2. Erste Schritte mit MuPAD

Oft wird man ein Computeralgebra-System wie MuPAD interaktiv bedienen, d. h., man gibt eine Anweisung wie z. B. die Multiplikation zweier Zahlen an das System und wartet dann, bis MuPAD das Ergebnis berechnet hat und auf dem Bildschirm ausgibt.

Durch Aufruf des MuPAD-Programms wird eine *Sitzung* gestartet. Der Aufruf ist abhängig vom verwendeten Betriebssystem und der benutzten MuPAD-Version. Hierzu sei auf die entsprechenden Informationen der Installationsanleitung für MuPAD verwiesen. MuPAD stellt ein Hilfesystem zur Verfügung, mit dem man sich dann jederzeit in der laufenden Sitzung über Details von Systemfunktionen, ihre Syntax, die Bedeutung der zu übergebenden Parameter etc. informieren kann. Der Umgang mit der MuPAD-Hilfe wird im folgenden Abschnitt vorgestellt. Der Aufruf von Hilfeseiten wird zumindest in der Anfangsphase der wohl wichtigste Systembefehl sein, mit dem der Einsteiger umgehen wird. Es folgt ein Abschnitt über die Benutzung MuPADs als „intelligenter Taschenrechner": das Rechnen mit Zahlen. Dies ist vermutlich der einfachste und intuitivste Teil dieser Anleitung. Danach werden einige der Systemfunktionen für symbolische Rechnungen vorgestellt. Dieser Abschnitt ist wenig systematisch; er soll lediglich einen ersten Eindruck von den symbolischen Fähigkeiten des Systems vermitteln.

Das Ansprechen von MuPAD erfolgt nach Starten des Programms durch die Eingabe von Befehlen in der MuPAD-Sprache. Das System befindet sich im Eingabemodus, d. h., es wartet auf eine Eingabe, wenn das so genannte MuPAD-Prompt erscheint. Unter Windows oder auf dem Macintosh ist dieses Prompt das Zeichen •, unter UNIX ist es >>. Zur Illustration von Beispielen wird im Weiteren das UNIX-Prompt verwendet. Durch das Drücken der <Return>-Taste (unter Windows und UNIX) wird eine Eingabe beendet und das eingegebene Kommando von MuPAD ausgewertet. Die Tastenkombination <Shift> und <Return> kann dazu verwendet werden, eine neue Zeile anzufangen, ohne die aktuelle Eingabe zu beenden. Auf dem Macintosh muss zur Ausführung eines Befehls die <Enter>-Taste betätigt werden, die <Return>-Taste bewirkt dort nur einen Zeilenumbruch und MuPAD befin-

det sich weiterhin im Eingabemodus. In allen graphischen Benutzungsoberflächen kann man die Rollen von <RETURN> und <SHIFT>+<RETURN> vertauschen, indem man im Menü „Ansicht“ auf „Optionen“ klickt und dann „Shift+Return“ als „Evaluationstaste“ wählt.

Die Eingabe von

```
>> sin(3.141)
```

gefolgt von <RETURN> bzw. <ENTER> liefert auf dem Bildschirm das Ergebnis

$$0.0005926535551$$

Hierbei wurde die (allgemein bekannte) Sinus-Funktion an der Stelle 3.141 aufgerufen, zurückgegeben wurde eine Gleitpunktnäherung des Sinus-Wertes, wie sie auch – mit eventuell weniger Stellen – ein Taschenrechner geliefert hätte.

Es können mehrere Befehle in einer Zeile eingegeben werden. Zwischen je zwei Befehlen muss entweder ein Semikolon oder ein Doppelpunkt stehen, je nachdem, ob das Ergebnis des ersten Befehls angezeigt werden soll oder nicht:

```
>> diff(sin(x^2), x); int(%, x)
```

$$2\,x\,\cos\left(x^2\right)$$

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

Hierbei bedeutet `x^2` das Quadrat von `x`, die MuPAD-Funktionen `diff` bzw. `int` führen die mathematischen Operationen „differenzieren“ bzw. „integrieren“ durch (Kapitel 7). Der Aufruf von `%` steht für den letzten Ausdruck (also hier für die Ableitung von $\sin(x^2)$). Der `%` unterliegende Mechanismus wird in Kapitel 12 vorgestellt.

Schließt man einen Befehl mit einem Doppelpunkt ab, so wird dieser von MuPAD ausgeführt, das Ergebnis wird aber nicht auf dem Bildschirm angezeigt. So kann die Ausgabe von nicht interessierenden Zwischenergebnissen unterdrückt werden:

```
>> Gleichungen := {x + y = 1, x - y = 1}:
>> solve(Gleichungen)
```

$$\{[x = 1, y = 0]\}$$

Hierbei wird dem Bezeichner `Gleichungen` eine aus 2 Gleichungen bestehende Menge zugewiesen. Der Befehl `solve(Gleichungen)` (englisch: *to solve* = lösen) liefert die Lösung. Dem Gleichungslöser ist das Kapitel 8 gewidmet.

Eine MuPAD-Sitzung in der Terminalversion wird mit dem Schlüsselwort `quit` beendet:

```
>> quit
```

Bei den MuPAD-Versionen mit graphischer Bedienoberfläche funktioniert dies nicht, hier müssen Sie den entsprechenden Menüpunkt auswählen.

2.1 Erklärungen und Hilfe

Wenn Sie nicht wissen, wie die korrekte Syntax eines MuPAD-Befehls lautet, so können Sie die benötigten Informationen unmittelbar in der laufenden Sitzungen dem Hilfesystem entnehmen. Mit der Funktion `info` erhält man zu vielen MuPAD-Funktionen eine kurze englische Erklärung:

```
>> info(solve)
   solve -- solve equations and inequalities [try ?solve\
    for options]

>> info(ln)
   ln -- the natural logarithm
```

Detailliertere Informationen erhält man auf der *Hilfeseite* der entsprechenden Funktion. Diese kann mittels `help("Funktionsname")` aufgerufen werden. Hierbei muss der Name in Anführungszeichen gesetzt werden, da die `help`-Funktion Zeichenketten als Eingabe erwartet, welche in MuPAD durch `"` erzeugt werden (Abschnitt 4.11). Als Abkürzung für `help` dient der Operator `?`, bei dem keine Anführungszeichen benutzt zu werden brauchen:

```
>> ?solve
```

Die Hilfeseiten in MuPAD werden je nach verwendeter Version formatiert. Das folgende Beispiel zeigt eine Hilfeseite im ASCII-Format, wie sie von der Terminalversion MuPADs als Antwort auf `?solve` geliefert wird:

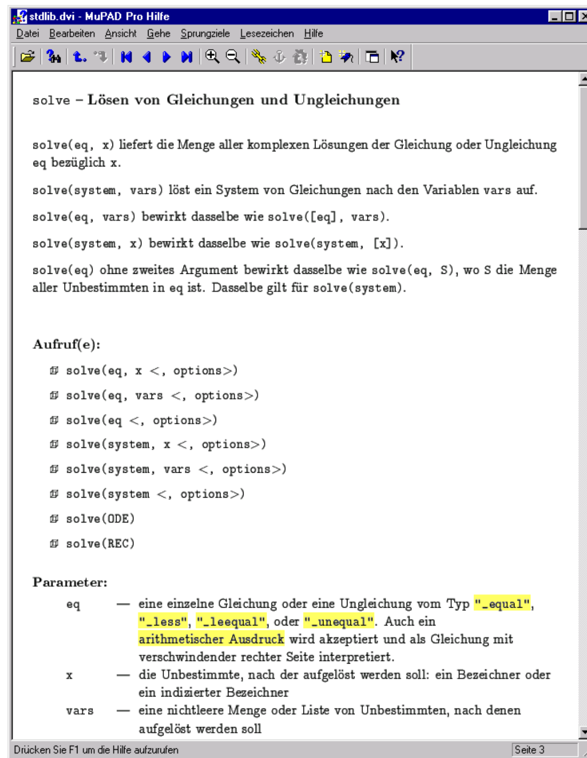
```
solve - Lösen von Gleichungen und Ungleichungen

Einführung

 solve(eq, x) liefert die Menge aller komplexen Lösungen der Gleichung
 oder Ungleichung eq bezüglich x.

 solve(system, vars) löst ein System von Gleichungen nach den Variablen
 vars auf.

 solve(eq, vars) bewirkt das selbe wie solve([eq], vars).

 solve(system, x) bewirkt das selbe wie solve(system, [x]).
```

```
 solve(eq) ohne zweites Argument bewirkt das selbe wie solve(eq, S), wobei S
 die Menge aller Unbestimmten in eq ist. Dasselbe gilt für solve(system).

Aufruf(e)

 solve(eq, x <, options>)
 solve(eq, vars <, options>)
 solve(eq <, options>)
 solve(system, x <, options>)
 solve(system, vars <, options>)
 solve(system <, options>)
 solve(ODE)
 solve(REC)

Parameter

 eq      -  eine einzelne Gleichung oder eine Ungleichung vom Typ
            "_equal", "_less", "_leequal", oder "_unequal". Auch ein
            arithmetischer Ausdruck wird akzeptiert und als Gleichung
            mit verschwindender rechter Seite interpretiert.
 x       -  die Unbestimmte, nach der aufgelöst werden soll: Ein
            Bezeichner oder ein indizierter Bezeichner
 vars    -  eine nichtleere Menge oder Liste von Unbestimmten, nach
            denen aufgelöst werden soll
 system  -  eine Menge, Liste, Tabelle oder ein Array von Gleichungen
            bzw. arithmetischen Ausdrücken. Letztere werden als
            Gleichungen mit verschwindender rechter Seite aufgefasst.
 ODE     -  eine gewöhnliche Differentialgleichung: Ein Objekt vom Typ
            ode.
 REC     -  eine Rekurrenzgleichung: Ein Objekt vom Typ rec.

...
```

