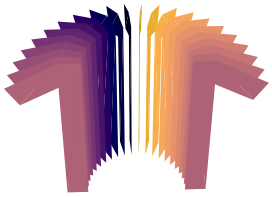
Kapitel 11

Definition eigener Funktionen



Nachdem im vorangegangenen Kapitel die in Maple bereits vordefinierten Funktionen im Mittelpunkt standen, geht es jetzt um die Definition eigener Funktionen. Maple stellt dazu verschiedene Mechanismen zur Verfügung, die je nach Anwendungsbereich mehr oder weniger gut geeignet sind. Wir werden die einfacheren Mechanismen in diesem Kapitel besprechen. Die weitergehenden Programmiermöglichkeiten werden ab Kapitel 28 behandelt.

Terme und Funktionen

Maple erlaubt es, Terme mit freien Variablen zu bilden und diese an einen Namen zu binden. Zur Termbildung können die vorhandenen Funktionen und Symbole in der üblichen Weise verwendet werden, etwa: `f:=x^2-3*x`. Maple kann mit solchen Termen symbolisch rechnen und sie in vielfältiger Weise umformen. Will man für die freien Variablen Werte einsetzen und für diese Ersetzung den Term auswerten, so verwendet man am besten den Befehl `eval` mit einer passenden Nebenbedingung.

```
f:=x^2-3*x;
```

$$f := x^2 - 3\,x$$

```
f1:=2*diff(f,x);
```

$$f1 := 4\,x - 6$$

```
eval(f1,x=3);
```

$$6$$

Will man diese Terme zur Definition von Funktionen verwenden, so benötigt man Verfahren, um diejenigen freien Variablen zu kennzeichnen, die als Argumente der Funktion verwendet werden sollen. Das einfachste Verfahren lehnt sich an die in der Mathematik übliche Beschreibung von Funktionen in der Form

$$f : x \mapsto Term$$

an, wobei die Variable x im Term frei (oder nicht, falls die Funktion konstant ist) vorkommt.

Maple stellt hierfür den Pfeiloperator -> zur Verfügung. Mit ihm lassen sich auch mehrstellige Funktionen in der folgenden Form definieren:

```
(variablensequenz) -> term;
```

Das Ergebnis dieser Definition ist eine Funktion mit den in der Variablensequenz angegebenen Parametern und einer durch den Term festgelegten Zuordnungsvorschrift. Funktionen in Maple sind 'First-Class-Objekte', das heißt, sie können wie alle anderen Objekte an Symbole gebunden, als Funktionsparameter an andere Funktionen übergeben und sogar als Funktionsergebnisse zurückgegeben werden. Ihre Verwendung ist durch nichts eingeschränkt. Funktionen werden auf Argumente angewendet, indem diese hinter der Funktion aufgeschrieben werden, eingeschlossen von einem Klammerpaar und durch Kommata getrennt.

Bei der Auswertung eines Funktionsaufrufs bindet Maple zunächst (und *nur* für diesen Aufruf) die Parameter der Funktion an die übergebenen Argumente. Dann erst wird der Funktionsterm mit diesen Bindungen ausgewertet.

Dieses Schema wird anhand der Beispiele rechts demonstriert. Die Funktionen werden sinnvollerweise bei ihrer Definition an ein Symbol (hier f) gebunden und dann über dieses Symbol aufgerufen. Die Differentiation lässt sich über den Term der Zuordnungsvorschrift durchführen (mit `diff`) oder mit dem Differentiationsoperator `D`. Im ersten Fall ist das Ergebnis ein Term, im zweiten Fall eine Funktion.

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

$$f := x \to x^2 - 3\,x$$

```
f(x), f(sin(y)), f(25.77);
```

$$x^2 - 3\,x, \sin(y)^2 - 3\,\sin(y), 586.7829$$

```
diff(f(x),x);
```

$$2\,x - 3$$

```
D(f);
```

$$x \to 2\,x - 3$$

g hat drei Parameter, weswegen die Parameterliste geklammert werden muss.

```
g:= (x,y,z)->(x^y)^z;
```

$$g := (x, y, z) \to x^{y^z}$$

```
g(a,b,c), g(2,3,4);
```

$$a^{b^c}, 4096$$

Funktionen, deren Wertemengen Zahlen sind, können mit den üblichen arithmetischen Operatoren verknüpft werden. Die Verkettung wird durch den Operator @ dargestellt.

```
f:=x->x^2: g:=y->y^3:
(f+2*g)(z), (f*g)(z), (f@g)(z);
```

$$z^2 + 2\,z^3, z^5, z^6$$

Beachten Sie, dass es keine Rolle spielt, ob die Funktion mit x oder mit y definiert wird – diese Zeichen stehen nur als Platzhalter für den Parameter der Funktion und sind in der Funktion nicht frei. `(f@g)(z)` entspricht $f(g(z))$. Bei der Verschachtelung von Funktionen muss natürlich auf die richtige Anzahl der Parameter geachtet werden – wenn f nur ein

skalares Ergebnis liefert, g aber zwei Parameter erwartet, kommt es zu einer Fehlermeldung.

