProblem'); fi;
      xyt:=stack[1];
      stack:=subsop(1=NULL,stack);
      xx:=xyt[1]; yy:=xyt[2]; tt:=xyt[3];
      lst:=lst,[pts];
      pts:=[xx,yy];
      elif i=f then
      xx:=xx+cos(tt);
      yy:=yy+sin(tt);
      pts:=pts,[xx,yy];
    fi
  od;
  plot({lst,[pts]},axes=none,scaling=constrained):
  end:


> bild:=proc(repeatrule::procedure, level::integer, theta::realcons)

  local j,q;
  global data;

  for j from 1 to level do
    q[j]:=tree([f],repeatrule,j,theta):
    if (j=1) then data:=q[j] else data:=q[j],data,q[j] fi:
  od;
  data:=[data]:
  end:
```

1. Baum: gerade

Abbildungsregel

```
> rr:=x->subs(f=(f,l,p,f,r,f,l,m,f,r,f),x):
```

Baum

```
> level:=4: theta:=Pi/7: bild(rr,4,Pi/7):
```

Äußerer am meisten verzweigter Baum

```
> plots[display](data[1]);
```

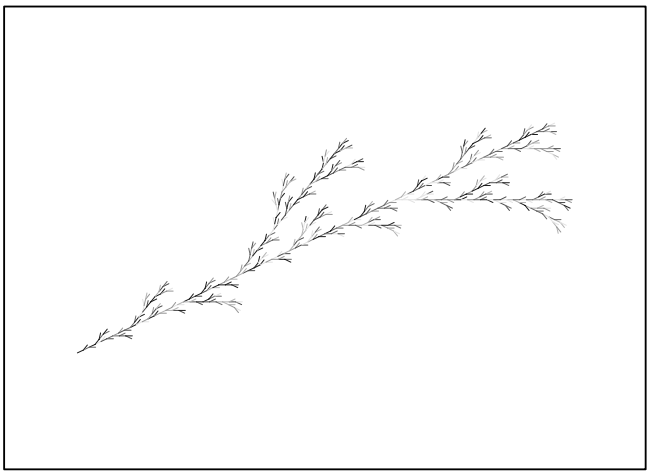


Abb. ani43_01.eps

Ansicht aller Teilbäume als Tableau

```
> qq:=array(1..nops(data),[]):
  for k from 1 to nops(data) do
    qq[k]:=plots[display](data[k],axes=none):
  od:
  plots[display](qq);
```

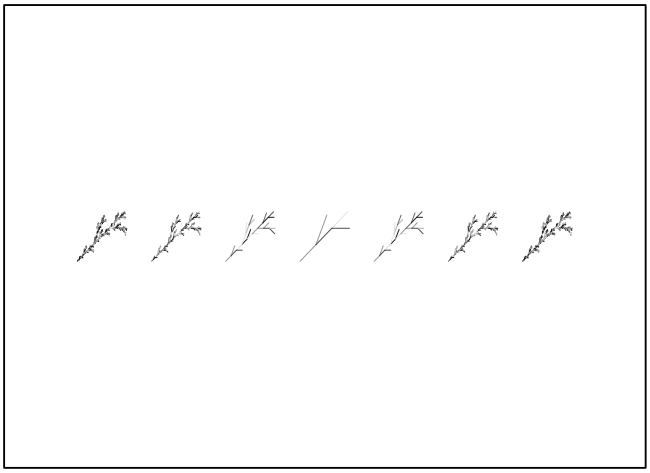


Abb. ani43_02.eps

Alle Bäume übereinander

```
> plots[display](data);
```

Die Abbildung ist hier wie beim File ani43_01.eps.

Alle Bäume nebeneinander