Der Rest der Ausgabe wird aus Platzgründen weggelassen. Abbildung 2.1 zeigt einen Ausschnitt des entsprechenden Hypertext-Dokuments, welches bei Benutzung einer graphischen Oberfläche angezeigt wird.

Das Hilfesystem ist als so genanntes Hypertextsystem realisiert. Aktive Worte sind unterstrichen oder eingerahmt. Wenn Sie darauf klicken, erhalten Sie weitere Erklärungen zu diesem Begriff. Die Beispiele auf den Hilfeseiten kann man durch Anklicken der dazugehörigen unterstrichenen oder eingerahmten Prompts automatisch in das Eingabefenster von MuPAD übertragen. Auf Windows-Systemen verwenden Sie bitte einen Doppelklick oder „*drag & drop*“.

Aufgabe 2.1: Informieren Sie sich über die Anwendungsweise des MuPAD-Differenzierers `diff`! Berechnen Sie die fünfte Ableitung von `sin(x^2)`!



stdlib.dvi - MuPAD Pro Hilfe

Datei Bearbeiten Ansicht Gehe Sprungziele Lesezeichen Hilfe

`solve` – **Lösen von Gleichungen und Ungleichungen**

`solve(eq, x)` liefert die Menge aller komplexen Lösungen der Gleichung oder Ungleichung `eq` bezüglich `x`.

`solve(system, vars)` löst ein System von Gleichungen nach den Variablen `vars` auf.

`solve(eq, vars)` bewirkt dasselbe wie `solve([eq], vars)`.

`solve(system, x)` bewirkt dasselbe wie `solve(system, [x])`.

`solve(eq)` ohne zweites Argument bewirkt dasselbe wie `solve(eq, S)`, wo `S` die Menge aller Unbestimmten in `eq` ist. Dasselbe gilt für `solve(system)`.

Aufruf(e):

- `solve(eq, x <, options>)`
- `solve(eq, vars <, options>)`
- `solve(eq <, options>)`
- `solve(system, x <, options>)`
- `solve(system, vars <, options>)`
- `solve(system <, options>)`
- `solve(ODE)`
- `solve(REC)`

Parameter:

- `eq` — eine einzelne Gleichung oder eine Ungleichung vom Typ `"_equal"`, `"_less"`, `"_leequal"`, oder `"_unequal"`. Auch ein arithmetischer Ausdruck wird akzeptiert und als Gleichung mit verschwindender rechter Seite interpretiert.
- `x` — die Unbestimmte, nach der aufgelöst werden soll: ein Bezeichner oder ein indizierter Bezeichner
- `vars` — eine nichtleere Menge oder Liste von Unbestimmten, nach denen aufgelöst werden soll

Drücken Sie F1 um die Hilfe aufzurufen | Seite 3

Abbildung 2.1. Das Hilfefenster unter Windows

2.2 Das Rechnen mit Zahlen

Man kann MuPAD wie einen Taschenrechner zum Rechnen mit Zahlen benutzen. Die folgende Eingabe liefert als Ergebnis eine rationale Zahl:

```
>> 1 + 5/2
```

$$\frac{7}{2}$$

Man sieht, dass MuPAD bei Rechnungen mit ganzen und rationalen Zahlen exakte Ergebnisse (im Gegensatz zu gerundeten Gleitpunktzahlen) liefert:

```
>> (1 + (5/2*3))/(1/7 + 7/9)^2
```

$$\frac{67473}{6728}$$

Das Symbol ^ steht dabei für das Potenzieren. MuPAD kann auch sehr große Zahlen effizient berechnen. Die Größe einer zu berechnenden Zahl ist lediglich durch den zur Verfügung stehenden Hauptspeicher Ihres Computers beschränkt. So ist z. B. die 123-te Potenz von 1234 diese ziemlich große Zahl:[1]

```
>> 1234^123
   170515806212727042875059727620626282654302313111068 29\
   04705296193221839138348680074713663067170605985726415\
   92314554345900570589670671499709086102539904846514793\
   13561730556366999395010462203568202735575775507008323\
   84441477783960263870670426857004040032870424806396806\
   96865587865016699383883388831980459159942845372414601\
   80942971772610762859524340680101441852976627983806720\
   3562799104
```

Neben der Grundarithmetik steht eine Reihe von MuPAD-Funktionen zum Rechnen mit Zahlen zur Verfügung. Ein einfaches Beispiel ist die Fakultät $n! = 1 \cdot 2 \cdots n$ einer natürlichen Zahl n, die in mathematischer Notation angefordert werden kann:

```
>> 100!
   93326215443944152681699238856266700490715968264381621\
   46859296389521759999322991560894146397615651828625369\
   7920827223758251185210916864000000000000000000000000
```

Mit `isprime` kann überprüft werden, ob eine natürliche Zahl eine Primzahl ist. Diese Funktion liefert entweder `TRUE` (wahr) oder `FALSE` (falsch):

```
>> isprime(123456789)
   FALSE
```

Mit `ifactor` (englisch: *integer factorization*) ergibt sich die Primfaktorzerlegung:

```
>> ifactor(123456789)
```

$$3^2 \cdot 3607 \cdot 3803$$

[1] In diesem Ergebnis bedeutet das Zeichen „Backslash" \ am Ende einer Zeile, dass das Ergebnis in der nächsten Zeile fortgesetzt wird.
Lange Ausgaben, die umbrochen werden, werden in MuPAD in einem anderen Format dargestellt als kurze Ausgaben. Details stehen in Kapitel 13.

2.2.1 Exakte Berechnungen

Betrachten wir nun den Fall, dass die Zahl $\sqrt{56}$ „berechnet“ werden soll. Hierbei ergibt sich das Problem, dass dieser Wert als irrationale Zahl nicht einfach in der Form Zähler/Nenner mit Hilfe ganzer Zahlen exakt darstellbar ist. Eine „Berechnung“ kann daher nur darauf hinauslaufen, eine *möglichst einfache* exakte Darstellung zu finden. Bei der Eingabe von $\sqrt{56}$ mittels `sqrt` (englisch: *square root* = Quadratwurzel) liefert MuPAD:

```
>> sqrt(56)
```

$$2\sqrt{14}$$

Das Ergebnis ist die Vereinfachung von $\sqrt{56}$ zu dem exakten Wert $2 \cdot \sqrt{14}$, wobei $\sqrt{14}$ oder auch $14^{1/2}$ in MuPAD die positive Lösung der Gleichung $x^2 = 14$ bedeutet. In der Tat ist dies wohl die einfachste exakte Darstellung des Ergebnisses. Man beachte, dass $\sqrt{14}$ von MuPAD als ein Objekt angesehen wird, welches bestimmte Eigenschaften hat (im Wesentlichen, dass sich das Quadrat zu 14 vereinfachen lässt). Diese werden automatisch benutzt, wenn mit solchen Symbolen gerechnet wird, z. B.:

```
>> sqrt(14)^4
```

$$196$$

Als weiteres Beispiel einer exakten Rechnung möge die Bestimmung des Grenzwertes

$$e = \lim_{n\to\infty} \left(1 + \frac{1}{n}\right)^n$$

dienen. Die Funktion `limit` berechnet Grenzwerte, der Bezeichner `infinity` steht in MuPAD für „Unendlich“:

```
>> limit((1 + 1/n)^n, n = infinity)
```

$$\mathrm{e}$$

Um diese Zahl einzugeben, müssen Sie entweder den Buchstaben `E` oder die Eingabe `exp(1)` verwenden; `exp` ist die MuPAD-Schreibweise für die Exponentialfunktion. MuPAD beherrscht auch (exakte) Rechenregeln für dieses Objekt. So liefert z. B. der natürliche Logarithmus `ln`:

```
>> ln(1/exp(1))
```

$$-1$$

Weitere exakte Berechnungen werden uns im Laufe dieser Einführung begegnen.