Funktionen können auch so definiert werden, dass sie mehrere Ergebnisse als Folge angeben. Im letzten Beispiel wird f in g eingesetzt. Umgekehrt wäre das nicht möglich, weil f nur einen Parameter erwartet, g aber eine Folge von zwei Parametern liefert.

```
f:=x->(x,1+1/x);
```

$$f := x \rightarrow x, 1 + x^{-1}$$

```
g:=(x,y)->(x^2/y, y^2/x);
```

$$g := (x, y) \rightarrow \frac{x^2}{y}, \frac{y^2}{x}$$

```
f(3), (g@f)(3);
```

$$3, 4/3, \frac{27}{4}, \frac{16}{27}$$

Natürlich ist man nicht darauf beschränkt, Funktionen zu definieren, die als Werte und Argumente Zahlen haben. Die folgende Funktion liefert zu einer eingegebenen Funktion eine Wertetabelle im Bereich von -3 bis 3 mit Schrittweite 1 in Form einer Liste von Paaren aus Argument und Funktionswert.

```
wertetab:=f->[seq([i,f(i)],i=-3..3)];
```

$$wertetab := f \rightarrow [seq([i, f(i)], i = -3..3)]$$

Die Funktion unapply

Es gibt Probleme, wenn Ergebnisse aus einer Berechnung in einer neuen Funktionsvorschrift gespeichert werden sollen. Eine unmittelbare Zuweisung scheitert daran, *dass Maple den hinter dem Pfeiloperator angegebenen Ausdruck nicht auswertet*, sondern in unveränderter Form speichert.

```
f:=x->1/(1+x^2):
f1:=x->diff(f(x),x);
```

$$f1 := x \rightarrow diff(f(x), x)$$

```
f1(1);
    Error, (in f1) wrong number
    (or type) of parameters in
    function diff;
```

Bei einem Funktionsaufruf bindet Maple zunächst die Parameter der Funktion an die übergebenen Argumente. In unserem Fall wird also x an 1 gebunden. Der auszuwertende Term ist aber `diff(f(x), x)`. Durch die Ersetzung steht statt einer Variablen jetzt für die zu berechnende Ableitung die Zahl 1 und damit erhalten wir die obige Fehlermeldung.

Es gibt zwei Lösungsmöglichkeiten. Zum einen können wir den Differentiationsoperator `D` verwenden. Weitergehend ist die Verwendung der Funktion `unapply`, die als Eingabe einen Term und ein oder mehrere Symbole (ungebunden) erwartet, den Term auswertet und dann als Zuordnungsvorschrift für eine Funktion mit den angegebenen Symbolen als Parameter verwendet.

```
f1:=unapply( diff(f(x),x) ,x);
```

$$f1 := x \to -\frac{2\,x}{(1+x^2)^2}$$

```
f2:=D(f);
```

$$f2 := x \to -\frac{2\,x}{(1+x^2)^2}$$

```
f1(1);
```

$$-1/2$$

`unapply` kann auch für mehrparametrische Funktionen verwendet werden. Die Funktion wird nur für jene Parameter definiert, die in `unapply` angegeben werden (also im ersten Beispiel rechts nur für x, y und z, nicht aber für a und b).

```
unapply(a*x^2+b*y^2+c*z^2,x,y,z);
```

$$(x, y, z) \to ax^2 + by^2 + cz^2$$

```
unapply( [1/x,1/(x+y),1/y], x,y);
```

$$(x, y) \to [x^{-1}, (x+y)^{-1}, y^{-1}]$$

Anonyme Funktionen

In den obigen Beispielen wurden die Funktionsvorschriften durch Zuweisungen an eine Variable gebunden. In vielen Maple-Funktionen muss als Parameter eine Funktion angegeben werden. Wie oben gesehen, sind Funktionen in Maple First-Class-Objekte. Sie können daher auch direkt ohne Bindung an einen Namen verwendet werden. Man spricht in einem solchen Fall auch von 'anonymen' Funktionen.