```
> qd:=plots[display](data,scaling=constrained,insequence=true):
  plots[display](qd);
```

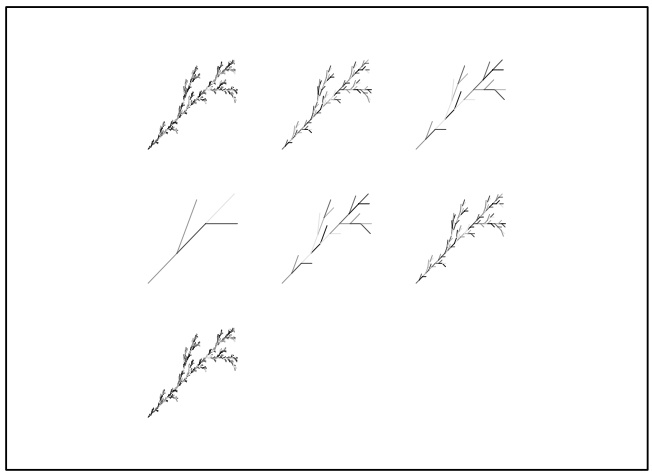


Abb.
ani43_04.eps

Animation

```
> plots[display](data,insequence=true,thickness=3);
```

GIF Animated Transparent File

```
> interface(plotdevice=gif,
      plotoutput=datei1,
      plotoptions=`width=200,height=200,
        transparent=true`);
  plots[display](data,insequence=true,thickness=3);
  interface(plotdevice=win);
```

2. Baum: spiralförmig [15]

Abbildungsregel

```
> rr:=x->subs(f=(f,l,p,f,r,f,l,m,f,r,f,l,f,m,r,f,p),x):
```

Baum

```
> bild(rr,3,Pi/7):
```

Alle Spiralbäume übereinander

```
> plots[display](data);
```

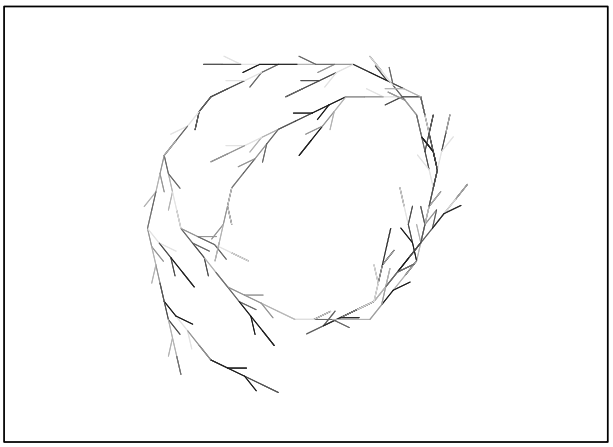


Abb.
ani43_05.eps

Alle Spiralbäume nebeneinander

```
> qd:=plots[display](data,scaling=constrained,insequence=true):
  plots[display](qd);
```

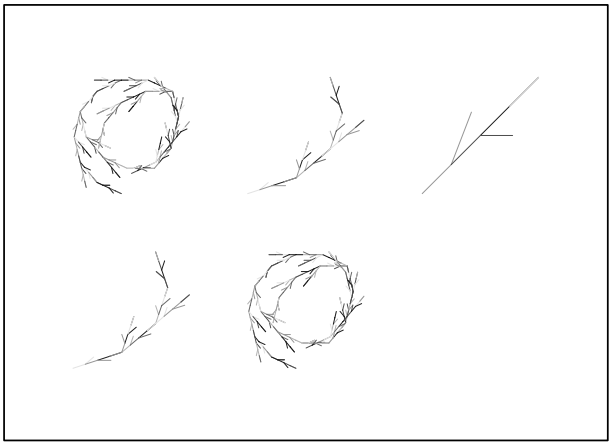


Abb.
ani43_06.eps

Animation

```
> plots[display](data,insequence=true,thickness=3);
```

GIF Animated Transparent File