2.2.2 Numerische Näherungen

Neben exakten Berechnungen ermöglicht MuPAD auch das Rechnen mit numerischen Näherungen. Wenn Sie z. B. $\sqrt{56}$ in Dezimalschreibweise annähern möchten, so müssen Sie die Funktion `float` (englisch: *floating point number* = Gleitpunktzahl) benutzen. Diese Funktion berechnet den Wert ihres Argumentes in so genannter *Gleitpunktdarstellung*:

```
>> float(sqrt(56))
    7.483314774
```

Die Genauigkeit der Näherung hängt vom Wert der globalen Variablen `DIGITS` ab, der Anzahl der Dezimalziffern für numerische Rechnungen. Ihr voreingestellter Standardwert ist 10:

```
>> DIGITS; float(67473/6728)
    10

    10.02868609
```

Globale Variablen wie `DIGITS` beeinflussen das Verhalten von MuPAD, sie werden auch *Umgebungsvariablen* genannt.[2] Im entsprechenden Abschnitt „Umgebungsvariablen" der MuPAD-Kurzreferenz [O 04] findet man eine vollständige Auflistung der in MuPAD implementierten Umgebungsvariablen.

Die Variable `DIGITS` kann jeden beliebigen ganzzahligen Wert zwischen 1 und $2^{31} - 1$ annehmen:

```
>> DIGITS := 100: float(67473/6728); DIGITS := 10:
   10.02868608799048751486325802615933412604042806183115\
   338882282996432818073721759809750297265160523187
```

Vor den nächsten Berechnungen haben wir den Wert von `DIGITS` auf 10 zurückgesetzt. Dieses kann auch durch den Befehl `delete DIGITS` erreicht werden.

---

[2] Es ist besondere Vorsicht angezeigt, wenn die selbe Rechnung mit verschiedenen Werten von `DIGITS` durchgeführt wird. Einige der komplexeren numerischen Algorithmen in MuPAD sind mit der Option *"remember"* implementiert, wodurch sich diese Algorithmen an frühere Ergebnisse erinnern (Abschnitt 18.9). Dies kann zu ungenauen numerischen Ergebnissen führen, wenn aus früheren Rechnungen Werte erinnert werden, die mit geringerer Genauigkeit berechnet wurden. Im Zweifelsfall sollte vor dem Heraufsetzen von `DIGITS` die MuPAD-Sitzung mit `reset()` neu initialisiert werden (Abschnitt 14.3), wodurch das MuPAD-Gedächtnis gelöscht wird.

Bei arithmetischen Operationen mit Zahlen rechnet MuPAD automatisch immer näherungsweise, sobald *mindestens eine* der beteiligten Zahlen in Gleitpunktdarstellung gegeben ist:

```
>> (1.0 + (5/2*3))/(1/7 + 7/9)^2
```

$$10.02868609$$

Bitte beachten Sie, dass die folgenden Aufrufe

```
>> 2/3*sin(2), 0.6666666666*sin(2)
```

beide *keine* näherungsweise Berechnung von sin(2) zur Folge haben, da `sin(2)` keine Zahl, sondern ein Ausdruck ist, der für den (exakten) Wert von sin(2) steht:

$$\frac{2\sin(2)}{3},\ 0.6666666666\sin(2)$$

Die Trennung der beiden Werte durch ein Komma erzeugt einen speziellen Datentyp, eine so genannte *Folge*. Dieser Typ wird in Abschnitt 4.5 genauer beschrieben. Um im obigen Fall eine Gleitpunktdarstellung zu erreichen, muss wieder die Funktion `float` benutzt werden:[3]

```
>> float(2/3*sin(2)), 0.6666666666*float(sin(2))
```

$$0.6061982846,\ 0.6061982845$$

Die meisten MuPAD-Funktionen wie etwa `sqrt`, die trigonometrischen Funktionen, die Exponentialfunktion oder der Logarithmus liefern automatisch numerische Ergebnisse, wenn ihr Argument eine Gleitpunktzahl ist:

```
>> sqrt(56.0), sin(3.14)
```

$$7.483314774,\ 0.001592652916$$

Mit den Konstanten π und e, dargestellt durch `PI` und `E = exp(1)`, kann man in MuPAD *exakt* rechnen:

```
>> cos(PI), ln(E)
```

$$-1,\ 1$$

Falls gewünscht, kann man wiederum mit Hilfe der Funktion `float` eine numerische Approximation dieser Konstanten erhalten:

[3] Beachten Sie die letzte Ziffer. Der zweite Befehl liefert ein etwas ungenaueres Ergebnis, da `0.666...` bereits eine Näherung von 2/3 ist und sich dieser Fehler auf das Endergebnis auswirkt.

```
>> DIGITS := 100:
   float(PI); float(E);
   delete DIGITS:
   3.141592653589793238462643383279502884197169399375105\
   8209749445923078164062862089986280348253421170068

   2.718281828459045235360287471352662497757247093699959\
   5749669676277240766303535475945713821785251664274
```

Aufgabe 2.2: Berechnen Sie $\sqrt{27} - 2\,\sqrt{3}$ und $\cos(\pi/8)$ exakt. Ermitteln Sie auf 5 Dezimalstellen genaue numerische Näherungen!

2.2.3 Komplexe Zahlen

Die imaginäre Einheit $\sqrt{-1}$ wird in MuPAD-Eingaben durch das Symbol `I` dargestellt, als Ausgabe im Formelsatz (vgl. Kapitel 13) wird ein aufrechtes i verwendet:

```
>> sqrt(-1), I^2
```

$$\mathrm{i},\ -1$$

Komplexe Zahlen können in MuPAD in der üblichen mathematischen Notation $x + y\,\mathrm{i}$ eingegeben werden, wobei der Real- bzw. Imaginärteil x bzw. y jeweils ganze Zahlen, rationale Zahlen oder auch Gleitpunktzahlen sein können:

```
>> (1 + 2*I)*(4 + I), (1/2 + I)*(0.1 + I/2)^3
```

$$2 + 9\,\mathrm{i},\ 0.073 - 0.129\,\mathrm{i}$$

Das Ergebnis von Rechenoperationen wird nicht immer nach Real- und Imaginärteil zerlegt zurückgeliefert, wenn neben Zahlenwerten symbolische Ausdrücke wie z. B. `sqrt(2)` verwendet wurden:

```
>> 1/(sqrt(2) + I)
```

$$\frac{1}{\sqrt{2} + \mathrm{i}}$$

Die Funktion `rectform` (englisch: *rectangular form*) erzwingt jedoch die Zerlegung nach Real- und Imaginärteil:

```
>> rectform(1/(sqrt(2) + I))
```

$$\frac{\sqrt{2}}{3} - \frac{\mathrm{i}}{3}$$

Mittels der Funktionen `Re` bzw. `Im` erhält man jeweils den Realteil x bzw. den Imaginärteil y einer komplexen Zahl $x + y\,\mathrm{i}$, der konjugiert komplexe Wert $x - y\,\mathrm{i}$ wird durch `conjugate` berechnet. Der Absolutbetrag $|x + y\,\mathrm{i}| = \sqrt{x^2 + y^2}$ wird von der MuPAD-Funktion `abs` geliefert:

```
>> Re(1/(sqrt(2) + I)), Im(1/(sqrt(2) + I)),
   abs(1/(sqrt(2) + I)), conjugate(1/(sqrt(2) + I)),
   rectform(conjugate(1/(sqrt(2) + I)))
```

$$\frac{\sqrt{2}}{3},\ \frac{-1}{3},\ \frac{\sqrt{3}}{3},\ \frac{1}{\sqrt{2}-\mathrm{i}},\ \frac{\sqrt{2}}{3}+\frac{\mathrm{i}}{3}$$

2.3 Symbolisches Rechnen

Dieser Abschnitt enthält einige Beispiele von MuPAD-Sitzungen, mit denen eine kleine Auswahl der symbolischen Möglichkeiten des Systems demonstriert werden soll. Die mathematischen Fähigkeiten stecken im Wesentlichen in den von MuPAD zur Verfügung gestellten Funktionen zum Differenzieren, zum Integrieren, zur Vereinfachung von Ausdrücken usw., wobei einige dieser Funktionen in den folgenden Beispielen vorgestellt werden. Diese Demonstration ist wenig systematisch: Es werden beim Aufrufen der Systemfunktionen Objekte unterschiedlichster Datentypen wie Folgen, Mengen, Listen, Ausdrücke etc. benutzt, die in Kapitel 4 dann jeweils einzeln vorgestellt und genauer diskutiert werden.