Im Beispiel rechts wird die Liste in $data2$ durch die Anwendung der Funktionsvorschrift Quadrieren, also $x \mapsto x^2$ mit dem Kommando `map` aus den Daten in $data1$ erzeugt.

```
data1:=[1,2,3,4,5]:
data2:=map(x->x^2, data1);
```

$$data2 := [1, 4, 9, 16, 25]$$

Aus den beiden Listen wird nun mit `zip` eine weitere, verschachtelte Liste gebildet, indem die Funktionsvorschrift $(x, y) \mapsto [x, y]$ angewendet wird. (Weitere Informationen zu `map` und `zip` finden Sie in Kapitel 9.)

```
data3:= zip( (x,y)->[x,y],
             data1, data2);
```

$$data3 := [[1, 1], [2, 4], [3, 9], [4, 16], [5, 25]]$$

In den beiden Kommandos rechts wird zuerst eine Liste mit zehn Zufallszahlen erzeugt, anschließend werden daraus alle Werte kleiner 50 herausgefiltert. Als anonyme Funktion wird $x \mapsto (x < 50)$ verwendet. Diese Funktion liefert je nach Inhalt von x die Wahrheitswerte *true* oder *false*, die für `select` als Entscheidungsgrundlage zur Selektion der Listenelemente dienen.

```
data:= [seq(rand(100)(),i=1..10)];
```

$$data := [60, 82, 92, 13, 77, 49, 35, 61, 48, 3]$$

```
select( x->(x<50), data);
```

$$[13, 49, 35, 48, 3]$$

Die untenstehende Funktion `tangente` erwartet als Eingabe eine differenzierbare Funktion f und eine Zahl x_0. Sie gibt die Funktion der Tangente an das Schaubild von f im Punkt $(x_0|f(x_0))$ zurück.

```
tangente:=(f,x0)->unapply(D(f)(x0)*(x-x0)+f(x0),x);
```

$$tangente := (f, x0) \rightarrow unapply(D(f)(x0) * (x - x0) + f(x0), x)$$

```
tangente(x->x^2,2)
```

$$x \rightarrow 4x - 4$$

Definition von Funktionen durch Prozeduren

Auch wenn diese Tatsache hinter der Oberfläche von Maple normalerweise verborgen bleibt, stellt Maple Funktionen in der Syntax der Maple-internen Programmiersprache als Prozeduren dar. Dieselbe Syntax wird zur Realisierung der meisten Maple-Befehle verwendet.

```
lprint(x->x^2);
    proc (x) options operator, arrow;
    x^2 end
```

Aus dem obigen Beispiel geht die prinzipielle Syntax von Prozedurdefinitionen hervor: Die Prozedur beginnt mit dem Schlüsselwort `proc`, dem eine geklammerte Folge der Parameter folgt. Anschließend können verschiedene Optionen genannt und mit `local` lokale Variablen definiert werden. Der Hauptteil der Prozedur besteht hier einfach nur aus x^2, er kann aber erheblich umfangreicher sein und Fallunterscheidungen mit `if ... then ... end if` sowie Schleifen mit `for ...do ... end do` enthalten. Die Prozedur endet mit dem Schlüsselwort `end proc`.

Damit steht Ihnen neben der Pfeilschreibweise und `unapply` eine dritte Variante zur Formulierung von Funktionen zur Verfügung. Die Definition von Funktionen mit `proc` ist aber nur dann sinnvoll, wenn Sie die zusätzlichen Möglichkeiten nutzen, die sich dadurch ergeben: etwa die automatische Überprüfung des Datentyps der Parameter oder die Option `remember`, damit sich die Funktion alle bereits berechneten Ergebnisse merkt und diese bei

einem späteren nochmaligen Aufruf nur noch einer Tabelle entnehmen muss. Generell ist der Einsatz von `unapply` und des Pfeiloperators in der Funktionsdefinition bei einfachen Funktionen angebracht, komplexere Funktionen sollten mit `proc` definiert werden.

Bei der Eingabe der Funktion müssen Sie die einzelnen Zeilen mit Shift + ↵ trennen und erst nach der letzten Zeile die Eingabe mit ↵ abschließen. Die Beispielfunktion `diffn` bildet die ersten n Ableitungen der übergebenen Funktion und gibt sie als Folge aus.

```
diffn:=proc(f,n)
  local i:
  seq((D@@i)(f), i=1..n);
end:
diffn(arctan, 4);
```

$$a \to \frac{1}{(1+a^2)}, \quad a \to -\frac{2\,a}{(1+a^2)^2}, \quad a \to \frac{8\,a^2}{(1+a^2)^3} - \frac{2}{(1+a^2)^2}, \quad a \to -\frac{48\,a^3}{(1+x^2)^4} + \frac{24\,a}{(1+a^2)^3}$$