```
> interface(plotdevice=gif,
      plotoutput=datei2,
      plotoptions=`width=200,height=200,transparent=true`);
  plots[display](data,insequence=true,thickness=3);
  interface(plotdevice=win);
```

Julia-Menge

Diese Fraktale sind drachenähnliche Figuren. Ihre Konstruktion basiert auf der Iterations-Theorie von P. FATOU und G. JULIA (Fixpunktiteration) sowie der nichtlinearen Rekursivität [16].
Betrachten wir die komplexe Funktion $f: z \to z^2 + \alpha$. Die Julia-Menge ist die Menge aller z der komplexen Ebene, bei der die Folge $f(z),\ f(f(z)),\ f(f(f(z))), \ldots$ nicht gegen ∞ tendiert. Für verschiedene Werte α erhält man unterschiedliche Muster.
Zunächst verständigen wir uns auf das Farbkonzept für die Bildsequenz.
Im zweiten Schritt werden die Julia-Mengen als Animationsfolge erstellt. Wir zeichnen eine solche Menge punktweise und lassen sie dann einfach schrittweise wachsen. Dieser Entwurf erfordert nicht geringe Rechenzeit (Arbeitsblatt *ani44_.mws*).

```
> pfad := `D:/Neundorf/Maple2/`;
  name1 := `ani44_1.ps`;
  name2 := `ani44_2.ps`;
  name3 := `ani44_3.ps`;
  name4 := `ani44_4.ps`;
  name5 := `ani44_5.gif`;
  name6 := `ani44_6.gif`;

  datei1 := cat(pfad,name1);
  datei2 := cat(pfad,name2);
  datei3 := cat(pfad,name3);
  datei4 := cat(pfad,name4);
  datei5 := cat(pfad,name5);
  datei6 := cat(pfad,name6);
```

Farbtest

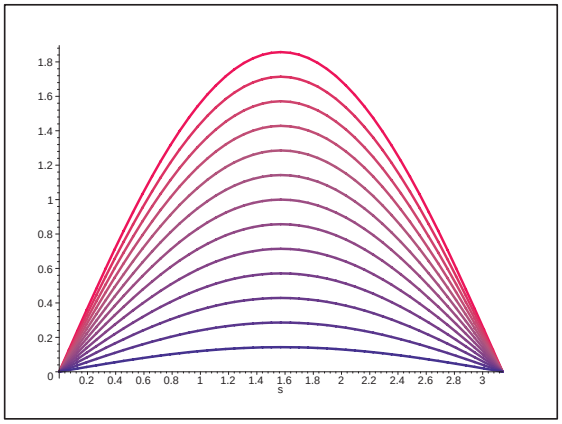


Abb.
ani44_1.ps

```
> for i from 1 to 13 do
    VV:=plot(i/7*sin(s),s=0..Pi,
         color=COLOR(RGB,i/13,abs(1,0.5-exp(1/i)),1-i/26),thickness=5):
    if i=1 then dl:=VV else dl:=dl,VV fi:
  od:
  display([dl]);
```

```
> interface(plotdevice=ps,
      plotoutput=datei1,
      plotoptions='color,portrait,width=400,height=400');
  plots[display]([dl]);
  interface(plotdevice=win);
```

Julia-Mengen

```
> c1:=-0.08:
  c2:= 0.74:
  for t from 1 by 1 to 13 do
   data:=[100,100]:
   for m from 20 to 180 do
     x0:=evalf(-2+m/50):
     for n from 60 to 540 do
       y0:=evalf(2-n/150):
       x:=evalf(x0):
       y:=evalf(y0):
       z:=0:
       i:=1:
       do
       x1:=evalf(x*x-y*y+c1):
       y1:=evalf(2*x*y+c2):
       x:=x1:
       y:=y1:
       z:=evalf(x*x+y*y):
       if (evalf(z)>4 or evalf(i)>=20) then break; fi;
       i:=i+1:
       od:
       if evalf(z)<=4 then data:=data,[m,n]:  fi:
     od:
   od:
   A0:=plot([data],style=point,symbol=point,
          color=COLOR(RGB,t/13,abs(1,0.5-exp(1/t)),1-t/26)):
   display(A0);
   if (t=1) then data2:=A0 else data2:=A0,data2,A0 fi:
   c2:=c2+1/200:
   c1:=c1+1/200:
  od:
```

3 Varianten der Animation