2.3.1 Einfache Beispiele

Ein symbolischer Ausdruck in MuPAD darf unbestimmte Größen (Bezeichner) enthalten, mit denen gerechnet werden kann. Der folgende Ausdruck enthält die beiden Unbestimmten x und y:

```
>> f := y^2 + 4*x + 6*x^2 + 4*x^3 + x^4
```

$$4\,x + 6\,x^2 + 4\,x^3 + x^4 + y^2$$

Der Ausdruck wurde hier durch den Zuweisungsoperator `:=` einem Bezeichner `f` zugewiesen, der nun als Abkürzung für den Ausdruck verwendet werden kann. Man sagt, der Bezeichner `f` hat nun als *Wert* den zugewiesenen Ausdruck. Beachten Sie bitte, dass MuPAD die eingegebene Reihenfolge der Summanden vertauscht hat.[4]

[4] Summanden werden intern nach gewissen Kriterien sortiert, wodurch das System beim Rechnen schneller auf diese Bausteine der Summe zugreifen kann. Solch

Zum Differenzieren von Ausdrücken stellt MuPAD die Systemfunktion `diff` zur Verfügung:

```
>> diff(f, x), diff(f, y)
```

$$12\,x + 12\,x^2 + 4\,x^3 + 4,\ 2\,y$$

Es wurde hierbei einmal nach x und einmal nach y abgeleitet. Mehrfache Ableitungen können durch mehrfache `diff`-Aufrufe oder auch durch einen einfachen Aufruf berechnet werden:

```
>> diff(diff(diff(f, x), x), x), diff(f, x, x, x)
```

$$24\,x + 24,\ 24\,x + 24$$

Alternativ kann zum Ableiten der Differentialoperator `'` benutzt werden, der einer Funktion ihre Ableitungsfunktion zuordnet:[5]

```
>> sin', sin'(x)
```

$$\cos,\ \cos(x)$$

Der Ableitungsstrich `'` ist nur eine verkürzte Eingabeform des Differentialoperators `D`, der mit dem Aufruf `D(Funktion)` die Ableitungsfunktion liefert:

```
>> D(sin), D(sin)(x)
```

$$\cos,\ \cos(x)$$

Integrale können durch `int` berechnet werden. Der folgende Aufruf, in dem ein Integrationsintervall angegeben wird, berechnet ein bestimmtes Integral:

---

eine Umsortierung der Eingabe geschieht natürlich nur bei mathematisch vertauschbaren Operationen wie z. B. der Addition oder der Multiplikation, wo die vertauschte Reihenfolge ein mathematisch äquivalentes Objekt ergibt.

[5] MuPAD verwendet beim Differentialoperator eine mathematisch saubere Notation: `'` bzw. `D` differenzieren Funktionen, während `diff` Ausdrücke ableitet. Im Beispiel verwandelt `'` den Namen der abzuleitenden Funktion in den Namen der Ableitungsfunktion. Oft wird eine nicht korrekte Notation wie z. B. $(x + x^2)'$ für die Ableitung der Funktion $F : x \mapsto x + x^2$ verwendet, wobei nicht streng zwischen der Abbildung F und dem Bildpunkt $f = F(x)$ an einem Punkt x unterschieden wird. MuPAD unterscheidet streng zwischen der *Funktion* F und dem *Ausdruck* $f = F(x)$, die durch unterschiedliche Datentypen realisiert werden. Die f zugeordnete Abbildung kann in MuPAD durch

```
>> F := x -> x + x^2:
```

definiert werden. Die Ableitung als Ausdruck kann somit auf zwei Arten erhalten werden:

```
>> diff(f, x) = F'(x);

                        2 x + 1 = 2 x + 1
```

Der MuPAD-Aufruf `f'` nach `f := x + x^2` ist in diesem Zusammenhang unsinnig.

```
>> int(f, x = 0..1)
```

$$y^2 + \frac{26}{5}$$

Der folgende Aufruf ermittelt eine Stammfunktion, einen Ausdruck in x mit einem symbolischen Parameter y. `int` liefert keinen allgemeinen Ausdruck für alle Stammfunktionen (mit additiver Konstante), sondern eine spezielle:

```
>> int(f, x)
```

$$x\,y^2 + 2\,x^2 + 2\,x^3 + x^4 + \frac{x^5}{5}$$

Versucht man, einen Ausdruck zu integrieren, dessen Stammfunktion nicht mit Hilfe elementarer Funktionen darstellbar ist, so liefert `int` sich selbst als symbolischen Ausdruck zurück:

```
>> Stammfunktion := int(1/(exp(x^2) + 1), x)
```

$$\int \frac{1}{\mathrm{e}^{x^2} + 1}\,\mathrm{d}x$$

Dieses Objekt hat aber durchaus mathematische Eigenschaften. Der Differenzierer erkennt, dass die Ableitung durch den Integranden gegeben ist:

```
>> diff(Stammfunktion, x)
```

$$\frac{1}{\mathrm{e}^{x^2} + 1}$$

Ein bestimmtes Integral, welches als symbolischer Ausdruck berechnet wird, stellt mathematisch einen Zahlenwert dar:

```
>> int(1/(exp(x^2) + 1), x = 0..1)
```

$$\int_0^1 \frac{1}{\mathrm{e}^{x^2} + 1}\,\mathrm{d}x$$

Dies ist in MuPAD eine exakte Darstellung dieser Zahl, welche nicht weiter vereinfacht werden konnte. Eine numerische Gleitpunktnäherung kann durch `float` berechnet werden:

```
>> float(%)
```

0.41946648

Das Symbol `%` (äquivalent zum Aufruf `last(1)`) steht dabei in MuPAD für den letzten berechneten Ausdruck (Kapitel 12).

MuPAD kennt die wichtigsten mathematischen Funktion wie die Wurzelfunktion `sqrt`, die Exponentialfunktion `exp`, die trigonometrischen Funktionen `sin`, `cos`, `tan`, die Hyperbelfunktionen `sinh`, `cosh`, `tanh`, die entsprechenden inversen Funktionen `ln`, `arcsin`, `arccos`, `arctan`, `arcsinh`, `arccosh`, `arctanh` sowie eine Reihe weiterer spezieller Funktionen wie z. B. die Gamma-Funktion, die `erf`-Funktion, Bessel-Funktionen etc. (die MuPAD-Kurzreferenz [O 04] gibt im Abschnitt „Spezielle mathematische Funktionen" einen Überblick). Dies heißt, dass MuPAD die entsprechenden Rechenregeln (z. B. die Additionstheoreme der trigonometrischen Funktionen) kennt und benutzt, numerische Gleitpunktnäherungen wie z. B. `float(exp(1))=2.718...` berechnen kann und spezielle Werte kennt (z. B. `sin(PI)=0`). Beim Aufruf dieser Funktionen liefern sich diese meist symbolisch zurück, da dies die einfachste exakte Darstellung des Wertes ist:

```
>> sqrt(2), exp(1), sin(x + y)
```

$$\sqrt{2},\ \mathrm{e},\ \sin(x+y)$$

Die Aufgabe des Systems ist es im Wesentlichen, solche Ausdrücke unter Ausnutzung der Rechenregeln zu vereinfachen oder umzuformen. So erzwingt z. B. die Systemfunktion `expand`, dass Funktionen wie `exp`, `sin` etc. mittels der entsprechenden Additionstheoreme „expandiert" werden, wenn ihr Argument eine symbolische Summe ist:

```
>> expand(exp(x + y)), expand(sin(x + y)),
   expand(tan(x + 3*PI/2))
```

$$\mathrm{e}^x\,\mathrm{e}^y,\ \cos(x)\,\sin(y)+\cos(y)\,\sin(x),\ -\frac{1}{\tan(x)}$$