Die Funktion `sort` sortiert Listen, wenn sie miteinander nicht vergleichbare Daten enthalten, nach der Maschinenadresse. Wir definieren eine Sortierfunktion, die Zahlen in sich der Größe nach und vor allen anderen Objekten anordnet, zudem Strings und Symbole lexikografisch nach den Zahlen anordnet und alle anderen Objekte nach Zahlen und Strings nach ihrer Maschinenadresse sortiert. Das erfordert mehrere Fallunterscheidungen. Diese werden in Maple durch die folgende Konstruktion der `if`–Anweisung realisiert:

```
if Bedingung then
   Anweisungsfolge 1
else
   Anweisungsfolge 2
end if
```

Bei Ausführung einer `if`–Anweisung wertet Maple die Bedingung aus und führt, falls dies den Wert `true` liefert, die Anweisungsfolge 1 aus, andernfalls die Anweisungsfolge 2. (Vor Maple6 muss die Anweisung mit dem Schlüsselwort `fi` beendet werden.)

Nun zu unserer Sortierfunktion. Sie selektiert aus der eingegebenen Liste jeweils die Zahlen, Strings und Symbole und schließlich den Rest der Liste in drei Teillisten, die an drei lokale Variablen gebunden werden. Diese Teillisten werden mit `sort` geordnet und dann wieder zu einer Liste zusammengesetzt.

```
sortreihe:=proc(liste)
   local nums,syms,rest;
   nums:=select(x->type(x,extended_numeric),liste);
   syms:=select(x->type(x,symbol) or type(x,string),liste);
   rest:=remove(x->(type(x,extended_numeric) or type(x,symbol) or type(x,string)),
                           liste);
   [ op(sort(nums)),op(sort(syms)),op(sort(rest))];
 end proc:

l:=[3,x,5,Pi,"Otto",u,sin(x),cos(x)+2]:
sort(l);sortreihe(l);
```

$$[3, 5, x, sin(x), u, "Otto", \pi, cos(x) + 2]$$

$$[3, 5, "Otto", \pi, u, x, sin(x), cos(x) + 2]$$

Hinweis: Die Möglichkeiten der Funktionsdefinition mit `proc` sind durch diesen Abschnitt nur angedeutet worden. Vertiefende Informationen finden Sie ab Kapitel 28 zum Thema Programmieren in Maple.

Stückweise zusammengesetzte Funktionen

Mit dem Kommando `piecewise` können Sie Funktionen stückweise zusammensetzen. Die Funktion erwartet ihre Parameter jeweils paarweise, wobei der erste Parameter eine Bedingung und der zweite Parameter eine Funktion ist, die dann gilt, wenn die Bedingung erfüllt ist. Durch einen zusätzlichen optionalen Parameter kann ein Defaultwert angegeben werden, der dann gilt, wenn keine der Bedingungen erfüllt ist.

Am ehesten wird der Umgang mit `piecewise` anhand eines Beispiels deutlich: Die Funktion f ist für x-Werte kleiner 0 mit 0 definiert, für x-Werte zwischen 0 und 1 mit der Funktion x, für x-Werte zwischen 1 und 2 mit dem Wert 1, für x-Werte zwischen 2 und 3 mit der Funktion $3 - x$ und für alle anderen x-Werte mit dem Wert 0.

```
f:=x->piecewise(x<0,2, x<1,x+2, x<2,3,
                x<3,5-x,2): f(x);
```

$$\begin{cases} 2 & x < 0 \\ x + 2 & x < 1 \\ 3 & x < 2 \\ 5 - x & x < 3 \\ 2 & \text{otherwise} \end{cases}$$

Das Beispiel zeigt auch die Verwendung der Funktion (numerische Auswertung für einen bestimmten Wert, Integration, Ableitung). Die Ableitung ist an einigen Stellen nicht definiert. (Das Schaubild hat an den Übergängen der Bereiche Knicke.)

```
f(2.6);
```

$$2.4$$

```
int(f(x), x=0..3);
```

$$8$$

```
fin:=unapply( int(f(x),x), x): fin(x);
```

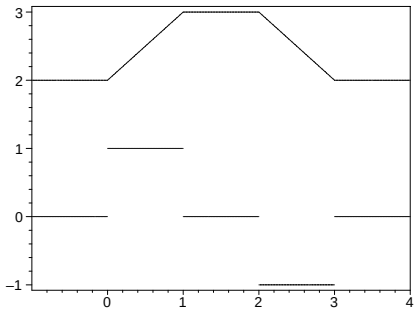
$$\begin{cases} 2x & x \le 0 \\ \frac{1}{2}x^2 + 2x & x \le 1 \\ 3x - \frac{1}{2} & x \le 2 \\ 5x - \frac{1}{2}x^2 - \frac{5}{2} & x \le 3 \\ 2x + 2 & 3 < x \end{cases}$$

```
f1:=D(f): f1(x);
```