```
> display([data2],insequence=true,axes=none);
> plots[display]([data2],insequence=true,axes=none);
> pl1:=display([data2],insequence=true,axes=none):
  pl1;
```

Einzelbilder

```
> plots[display](data2[1],insequence=true,axes=none);
> plots[display](data2[13],insequence=true,axes=none);
> display([data2],axes=none); # Ansichten uebereinander
```

Ansicht aller ungeraden Entwicklungsstufen als Tableau

```
> pp:=array(1..2,1..4,[]):
  for j from 1 to 4 do
    pp[1,j]:=display(data2[2*j-1]):
    pp[2,j]:=display(data2[2*j+5]):
  od:
> plots[display](pp);
```

PS File

```
> interface(plotdevice=ps,
      plotoutput=datei2,
      plotoptions=`color,portrait,width=160,height=420`);
  plots[display](data2[1],axes=none);
  interface(plotdevice=win);
> interface(plotdevice=ps,
      plotoutput=datei3,
      plotoptions=`color,portrait,width=160,height=420`);
  plots[display](data2[13],axes=none);
  interface(plotdevice=win);
```

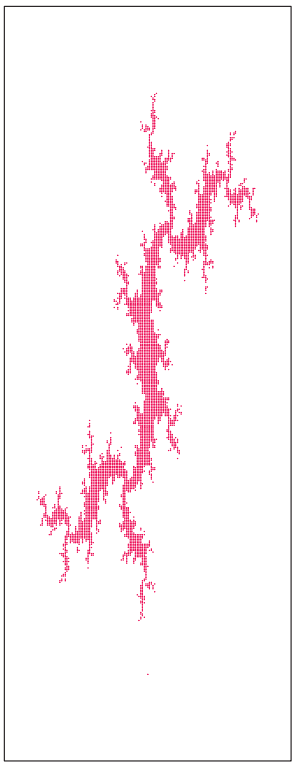


Abb. ani44_2.ps

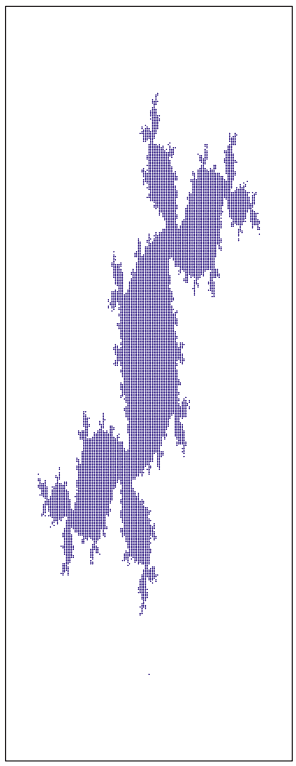


Abb. ani44_3.ps