Allgemein gesprochen ist eine der Hauptaufgaben eines Computeralgebrasystems, Ausdrücke zu manipulieren und zu vereinfachen. MuPAD stellt zur Manipulation neben `expand` die Funktionen `collect`, `combine`, `normal`, `partfrac`, `radsimp`, `rewrite` und `simplify` zur Verfügung, die in Kapitel 9 genauer vorgestellt werden. Einige dieser Hilfsmittel sollen hier schon erwähnt werden:

Mit `normal` werden rationale Ausdrücke zusammengefasst, d. h. auf einen gemeinsamen Nenner gebracht:

```
>> f := x/(1 + x) - 2/(1 - x): g := normal(f)
```

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

Gemeinsame Faktoren in Zähler und Nenner werden durch `normal` gekürzt:

```
>> normal(x^2/(x + y) - y^2/(x + y))
```

$$x - y$$

Umgekehrt wird ein rationaler Ausdruck durch `partfrac` (englisch: *partial fraction* = Partialbruch) in eine Summe rationaler Terme mit einfachen Nennern zerlegt:

```
>> partfrac(g, x)
```

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

Die Funktion `simplify` (englisch: *to simplify* = vereinfachen) ist ein universeller Vereinfacher, mit dem MuPAD eine möglichst einfache Darstellung eines Ausdrucks zu erreichen versucht:

```
>> simplify((exp(x) - 1)/(exp(x/2) + 1))
```

$$\mathrm{e}^{\frac{x}{2}} - 1$$

Die Vereinfachung kann durch Übergabe zusätzlicher Argumente an `simplify` vom Nutzer gesteuert werden (siehe `?simplify`).

Die Funktion `radsimp` vereinfacht Zahlenausdrücke mit Radikalen (Wurzeln):

```
>> f := sqrt(4 + 2*sqrt(3)): f = radsimp(f)
```

$$\sqrt{\sqrt{3} + 2}\,\sqrt{2} = \sqrt{3} + 1$$

Hierbei wurde eine Gleichung erzeugt, die ein zulässiges Objekt ist.

Eine weitere wichtige Funktion ist der Faktorisierer `factor`, der einen Ausdruck in ein Produkt einfacherer Ausdrücke zerlegt:

```
>> factor(x^3 + 3*x^2 + 3*x + 1),
   factor(2*x*y - 2*x - 2*y + x^2 + y^2),
   factor(x^2/(x + y) - z^2/(x + y))
```

$$(x+1)^3,\ (x+y-2)\cdot(x+y),\ \frac{(x-z)\cdot(x+z)}{(x+y)}$$

Die Funktion `limit` berechnet Grenzwerte. Beispielsweise hat die Funktion $\sin(x)/x$ für $x = 0$ eine stetig behebbare Definitionslücke, wobei der dort passende Funktionswert durch den Grenzwert für $x \to 0$ gegeben ist:

```
>> limit(sin(x)/x, x = 0)
```

1

Man kann auf mehrere Weisen eigene Funktionen innerhalb einer MuPAD-Sitzung definieren. Ein einfacher und intuitiver Weg benutzt den Abbildungsoperator `->` (das Minuszeichen gefolgt vom „größer“-Zeichen):

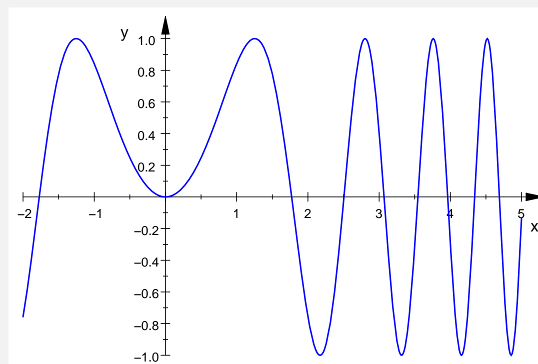
```
>> F := x -> x^2:  F(x), F(y), F(a + b), F'(x)
```

$x^2,\ y^2,\ (a+b)^2,\ 2\,x$

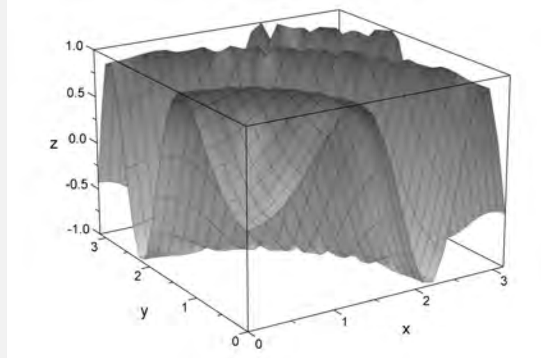
In Kapitel 18 wird auf die Programmiermöglichkeiten MuPADs eingegangen und die Implementierung beliebig komplexer Algorithmen durch MuPAD-Prozeduren beschrieben. Prozeduren bieten die Möglichkeit, komplizierte Funktionen in MuPAD selbst zu definieren.

Die MuPAD-Versionen, die innerhalb einer Fensterumgebung arbeiten, können die graphischen Fähigkeiten der Umgebung benutzen, um unmittelbar mathematische Objekte zu visualisieren. Die relevanten MuPAD-Funktionen zur Erzeugung von Graphiken sind `plotfunc2d` und `plotfunc3d` sowie die in der Graphik-Bibliothek `plot` installierten Routinen. Mit `plotfunc2d` bzw. `plotfunc3d` können Funktionen mit einem bzw. zwei Argumenten gezeichnet werden:

```
>> plotfunc2d(sin(x^2), x = -2..5)
```



```
>> plotfunc3d(sin(x^2 + y^2), x = 0..PI, y = 0..PI)
```



Je nach MuPAD-Version öffnet MuPADs Graphikmodul ein separates Fenster oder die Graphik erscheint im Notebook unterhalb des Aufrufs des Graphikbefehls. Die Graphiken lassen sich interaktiv manipulieren. Eine Beschreibung der graphischen Möglichkeiten MuPADs findet sich in Kapitel 11.

Eine wichtige Aufgabe für ein Computeralgebrasystem ist sicherlich das Lösen von Gleichungen bzw. Gleichungssystemen. MuPAD stellt hierfür die Funktion `solve` zur Verfügung:

```
>> Gleichungen := {x + y = a, x - a*y = b}:
>> Unbekannte := {x, y}:
>> Optionen := IgnoreSpecialCases:
>> solve(Gleichungen, Unbekannte, Optionen)
```

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

Hierbei werden eine Menge mit 2 Gleichungen und eine Menge mit den Unbekannten angegeben, nach denen aufgelöst werden soll. Das Ergebnis ist durch vereinfachte Gleichungen gegeben, aus denen die Lösung abgelesen werden kann. Im obigen Beispiel tauchen neben `x` und `y` die symbolischen Parameter `a` und `b` in den Gleichungen auf, weshalb `solve` durch Angabe der Unbekannten mitgeteilt wird, nach welchen Symbolen aufgelöst werden soll. Diese Lösung ist nur korrekt, wenn `a` nicht -1 ist. Die Option `IgnoreSpecialCases` („ignoriere Spezialfälle") sagt MuPAD, dass wir an diesem speziellen Fall nicht interessiert sind.

Ohne diese Option liefert MuPAD eine komplette Lösung mit Fallunterscheidung:

```
>> solve(Gleichungen, Unbekannte)
```

$$\begin{cases} \left\{\left[x = \frac{b+a^2}{a+1}, y = \frac{a-b}{a+1}\right]\right\} & \text{if} \quad a \neq -1 \\ \{[x = -y - 1]\} & \text{if} \quad a = -1 \wedge b = -1 \\ \emptyset & \text{if} \quad b \neq -1 \wedge a = -1 \end{cases}$$

Im folgenden Beispiel wird nur eine Gleichung in einer Unbekannten übergeben, wobei `solve` automatisch die Unbestimmte aus der Gleichung herausgreift und danach auflöst:

```
>> solve(x^2 - 2*x + 2 = 0)
```

$$\{[x = 1 - \mathrm{i}], [x = 1 + \mathrm{i}]\}$$

Das Ausgabeformat ändert sich, wenn zusätzlich die Unbestimmte `x` angegeben wird, nach der aufgelöst werden soll:

```
>> solve(x^2 - 2*x + 2 = 0, x)
```

$$\{1 - \mathrm{i}, 1 + \mathrm{i}\}$$

Das Ergebnis ist wieder eine Menge, welche die beiden (komplexen) Lösungen der quadratischen Gleichung enthält. Eine detailliertere Beschreibung von `solve` findet sich in Kapitel 8.

