Eine (sehr) kurze Einleitung in MuPAD

v1.2/21.09.2007 dani

Inhaltsverzeichnis

1 MuPAD-Notebooks

Das Computeralgebra-Systems MuPAD arbeitet mit Notebooks. Dies sind Arbeitsblätter, in denen man mathematische Ausdrücke berechnen und Befehle eingegeben kann. Zusätzlich kann man aber auch begleitenden Text und andere Bemerkungen hinzufügen.

Der aktuelle Text, den du gerade liest, befindet sich in einer *Textregion*, erkennbar an der *normalen* Schrift. MuPAD-Ausdrücke und -Befehle müssen dagegen in einer *Eingaberegion* eingegeben werden, erkennbar an dem roten Punkt am Beginn der Zeile. Die MuPAD-Ausgaben werden in einer *Ausgaberegion* dargestellt, normalerweise in blau.

1.1 Eingaben

Es ist ganz wichtig, die Zeichen `,` und `;` sowie `:` richtig einzusetzen. Eingaben werden in MuPAD mit einem `RETURN` abgeschlossen. MuPAD wertet daraufhin die Eingabe aus, führt die Berechnung durch und zeigt das Ergebnis an.

Eingaben können auch mit einem Semikolon abgeschlossen werden, was den gleichen Effekt erzielt. Allerdings darf am Ende eines Kommandos auch ein Doppelpunkt stehen. In diesem Fall wird die Berechnung zwar ebenso durchgeführt, das Resultat aber nicht am Bildschirm angezeigt. Durch den Einsatz eines Doppelpunktes kann vermieden werden, dass ohnehin nur als Zwischenschritt durchgeführte Anweisungen seitenweise Ergebnisse produzieren, die in diesem Stadium nicht von Interesse sind.

Es besteht auch die Möglichkeit, sich die Ergebnisse mehrerer Kommandos in einer Zeile ausgeben zu lassen. Diese Kommandos müssen dann durch Kommata voneinander getrennt werden. Diese Vorgehensweise hat den Vorteil, das sie platzsparend ist und einen direkten Vergleich der Ergebnisse in einer Zeile erlaubt. Selbstverständlich können all diese Methoden beliebig miteinander vermischt werden.

In diesem Zusammenhang ist die umgekehrte Situation erwähnenswert, nämlich dass ein Kommando sich über mehrere Zeilen erstreckt. Mit `SHIFT/RETURN` gelangt man dann in die nächste Zeile, ohne die Berechnung zu starten.

1.2 MuPAD als Taschenrechner

Das Programm beherrscht natürlich alle Grundrechenarten inklusive Potenz und Fakultät.

- `1+2`
  3

Es können auch mehrere Terme eingegeben werden, die mit einem Komma voneinander getrennt sind. Beachte aber, das Multiplikationen immer explizit mit dem Multiplikationszeichen $*$ geschrieben werden müssen.

- `2*3, 12/5, 2^3, 6!, 4*(7-2), 1/2 + 1/3, PI`
  $6,\ \frac{12}{5},\ 8,\ 720,\ 20,\ \frac{5}{6},\ \pi$

MuPAD schreckt auch nicht vor abenteuerlichen mathematischen Ausdrücken zurück.

- `sqrt( 14+3*sqrt(3+2*sqrt(5-12*sqrt(3-2*sqrt(2)))))`

$$\sqrt{3\sqrt{2\sqrt{5-12\sqrt{3-2\sqrt{2}}}+3}+14}$$

In einigen Fällen versucht sich MuPAD allerdings vor der mühsamen Vereinfachung eines Termes zu drücken. Dann man es mit dem Befehl `simplify` auch zu solch niederen Tätigkeiten zwingen.

- `simplify( sqrt(14+3*sqrt(3+2*sqrt(5-12*sqrt(3-2*sqrt(2))))) )`

$$\sqrt{2}+3$$

Es geht doch ... Alle Ergebnisse werden exakt angegeben und keine Näherungslösungen wie beim Taschenrechner verwendet. Selbstverständlich ist dies auf besonderen Wunsch auch mit dem Befehl `float` möglich.

- `float( sqrt(2) )`

```
1.414213562
```

Die Genauigkeit für numerische Berechnungen ist durch die Systemvariable `DIGITS` festgelegt und beträgt normalerweise 10 Stellen. Durch eine Veränderung dieser Variablen kann eine andere Ausgabe erzwungen werden.

- `DIGITS:=20 : float(sqrt(2)); DIGITS:=10 :`

```
1.4142135623730950488
```

Allerdings gilt die neue Genauigkeit jetzt auch für alle folgenden numerischen Berechungen und sollte daher meistens schnell wieder auf den Standardwert 10 zurückgesetzt werden.

`DIGITS` hat aber keinen Einfluss auf die Genauigkeit symbolischer Berechnungen. Die Fakultät von 100 hat 150 Stellen, weil MuPAD das Wort symbolisch als *unendlich genau* interpretiert. Trotz der Größe der Zahl sieht MuPAD keinen Anlass, auf numerische Verfahren umzusteigen.