```
> interface(plotdevice=ps,
      plotoutput=datei4,
      plotoptions=`color,portrait`);
  plots[display](pp,axes=none);
  interface(plotdevice=win);
```

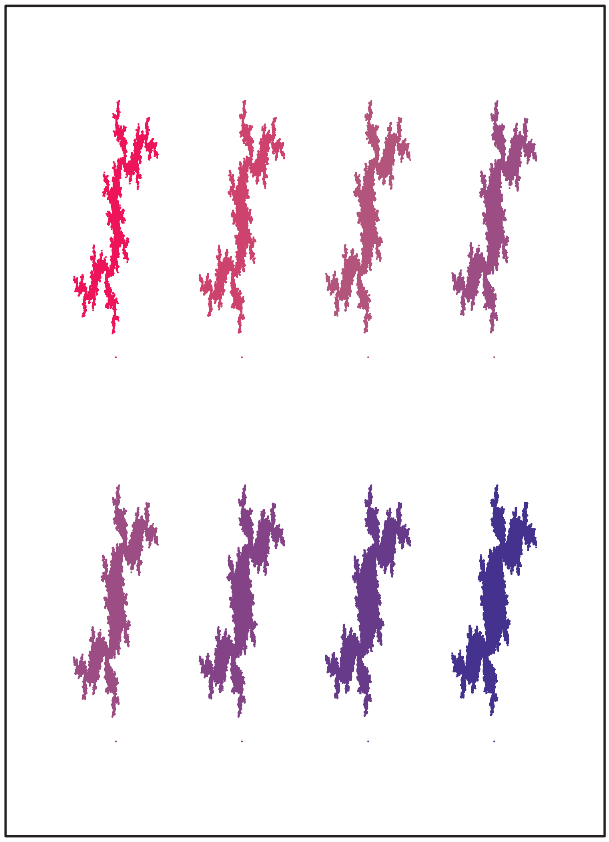


Abb.
ani44_4.ps

GIF Animated Transparent File
1.Juliamenge

```
> interface(plotdevice=gif,
      plotoutput=datei5,
      plotoptions=‘transparent=true,width=160,height=420‘):
  display(data2[1],insequence=true,axes=none);
  interface(plotdevice=win);
```

Animation

```
> interface(plotdevice=gif,
      plotoutput=datei6,
      plotoptions=‘transparent=true,width=160,height=420‘):
  display([data2],insequence=true,axes=none);
  interface(plotdevice=win);
```

Die Animation auf der Homepage kann dann wie folgt ablaufen.
Es wird die Julia-Menge des 1.Bildes ani44_5.gif zunächst nur angezeigt. Sobald man mit der Maus in diesen Bildbereich geht, läuft die Animation ab, also die Bildfolge ani44_6.gif.

8 Anhang

Anhang A

Maple-Pakete zur Grafik und mit Grafikkomponenten

Pakete mit ihren Routinen werden durch das Kommando `with` eingebunden.

```
> restart:
> with(plots);

  [animate, animate3d, animatecurve, changecoords, complexplot,
   complexplot3d, conformal, contourplot, contourplot3d,
   coordplot, coordplot3d, cylinderplot, densityplot, display,
   display3d, fieldplot, fieldplot3d, gradplot, gradplot3d,
   implicitplot, implicitplot3d, inequal, listcontplot,
   listcontplot3d, listdensityplot, listplot, listplot3d,
   loglogplot, logplot, matrixplot, odeplot, pareto, pointplot,
   pointplot3d, polarplot, polygonplot, polygonplot3d,
   polyhedra_supported, polyhedraplot, replot, rootlocus,
   semilogplot, setoptions, setoptions3d, spacecurve,
   sparsematrixplot, sphereplot, surfdata, textplot, textplot3d,
   tubeplot]

> with(plots,animate);
  [animate]

> with(plottools);
  [arc, arrow, circle, cone, cuboid, curve, cutin, cutout, cylinder,
   disk, dodecahedron, ellipse, ellipticArc, hemisphere,
   hexahedron, homothety, hyperbola, icosahedron, line,
   octahedron, pieslice, point, polygon, project, rectangle,
   reflect, rotate, scale, semitorus, sphere, stellate,
   tetrahedron, torus, transform, translate, vrml]