Die Funktionen `sum` und `product` können symbolische Summen und Produkte verarbeiten. Die wohlbekannte Summe der arithmetischen Reihe $1+2+\cdots+n$ ergibt sich beispielsweise durch

```
>> sum(i, i = 1..n)
```

$$\frac{n\ (n+1)}{2}$$

Das Produkt $1 \cdot 2 \cdots n$ läßt sich als Fakultät $n!$ schreiben:

```
>> product(i^3, i = 1..n)
```

$$n!^3$$

Für die Darstellung von Matrizen und Vektoren hält MuPAD mehrere Datentypen bereit. Es können Felder (Abschnitt 4.9) benutzt werden; es ist jedoch wesentlich intuitiver, statt dessen den Datentyp „Matrix“ zu benutzen. Zur Erzeugung dient die Systemfunktion `matrix`:

```
>> A := matrix([[1, 2], [a, 4]])
```

$$\begin{pmatrix} 1 & 2 \\ a & 4 \end{pmatrix}$$

Eine angenehme Eigenschaft von so konstruierten Objekten ist, dass die Grundarithmetik `+`, `*`, etc. automatisch der mathematischen Bedeutung entsprechend umdefiniert („überladen“) ist. Man kann Matrizen (geeigneter Dimension) beispielsweise mit `+` (komponentenweise) addieren oder mit `*` multiplizieren:

```
>> B := matrix([[y, 3], [z, 5]]):
>> A, B, A + B, A * B
```

$$\begin{pmatrix} 1 & 2 \\ a & 4 \end{pmatrix}, \begin{pmatrix} y & 3 \\ z & 5 \end{pmatrix}, \begin{pmatrix} y+1 & 5 \\ a+z & 9 \end{pmatrix}, \begin{pmatrix} y+2\,z & 13 \\ 4\,z+a\,y & 3\,a+20 \end{pmatrix}$$

Die Funktion `linalg::det` aus der `linalg`-Bibliothek für lineare Algebra (Abschnitt 4.15.4) berechnet die Determinante:

```
>> linalg::det(A)
```

$$4 - 2\,a$$

Die Potenz `A^(-1)` liefert die Inverse der Matrix `A`:

```
>> A^(-1)
```

$$\begin{pmatrix} -\frac{2}{a-2} & \frac{1}{a-2} \\ \frac{a}{2\,a-4} & -\frac{1}{2\,a-4} \end{pmatrix}$$

Spaltenvektoren der Dimension n können als $n \times 1$-Matrizen aufgefasst werden:

```
>> b := matrix([1, x])
```

$$\begin{pmatrix} 1 \\ x \end{pmatrix}$$

Die Lösung $A^{-1}\mathbf{b}$ des linearen Gleichungssystem $A\mathbf{x} = \mathbf{b}$ mit der obigen Koeffizientenmatrix A und der gerade definierten rechten Seite $\mathbf{b}$ lässt sich demnach bequem folgendermaßen ermitteln:

```
>> Loesungsvektor := A^(-1)*b
```

$$\begin{pmatrix} \frac{x}{a-2} - \frac{2}{a-2} \\ \frac{a}{2\,a-4} - \frac{x}{2\,a-4} \end{pmatrix}$$

Die Funktion `normal` kann mit Hilfe der Systemfunktion `map` auf die Komponenten des Vektors angewendet werden, wodurch sich die Darstellung vereinfacht:

```
>> map(%, normal)
```

$$\begin{pmatrix} \frac{x-2}{a-2} \\ \frac{a-x}{2\,a-4} \end{pmatrix}$$

Zur Probe wird die Matrix `A` mit diesem Lösungsvektor multipliziert, wodurch sich die rechte Seite `b` des Gleichungssystems ergeben sollte:

```
>> A*%
```

$$\begin{pmatrix} \frac{2\,(a-x)}{2\,a-4} + \frac{x-2}{a-2} \\ \frac{4\,(a-x)}{2\,a-4} + \frac{a\,(x-2)}{a-2} \end{pmatrix}$$

Das Ergebnis hat zunächst wenig Ähnlichkeit mit der ursprünglichen rechten Seite. Es muss noch vereinfacht werden, um es identifizieren zu können:

```
>> map(%, normal)
```

$$\begin{pmatrix} 1 \\ x \end{pmatrix}$$

Abschnitt 4.15 liefert weitere Informationen zum Umgang mit Matrizen und Vektoren.

Aufgabe 2.3: Multiplizieren Sie den Ausdruck $(x^2 + y)^5$ aus!

Aufgabe 2.4: Verifizieren Sie mit MuPAD: $\dfrac{x^2 - 1}{x + 1} = x - 1$!

Aufgabe 2.5: Zeichnen Sie den Graphen der Funktion $f(x) = 1/\sin(x)$ im Bereich $1 \le x \le 10$!

Aufgabe 2.6: Informieren Sie sich genauer über die Funktion `limit`! Überprüfen Sie mit MuPAD die folgenden Grenzwerte:

$$\lim_{x\to 0}\frac{\sin(x)}{x}=1,\quad \lim_{x\to 0}\frac{1-\cos(x)}{x}=0,\quad \lim_{x\to 0+}\ln(x)=-\infty,$$
$$\lim_{x\to 0}x^{\sin(x)}=1,\quad \lim_{x\to\infty}\left(1+\frac{1}{x}\right)^x=e,\quad \lim_{x\to\infty}\frac{\ln(x)}{e^x}=0,$$
$$\lim_{x\to 0}x^{\ln(x)}=\infty,\quad \lim_{x\to\infty}\left(1+\frac{\pi}{x}\right)^x=e^\pi,\quad \lim_{x\to 0-}\frac{2}{1+e^{-1/x}}=0\ !$$

Der Grenzwert $\lim_{x\to 0} e^{\cot(x)}$ existiert nicht. Wie reagiert MuPAD?

Aufgabe 2.7: Informieren Sie sich genauer über die Funktion `sum`! Der Aufruf `sum(f(k), k = a..b)` berechnet eine *geschlossene Form* einer endlichen oder unendlichen Summe. Überprüfen Sie mit MuPAD die folgende Identität:

$$\sum_{k=1}^{n}(k^2+k+1)=\frac{n\,(n^2+3\,n+5)}{3}\ !$$

Bestimmen Sie die Werte der folgenden Reihen:

$$\sum_{k=0}^{\infty}\frac{2\,k-3}{(k+1)\,(k+2)\,(k+3)}\,,\quad \sum_{k=2}^{\infty}\frac{k}{(k-1)^2\,(k+1)^2}\ !$$

Aufgabe 2.8: Berechnen Sie $2\cdot(A+B)$, $A\cdot B$ und $(A-B)^{-1}$ für

$$A=\begin{pmatrix}1&2&3\\4&5&6\\7&8&0\end{pmatrix},\quad B=\begin{pmatrix}1&1&0\\0&0&1\\0&1&0\end{pmatrix}!$$

2.3.2 Eine Kurvendiskussion

In der folgenden Beispielsitzung sollen einige der im letzten Abschnitt vorgestellten Systemfunktionen dazu benutzt werden, eine Kurvendiskussion für die rationale Funktion

$$f:\ x\ \mapsto\frac{(x-1)^2}{x-2}+a$$

mit einem beliebigen Parameter a durchzuführen. Zunächst sollen mit MuPAD einige charakteristische Größen dieser Funktion bestimmt werden.

```
>> f := x -> (x - 1)^2/(x - 2) + a:
>> Problemstellen := discont(f(x), x)
```

$$\{2\}$$

Die Funktion `discont` sucht dabei Unstetigkeitsstellen (englisch: *discontinuities*) der durch den Ausdruck $f(x)$ gegebenen Funktion bzgl. der Variablen `x`. Es wird eine Menge von Unstetigkeitsstellen zurückgeliefert, d. h., das obige f ist für jedes $x \neq 2$ definiert und dort stetig. Wie man der Formel ansieht, handelt es sich bei $x = 2$ um eine Polstelle. In der Tat findet MuPAD die Grenzwerte $\mp\infty$ (englisch: *infinity* = Unendlich), wenn man sich von links bzw. rechts dieser Stelle nähert:

```
>> limit(f(x), x = 2, Left), limit(f(x), x = 2, Right)
```

$$-\infty, \infty$$

Die Nullstellen von f erhält man durch Auflösen der Gleichung $f(x) = 0$:

```
>> Nullstellen := solve(f(x) = 0, x)
```

$$\left\{1 - \frac{\sqrt{a\ (a+4)}}{2} - \frac{a}{2}, \frac{\sqrt{a\ (a+4)}}{2} - \frac{a}{2} + 1\right\}$$