$$\begin{cases} 0 & \text{x ¡ 0} \\ undefined & x = 0 \\ 1 & x < 1 \\ undefined & x = 1 \\ 0 & x < 2 \\ undefined & x = 2 \\ -1 & x < 3 \\ undefined & x = 3 \\ 0 & 3 < x \end{cases}$$

Die Abbildung rechts zeigt die Funktion f (oben) und ihre Ableitung *f1*. Die Option `discont=true` sorgt dafür, dass die Unstetigkeitsstellen von $f1$ korrekt gezeichnet werden.

```
plot( {f,f1},-1..4, axes=boxed,discont=true);
```



`convert(..,piecewise)` wandelt eine Funktionsdefinition in eine abschnittsweise Definition um, wenn die Funktion intern Fallunterscheidungen benötigt (z.B. wegen der Verwendung von `abs` oder `sig`). Im Beispiel rechts ergeben sich die Fallunterscheidungen aus denen der `abs`- und `signum`-Funktion.

```
convert(-signum(x)*abs(1-abs(x)),
        piecewise);
```

$$\begin{cases} -1 - x & x \le -1 \\ 1 + x & x < 0 \\ 0 & x = 0 \\ -1 + x & x < 1 \\ 1 - x & 1 \le x \end{cases}$$

Die folgenden Kommandos geben ein letztes Beispiel für die Vielseitigkeit von `piecewise`: Darin wird eine Funktion definiert, die einen Kreisbogen beschreibt. `convert(,..piecewise)` ersetzt die Bedingung $x^2 \le r^2$ durch Bedingungen, in die x als einfache Variable eingeht. Die `assume`-Eigenschaft für r ist Voraussetzung für diese Konversion.

```
assume(r>0); f:=unapply( piecewise(x^2<=r^2, sqrt(r^2-x^2),0), x): f(x);
```

$$f := \begin{cases} \sqrt{r^{\sim 2} - x^2} & x^2 \le r^{\sim 2} \\ 0 & otherwise \end{cases}$$

```
convert(f(x),piecewise,x);
```

$$\begin{cases} 0 & x \le -r^{\sim} \\ \sqrt{r^{\sim 2} - x^2} & x \le r^{\sim} \\ 0 & r^{\sim} < x \end{cases}$$

Syntaxzusammenfassung

```
x -> Term(x);
(x,y,..) -> Term(x,y,..);
```
definiert eine ein- bzw. mehrparametrische Funktion mit dem Pfeiloperator.

```
proc(x, y, ..)
  local l1,l2,..;
  ausdruck_1(x,y,..);
          :
  ausdruck_n(x,y,..);
end proc:
```
definiert eine mehrparametrische Funktion durch eine Prozedur. Eine detaillierte Beschreibung der Sprachelemente von Maple finden Sie in Kapitel 29.

```
f:=unapply(Term(x,y,..), {x,y,..});
```
erstellt aus dem angegebenen Term und den Variablen $x, y, ..$ eine Funktion mit den Variablen als Parametern und dem Term als Zuordnungsvorschrift. Im Gegensatz zum Pfeiloperator wird der Term zuvor ausgewertet. Aus `unapply(sin(x*y), x, y)` wird $(x, y) \rightarrow sin(xy)$.

```
piecewise(bed1, f1, bed2, f2, ..., bedn, fn, fdefault);
```
definiert einen Term mit Fallunterscheidungen. Er ist für alle x-Werte, die die Bedingung *bed1* erfüllen, durch $f1$ definiert, für alle x-Werte, die die Bedingung *bed2* erfüllen, durch $f2$ etc. Für x-Werte, die keine der angegebenen Bedingungen erfüllen, gilt die optionale Defaultfunktion *fdefault*.

```
convert(f, piecewise, x);
```
konvertiert f in einen stückweise definierten Term. Diese Konversion ist sinnvoll, wenn f Funktionen wie `abs`, `signum` oder `Heaviside` enthält.

```
convert(f, Heaviside);
```
konvertiert f in eine Funktion, in der `Heaviside` verwendet wird, um Unstetigkeitsstellen zu überbrücken. Diese Konversion ist vor allem für stückweise definierte Funktionen (`piecewise`) sinnvoll.