- `100!`

```
933262154439441526816992388562667004907159682643816214\
685929638952175999932299156089414639761565182862536979\
208272237582511852109168640000000000000000000000000
```

Kürzer, aber eben auch viel ungenauer wie beim Taschenrechner, geht es natürlich auch.

- `float(100!)`

$$9.332621544 \cdot 10^{157}$$

1.3 Variable

Um sich Tipparbeit zu ersparen, kann man mehrfach benutzte Ausdrücke mit einem Aliasnamen, einer sogenannten Variablen, belegen. Dabei muss man den Zuweisungsoperator `:=` benutzen. Mit den Definitionen

- `z:=sqrt(2) : a:=sqrt(8) :`

lässt sich dann genauso wie mit den Originalausdrücken rechnen.

- `z*a`
  `4`

Aber Vorsicht: ab sofort werden diese Variable in allen weiteren Ausdrücken wieder durch das Original ersetzt. Wer also mit `(a+b)^2` jetzt die bekannte binomische Formel erwartet, wird enttäuscht, da für a der soeben festgelegte Wert eingesetzt wird.

- `(a+b)^2`
  $\left(b+2\sqrt{2}\right)^2$

Der Gebrauch von Variablen bietet also große Vorteile, muss allerdings auch genau überlegt sein muss. Häufig in der Mathematik benutzte Variablen wie a, b, c, x, y, f und g verbieten sich also von selbst. Da als Variablennamen aber beliebig lange Zeichenkombinationen aus Buchstaben und Ziffern benutzt werden können, stehen genügend viele Namen zur Verfügung. Die einzige Bedingung an die zu vergebenden Namen besteht darin, das das allererste Zeichen auf alle Fälle ein Buchstabe sein muss.

Zudem gibt es noch einen Rettungsanker: wenn man nicht mehr genau weiß, ob ein bestimmter Variablenname schon vergeben wurde oder nicht, kann eine bereits erfolgte Definition mit dem Befehl `delete variablenname` wieder aufgehoben werden. Danach steht dieser Name wieder ohne Einschränkung zur Verfügung.

- `delete a : a`
  `a`

Variable können auch mehrfach deklariert werden. Dabei wird immer die vorherige Bedeutung überschrieben und geht damit verloren.

1.4 Konstante und Funktionen

Die nachfolgende Tabelle gibt eine Übersicht über einige gebräuchliche mathematische Konstanten und Funktionen.

Konstante

`PI`	die Zahl π

Elementare Rechenoperatoren

`+ - * /`	Grundrechenarten
`^`	Potenz
`!`	Fakultät

Mathematische Funktionen

`abs`	Betrag einer Zahl
`sqrt(x)`	Quadratwurzel einer Zahl
`log(b,x)`	Logarithmus von x zur Basis b
`sin(x)`	trigonometrische Funktionen
`cos(x)`	(x wird im Bogenmaß gemessen)
`tan(x)`	
`arcsin(x)`	Umkehrfunktionen der trigonometrischen Funktionen
`arccos(x)`	(x wird im Bogenmaß gemessen)
`arctan(x)`	

2 Mathematische Ausdrücke

Obwohl diese Überschrift reichlich abstrakt klingt, behandelt dieser Abschnitt eines der wichtigsten und für Computeralgebraprogramme auch schwierigsten Themen: dem Vereinfachen mathematischer Ausdrücke. Das Thema ist deswegen so wichtig, weil Programme wie MuPAD dahin tendieren, unglaublich lange Formeln zu produzieren. Diese Formeln sind zwar korrekt, aber zum Verständnis oder zum Weiterverarbeiten manchmal nicht geeignet.

In vielen Fällen ist auch nicht eindeutig klar, was *einfach* eigentlich heißen soll. Der Term $(x-2)\cdot(x+2)\cdot(x^2-4)$ ist in der ausmultiplizierten Form x^4-16 deutlich kürzer. MuPAD findet allerdings die faktorisierte Form schöner.

- `(x-2)*(x+2)*(x^2-4)`
  $\left(x^2-4\right)\ (x-2)\ (x+2)$

Bei $(x-1)^{10}+1$ stimmt MuPAD dann wieder mit uns überein.

- `(x-1)^10+1`
  $(x-1)^{10}+1$

2.1 simplify

Das Kommando `simplify` versucht mit verschiedenen Transformationsregeln einen mathematischen Ausdruck zu vereinfachen. Das Ergebnis ist in sehr vielen Fällen wirklich gut, kann aber selbst bei einfachen Beispielen manchmal nicht gerade überzeugen.

- `simplify( (x-1)^10+1 )`
  $x^{10}-10\,x^9+45\,x^8-120\,x^7+210\,x^6-252\,x^5+210\,x^4-120\,x^3+45\,x^2-10\,x+2$

Für den Benutzer eines Programmes wie MuPAD bedeutet dies, dass er in vielen Fällen ein berechnetes Ergebnis noch nach seinen Wünschen gestalten muss.