Abhängig von `a` können diese Nullstellen *echt komplex* sein, was bedeutet, dass f über der reellen Achse dann keine Nullstellen hat. Nun sollen Extremstellen von f ermittelt werden. Dazu wird die erste Ableitung `f'` gebildet und deren Nullstellen gesucht:

```
>> f'(x)
```

$$\frac{2\,x-2}{x-2} - \frac{(x-1)^2}{(x-2)^2}$$

```
>> Extremstellen := solve(f'(x) = 0, x)
```

$$\{1, 3\}$$

Dies sind Kandidaten für lokale Extrema. Es könnten jedoch auch Sattelpunkte an diesen Stellen vorliegen. Falls die zweite Ableitung f'' von f an diesen Stellen nicht verschwindet, handelt es sich wirklich um lokale Extrema. Dies wird nun überprüft:

```
>> f''(1), f''(3)
```

$$-2, 2$$

Aus den bisherigen Ergebnissen können folgende Eigenschaften von f abgelesen werden: Die Funktion besitzt für jeden Wert des Parameters a ein lokales Maximum an der Stelle $x = 1$, eine Polstelle bei $x = 2$ und ein lokales Minimum an der Stelle $x = 3$. Die zugehörigen Extremwerte an diesen Stellen sind von a abhängig:

```
>> Maxwert := f(1)
```

$$a$$

```
>> Minwert := f(3)
```

$$a + 4$$

Für $x \to \mp\infty$ strebt f gegen $\mp\infty$:

```
>> limit(f(x), x = -infinity),
   limit(f(x), x = infinity)
```

$$-\infty, \infty$$

Das Verhalten von f für große Werte von x kann genauer angegeben werden. Die Funktion stimmt dort näherungsweise mit der linearen Funktion $x \mapsto x + a$ überein:

```
>> series(f(x), x = infinity)
```

$$x + a + \frac{1}{x} + \frac{2}{x^2} + \frac{4}{x^3} + \frac{8}{x^4} + O\left(\frac{1}{x^5}\right)$$

Hierbei wurde der Reihenentwickler `series` (englisch: *series* = Reihe) eingesetzt, um eine so genannte asymptotische Entwicklung der Funktion zu berechnen (Abschnitt 4.13).

Die gefundenen Ergebnisse können leicht anschaulich überprüft werden, indem der Graph von f für verschiedene Werte von `a` gezeichnet wird:

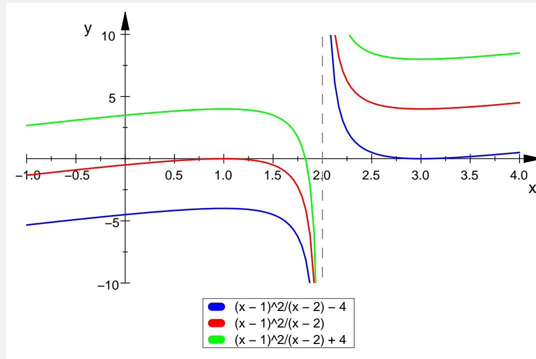
```
>> F := subs(f(x), a = -4):
   G := subs(f(x), a = 0):
   H := subs(f(x), a = 4):
   F, G, H
```

$$\frac{(x-1)^2}{x-2} - 4, \ \frac{(x-1)^2}{x-2}, \ \frac{(x-1)^2}{x-2} + 4$$

Mit der Funktion `subs` (Kapitel 6) werden Teilausdrücke ersetzt: Hier wurden für a die konkreten Werte -4, 0 bzw. 4 eingesetzt.

Die Funktionen F, G und H können nun gemeinsam in einer Graphik dargestellt werden:

```
>> plotfunc2d(F, G, H, x = -1..4)
```



2.3.3 Elementare Zahlentheorie

MuPAD bietet eine Anzahl von Funktionen der elementaren Zahlentheorie, z. B.:

- `isprime(n)` testet, ob $n \in \mathbb{N}$ eine Primzahl ist,
- `ithprime(n)` liefert die n-te Primzahl zurück,
- `nextprime(n)` liefert die kleinste Primzahl $\geq n$,
- `ifactor(n)` liefert die Primfaktorzerlegung von n.

Diese Routinen sind recht schnell, können aber mit sehr geringer Wahrscheinlichkeit falsche Ergebnisse liefern, da sie probabilistische Primzahltests verwenden.[6] Um eine Zahl garantiert fehlerfrei zu testen, kann statt `isprime` die (langsamere) Funktion `numlib::proveprime` verwendet werden.

Zunächst soll eine Liste aller Primzahlen bis 10 000 erzeugt werden. Dies kann auf viele Arten geschehen, z. B.:

```
>> Primzahlen := select([$ 1..10000], isprime)
       [2, 3, 5, 7, 11, 13, 17, ..., 9949, 9967, 9973]
```

[6] In der Praxis braucht man sich darüber keine Sorgen zu machen, denn das Risiko einer falschen Antwort ist vernachlässigbar: Die Wahrscheinlichkeit eines Hardwarefehlers ist viel größer als die Wahrscheinlichkeit, dass der randomisierte Test bei korrekt funktionierender Hardware ein falsches Ergebnis liefert.

Aus Platzgründen ist das Ergebnis hier nicht komplett abgedruckt. Zunächst wurde mittels des Folgengenerators `$` (Abschnitt 4.5) die Folge aller natürlichen Zahlen bis 10 000 erzeugt. Durch Klammerung mit `[ ]` entsteht hieraus eine MuPAD-Liste. Dann wurden mit `select` (Abschnitt 4.6) diejenigen Elemente herausgegriffen, für die die als zweites Argument übergebene Funktion `isprime` den Wert `TRUE` liefert. Die Anzahl dieser Primzahlen ist die Anzahl der Elemente in der Liste, diese wird durch `nops` (Abschnitt 4.1) berechnet:

```
>> nops(Primzahlen)
    1229
```

Alternativ kann die selbe Primzahlliste durch

```
>> Primzahlen := [ithprime(i) $ i = 1..1229]:
```

erzeugt werden. Hierbei wurde ausgenutzt, dass die Anzahl der gesuchten Primzahlen bereits bekannt ist. Man kann auch zunächst eine zu große Liste von Primzahlen erzeugen, von denen mittels `select` diejenigen herausgegriffen werden, die kleiner als 10 000 sind:

```
>> Primzahlen := select([ithprime(i) $ i=1..5000],
                        x -> (x<=10000)):
```

Hierbei ist das Objekt `x -> (x <= 10000)` eine Abbildung, die jedem `x` die Ungleichung `x <= 10000` zuordnet. Nur diejenigen Listenelemente, für die sich diese Ungleichung zu `TRUE` auswerten läßt, werden durch den `select`-Befehl herausgefiltert.