> with(DEtools);
  [DEnormal, DEplot, DEplot3d, DEplot_polygon, DFactor, Dchangevar,
   GCRD, LCLM, PDEchangecoords, RiemannPsols, abelsol, adjoint,
   autonomous, bernoullisol, buildsol, buildsym, canoni,
   chinisol, clairautsol, constcoeffsols, convertAlg, convertsys,
   dalembertsol, de2diffop, dfieldplot, diffop2de, eigenring,
   endomorphism_charpoly, equinv, eta_k, eulersols, exactsol,
   expsols, exterior_power, formal_sol, gen_exp, generate_ic,
   genhomosol, hamilton_eqs, indicialeq, infgen, integrate_sols,
   intfactor, kovacicsols, leftdivision, liesol, line_int,
   linearsol, matrixDE, matrix_riccati, moser_reduce, mult,
   newton_polygon, odeadvisor, odepde, parametricsol,
```

```
   phaseportrait, poincare, polysols, ratsols, reduceOrder,
   regular_parts, regularsp, riccati_system, riccatisol,
   rightdivision, separablesol, super_reduce, symgen,
   symmetric_power, symmetric_product, symtest, transinv,
   translate, untranslate, varparam, zoom]

> with(PDEtools);

  [PDEplot, build, charstrip, dchange, difforder, mapde, splitstrip,
   splitsys]
```

Anhang B

Zusammenstellung von wichtigen Adressen

1. World Wide Web (WWW) mit Maple-Seiten

http://daisy.uwaterloo.ca/	Maple from Symbolic Computation Group (MAPLE resources, Univ. Waterloo, Ontario CA)
http://daisy.uwaterloo.ca/~khare/numeric/fractal.html	K.Hare Maple for Numerics: Roots, Fractals
http://www.maplesoft.com	Maple WWW Home Page (Waterloo Maple Inc.)
http://www.scientific.de/maple/index.html	Deutsche Maple Home Page
http://SunSite.informatik.rwth-aachen.de/maple/mplgesch.htm	Maple an der RWTH Aachen (Geschichte)
http://SunSite.informatik.rwth-aachen.de/maple/frame09.htm	Einführung in Maple V
http://SunSite.informatik.rwth-aachen.de/maple/mpllinks.htm	Maple Links
http://www.mthcsc.wfu.edu/maple/tutorial.html	Maple Tutorial, Wake Forest University
http://www.uni-karlsruhe.de/~Maple/	Maple Seite der Uni Karlsruhe
http://www.uni-essen.de/hrz/mathe/maple/	Maple Seite der Uni Essen
http://www.minet.uni-jena.de/~schmitzm/maple/index.html	Mathematik Didatik mit Maple an Uni Jena
http://www.maplesoft.com/CyberMath/index.html	Maple
http://userwst1.fh-reutlingen.de/dimkomma/index.html	Maple am Isolde-Kurz-Gymnasium Reutlingen
http://www.kp.tue.bw.schule.de/jsindex.htm	Maple am Kepler-Gymnasium Tuebingen
http://www.elektronikschule.de/dbg/lehrer/ritters/mathe/maple/mapind.htm	Maple-Worksheet von A.Rittershofer EST Tettnang
http://daisy.uwaterloo.ca/	Maple Developers' Home Page
http://saaz.lanl.gov/maple/Maple_Home.html	A Maple Tutorial (The Cluster Team Presents)
http://saaz.lanl.gov/maple/Maple_Page13.html	Maple V on the Web
http://web.mit.edu/afs/athena/software/maple/www/home.html	Maple at MIT (Cambridge MA, USA)
http://www-math.cc.utexas.edu/math/Maple/	Maple index page, UT Austin
ftp://ftp.unibe.ch/pub/Maple/algcurve/	AlgCurve Package maintained at University Bern
ftp://galois.maths.qmw.ac.uk/pub/mapledoc	Maple Introduction
ftp://ftp.hprc.utoronto.ca/pub/ednet/maths/maple/readme.html	Experiments using Maple
http://krum.rz.uni-mannheim.de/cafgbench.html	Computer Algebra Benchmarks (RZ/Uni Mannheim)

2. Verzeichnisse mit Maple-Arbeitblättern und Maple-Files

Zu den Kapiteln 1-7 des Skripts liegen die entsprechenden Arbeitsblätter **.mws* und diverse Daten- und Grafikfiles vor. Die dortigen Verzeichnispfade sind i.a. anzupassen.

```
graph22.mws, ani0_.mws, expo2.mws, exam2.mws, reihe2.mws, ani1_mws,
ani2_.mws, ani31_.mws, ani32_.mws, ani33_.mws, ani34_.mws,
ani41_.mws, ani42_.mws, ani43_.mws, ani44_.mws,
*.res, *.pcx, *.gif, *.ps, *jpg, *.eps
```

Die Dateien sind zu finden im Novell-Netz PIVOT des Instituts für Mathematik bzw. auf der persönlichen Homepage im Internet.

\\PIVOT\SHARE Q:\NEUNDORF\STUD_M93\MAPLE2

Homepage Navigator → Publications → Computeralgebra → Maple2

e-mail: *neundorf@mathematik.tu-ilmenau.de*

Homepage:

http://imath.mathematik.tu-ilmenau.de/∼neundorf/index_de.html

Literatur

[1] Monagan, M.: Programming in Maple: The Basics.
Institut für Wissenschaftliches Rechnen ETH-Zentrum, CH-8092 Zürich.

[2] Monagan, M. et al.: Maple V Programming Guide. Springer New York 1996.

[3] Klimek, G. und Klimek, M.: Discovering Curves and Surfaces with Maple. Springer-Verlag New York 1997.

[4] Kofler, M.: Maple V Release 4. Addison Wesley Bonn 1996.

[5] Walz, A.: Maple V: Rechnen und Programmieren mit Rel.4. Oldenbourg München 1998.

[6] Westermann, T.: Mathematik für Ingenieure mit Maple. Springer-Verlag Berlin 1996.

[7] Werner, W.: Mathematik lernen mit Maple, Bd. 1,2. Ein Lehr-und Arbeitsbuch für das Grundstudium. dpunkt Heidelberg 1996, 1998 (CD).

[8] Char, B. et al.: Maple V Language Reference Manual. Springer-Verlag 1991.

[9] Release Notes for Maple V Release 3. Waterloo Maple Software.

[10] First Leaves: Tutorial Introduction to Maple. Springer-Verlag.

[11] Waterloo Maple Inc.: Maple V Student Version Release 4.
Springer-Verlag Berlin 1996 (CD Windows/Mac).

[12] Neundorf, W.: Programming in Maple V Release 5. Extended Basics.
Preprint M 07/99 IfMath der TU Ilmenau, Februar 1999.

[13] Neundorf, W.: Mathematica - PostScript - TEX.
Preprint M 07/96 IfMath der TU Ilmenau, April 1996.

[14] Neundorf, W.: Vorlesungsskript Numerische Mathematik, IfMath TUI 1999 (in TEX).
Teil 5: Nichtlineare Gleichungen und Gleichungssysteme,
Teil 6: Polynome, Interpolation, Splines, Differentiation,
Teil 8: Approximation, Ausgleichsrechnung.

[15] Reinhardt, R.: Übungen zur Höheren Mathematik im Ingenieurwesen.
Anleitungen und Skripte unter Verwendung von Maple, IfMath TUI 1999-2000.

[16] Peitgen, H.-O., Jürgens, H. und Saupe, D.: Fractals for the Classroom I,II.
Springer-Verlag New York 1992.

[17] Nicolaides, R.A. und Walkington, N.J.: Maple: A Comprehensive Introduction. Cambridge University Press 1996.

Anschrift:

Dr. Werner Neundorf, DM Brigitte Walther
Technische Universität Ilmenau, Institut für Mathematik
PF 10 0565
D - 98684 Ilmenau

email: neundorf@mathematik.tu-ilmenau.de, brigitte.walther@mathematik.tu-ilmenau.de