Die nächste Variante benutzt eine `repeat`-Schleife (Kapitel 16), in der mit Hilfe des Konkatenationsoperators `.` (Abschnitt 4.6) so lange Primzahlen i an die Liste angehängt werden, bis die nächstgrößere Primzahl, die durch `nextprime(i + 1)` berechnet wird, den Wert 10 000 überschreitet. Begonnen wird mit der leeren Liste und der ersten Primzahl $i = 2$:

```
>> Primzahlen := [ ]: i := 2:
>> repeat
     Primzahlen := Primzahlen . [i];
     i := nextprime(i + 1)
   until i > 10000 end_repeat:
```

Wir betrachten nun die Vermutung von Goldbach:

> *„Jede gerade Zahl größer als 2 kann als Summe zweier Primzahlen geschrieben werden.“*

Diese Vermutung soll für die geraden Zahlen bis 10000 überprüft werden. Dazu werden zunächst die ganzen Zahlen `[4, 6, ..., 10000]` erzeugt:

```
>> Liste := [2*i $ i = 2..5000]:
>> nops(Liste)
```

4999

Von diesen Zahlen werden die Elemente ausgewählt, die sich nicht in der Form „Primzahl + 2" schreiben lassen. Dazu wird überprüft, ob für eine Zahl i in der Liste $i - 2$ eine Primzahl ist:

```
>> Liste := select(Liste, i -> not isprime(i - 2)):
>> nops(Liste)
```

4998

Die einzige Zahl, die hierbei eliminiert wurde, ist 4 (denn für größere gerade Zahlen ist $i-2$ gerade und größer als 2, also keine Primzahl). Von den verbleibenden Zahlen werden nun diejenigen der Form „Primzahl + 3" eliminiert:

```
>> Liste := select(Liste, i -> not isprime(i - 3)):
>> nops(Liste)
```

3770

Es verbleiben 3770 ganze Zahlen, die weder von der Form „Primzahl + 2" noch von der Form „Primzahl + 3" sind. Der Test wird nun durch eine `while`-Schleife (Kapitel 16) fortgesetzt, wobei jeweils die Zahlen selektiert werden, die nicht von der Form „Primzahl + j" sind. Dabei durchläuft j die Primzahlen > 3. Die Anzahl der verbleibenden Zahlen werden in jedem Schritt mittels eines `print`-Befehls (Abschnitt 13.1.1) ausgegeben, die Schleife bricht ab, sobald die Liste leer ist:

```
>> j := 3:
>> while Liste <> [] do
       j := nextprime(j + 1):
       Liste := select(Liste,
                       i -> not isprime(i - j)):
       print(j, nops(Liste)):
   end_while:
                          5, 2747

                            ...

                           167, 1

                           173, 0
```

Die Goldbach-Vermutung ist damit für alle geraden Zahlen bis 10 000 richtig. Es wurde sogar gezeigt, dass sich all diese Zahlen als Summe zweier Primzahlen schreiben lassen, von denen eine kleiner gleich 173 ist.

Als weiteres Beispiel soll nun eine Liste der Abstände zwischen je zwei aufeinanderfolgenden Primzahlen bis 500 erstellt werden:

```
>> Primzahlen := select([$ 1..500], isprime):
>> Luecken := [Primzahlen[i] - Primzahlen[i - 1]
              $ i = 2..nops(Primzahlen)]
   [1, 2, 2, 4, 2, 4, 2, 4, 6, 2, 6, 4, 2, 4, 6, 6, 2,

      6, 4, 2, 6, 4, 6, 8, 4, 2, 4, 2, 4, 14, 4, 6, 2,

      10, 2, 6, 6, 4, 6, 6, 2, 10, 2, 4, 2, 12, 12, 4,

      2, 4, 6, 2, 10, 6, 6, 6, 2, 6, 4, 2, 10, 14, 4, 2,

      4, 14, 6, 10, 2, 4, 6, 8, 6, 6, 4, 6, 8, 4, 8, 10,

      2, 10, 2, 6, 4, 6, 8, 4, 2, 4, 12, 8, 4, 8]
```

Mit dem indizierten Aufruf `Primzahlen[i]` wird hierbei auf das i-te Element der Liste zugegriffen.

Eine alternative Möglichkeit bietet die Funktion `zip` (Abschnitt 4.6). Der Aufruf `zip(a, b, f)` verknüpft die Listen $a = [a_1, a_2, \ldots]$ und $b = [b_1, b_2, \ldots]$ elementweise mit der Funktion f: Die resultierende Liste ist $[f(a_1, b_1), f(a_2, b_2), \ldots]$. Das Ergebnis hat so viele Elemente wie die kürzere der beiden Listen. Für die gegebene Primzahlliste $a = [a_1, \ldots, a_n]$ können die gewünschten Differenzen durch das Verknüpfen mit einer „verschobenen" Listenkopie $b = [a_2, \ldots, a_n]$ mit der Funktion $(x, y) \mapsto y - x$ erstellt werden. Zuerst wird die „verschobene" Liste erzeugt, indem von einer Kopie der Primzahlliste das erste Element gelöscht wird, wobei sich die Liste verkürzt:

```
>> b := Primzahlen: delete b[1]:
```

Der folgende Aufruf ergibt das selbe Ergebnis wie oben:

```
>> Luecken := zip(Primzahlen, b, (x, y) -> (y - x)):
```

Eine andere nützliche Funktion ist der schon in Abschnitt 2.2 vorgestellte Faktorisierer `ifactor` zur Zerlegung einer ganzen Zahl in ihre Primfaktoren: Der Aufruf `ifactor(n)` liefert ein Objekt vom selben Typ wie `factor`, nämlich `Factored`. Objekte vom Datentyp `Factored` werden in einer intuitiv lesbaren Form auf dem Bildschirm ausgegeben:

```
>> ifactor(-123456789)
```

$$-3^2 \cdot 3607 \cdot 3803$$

Intern werden die Primfaktoren und die Exponenten jedoch in Form einer Liste gespeichert, auf deren Elemente man mittels `op` oder indiziert zugreifen kann. Die Hilfeseiten zu `ifactor` und `Factored` geben nähere Informationen.

Die interne Liste hat das Format

$$[s, p_1, e_1, \ldots, p_k, e_k]$$

mit Primzahlen $p_1, \ldots, p_k$, deren Exponenten $e_1, \ldots, e_k$ und dem Vorzeichen $s = \pm 1$; es gilt $n = s \cdot p_1^{e_1} \cdot p_2^{e_2} \cdots p_k^{e_k}$:

```
>> op(%)
```

$$-1,\ 3,\ 2,\ 3607,\ 1,\ 3803,\ 1$$

Mit Hilfe der Funktion `ifactor` soll nun bestimmt werden, wie viele der ganzen Zahlen zwischen 2 und 10 000 genau 2 verschiedene Primfaktoren besitzen. Dazu wird ausgenutzt, dass die von `ifactor(n)` gelieferte Liste $2\,m{+}1$ Elemente hat, wobei m die Anzahl der unterschiedlichen Primfaktoren von n ist. Damit liefert die Funktion

```
>> m := (nops@ifactor - 1)/2:
```

die Anzahl der Primfaktoren. Das Zeichen `@` bildet hierbei die Hintereinanderschaltung (Abschnitt 4.12) der Funktionen `ifactor` und `nops`, bei einem Aufruf `m(k)` wird also `m(k) = (nops(ifactor(k)) - 1) / 2` berechnet. Es wird eine Liste der Werte $m(k)$ für die Zahlen $k = 2, \ldots, 10000$ gebildet:

```
>> Liste := [m(k) $ k = 2..10000]:
```

In der folgenden `for`-Schleife (Kapitel 16) wird die Anzahl der Zahlen ausgegeben, die genau $i = 1, 2, 3, \ldots$ unterschiedliche Primfaktoren besitzen:

```
>> for i from 1 to 6 do
        print(i, nops(select(Liste, x -> (x = i))))
   end_for:
                              1, 1280

                              2, 4097

                              3, 3695

                               4, 894

                               5, 33

                                6, 0
```

Damit existieren im durchsuchten Bereich 1280 Zahlen mit genau einem Primfaktor,[7] 4097 Zahlen mit genau 2 verschiedenen Primfaktoren usw. Es ist leicht einzusehen, dass keine ganze Zahl mit 6 verschiedenen Primfaktoren gefunden wurde: Die kleinste dieser Zahlen, $2 \cdot 3 \cdot 5 \cdot 7 \cdot 11 \cdot 13$, ist bereits größer als 10 000.

Zahlreiche Funktionen zur Zahlentheorie sind in der Bibliothek `numlib` enthalten, unter Anderem die Funktion `numlib::numprimedivisors`, die die selbe Funktion wie das oben angegebene `m` erfüllt. Zum Umgang mit MuPAD-Bibliotheken verweisen wir auf das Kapitel 3.

Aufgabe 2.9: Von besonderem Interesse waren schon immer Primzahlen der Form $2^n \pm 1$.

a) Primzahlen der Form $2^p - 1$ (mit einer Primzahl p) sind unter dem Namen *Mersenne-Primzahlen* bekannt. Gesucht sind die ersten Mersenne-Primzahlen im Bereich $1 < p \leq 1000$.

b) Für $n \in \mathbb{N}$ heißt $2^{(2^n)} + 1$ die *n-te Fermatsche Zahl*. Fermat vermutete, dass alle diese Zahlen Primzahlen sind. Widerlegen Sie diese Vermutung!

---

[7] Es war bereits festgestellt worden, dass es 1229 Primzahlen in diesem Bereich gibt. Wie erklärt sich die Differenz?