2.2 expand

MuPAD lässt Produkte normalerweise stehen, ohne sie auszumultiplizieren.

- `(x−1)∗(x−2)∗(x−3)`
  $(x-1)\cdot(x-2)\cdot(x-3)$

Der `expand`-Befehl führt die Multiplikationen dagegen aus.

- `expand( (x−1)∗(x−2)∗(x−3) )`
  $x^3-6\,x^2+11\,x-6$

2.3 factor

`factor` ist das Gegenstück zu `expand`, soweit es das Ausmultiplizieren von Produkten anbelangt. Dieser Befehl versucht eine Summe von Einzeltermen in eine Produktform zu verwandeln.

- `factor( x^3−6∗x^2+11∗x−6 )`
  $(x-3)\cdot(x-1)\cdot(x-2)$

Brüche werden zuerst in ihre Normalform umgewandelt und anschließend Zähler und Nenner faktorisiert.

- `factor( (x^2+x−6)/(x^2−1) )`
  $\frac{(x+3)\cdot(x-2)}{(x-1)\cdot(x+1)}$

2.4 normal

Dieser Befehl liefert die Normalform eines rationalen Ausdrucks, d.h. einen gekürzten Bruch mit ausmultiplizierten Zähler und Nenner.

- `normal( (x^2−1)/(x+1) )`
  $x-1$

- `normal( 1/(x+1) + 1/(y+1) )`
  $\frac{x+y+2}{x+y+x\,y+1}$

2.5 numer

Der `numer`-Befehl liefert den Zähler eines Bruches

- `numer(-3/4)`
  -3

- `numer( (x^2-1)/(x^3-x^2+x-1) )`
  $x^2 - 1$

2.6 denom

Der `denom`-Befehl liefert den Nenner eines Bruches

- `denom( -3/4 )`
  `4`

- `denom( (x^2-1)/(x^3-x^2+x-1) )`
  $x^3 - x^2 + x - 1$

3 Folgen, Mengen und Listen

MuPAD kennt verschiedene Formen, um mehrere Ausdrücke ordnend zusammenzufassen. Obwohl diese Formen auf den ersten Blick recht ähnlich aussehen, unterscheiden sie sich intern fundamental und müssen daher auch mit verschiedenen Methoden bearbeitet werden.

3.1 Folgen

Folgen bzw. Aufzählungen stellen den elementarsten Datentyp zur Verwaltung von mehreren Ausdrücken dar. Sie können durch eine einfache Aufzählung erzeugt werden.

- `2,3,5,7`
  2, 3, 5, 7

Es steht aber auch der $-Operator zur Verfügung, um ein Folge aus einer Formel zu erzeugen. Im folgenden Beispiel wird eine Folge von Quadratzahlen erzeugt, wobei die Variable i alle Zahlen von 1 bis 9 mit einer vorgegeben Schrittweite von 1 durchläuft.

- `i^2 $ i=1..9`
  1, 4, 9, 16, 25, 36, 49, 64, 81

Zusätzlich kann auch die Schrittweite verändert werden.

- `i^2 $ i=1..9 step 2`
  1, 9, 25, 49, 81

Gibt es kein Schema für die Basiswerte der Quadrate, können diese auch explizit in einer Liste aufgezählt werden.

- `i^2 $ i in [1,3,4,8,9]`
  1, 9, 16, 64, 81

3.2 Listen

Listen unterscheiden sich von Folgen rein optisch nur durch die umgebenden eckigen Klammern, sind aber deutlich leistungsfähiger. Prinzipiell werden sie wie Folgen erzeugt, nur das zusätzlich eckige Klammern angegeben werden.

- `[2,3,5,7], [i^2 $ i=1..9], [i^2 $ i=1..9 step 2]`
  [2, 3, 5, 7], [1, 4, 9, 16, 25, 36, 49, 64, 81], [1, 9, 25, 49, 81]

Mit ihnen kann man aber viel besser arbeiten. So lässt sich z.B. jedes einzelne Listenelement auslesen, in dem die Position in eckigen Klammern angegeben wird. Der folgende Befehl liefert z.B. das allererste Element der Liste.

- `lst:=[3,2,1,4] : lst[1]`
  3

Zum Sortieren einer Liste ist der Befehl `sort` vorgesehen.

- `sort( [3,2,1,4] )`
  [1, 2, 3, 4]

Besonders mächtig ist der `map`-Befehl, der eine Operation gleichzeig auf alle Elemente der Liste anwendet. Die Syntax lautet `map(liste,operation)`. Damit kann man z.B. gleichzeitig aus allen Zahlen einer Liste die Wurzel ziehen.

- `lst:=[2,3,5,7] : w:=map(lst,sqrt)`
  $\left[\sqrt{2}, \sqrt{3}, \sqrt{5}, \sqrt{7}\right]$

Das Ergebnis ist glücklicherweise wieder eine Liste, so dass alle Wurzeln leicht in Dezimalzahlen umgewandelt werden können.

- `map(w,float)`
  [1.414213562, 1.732050808, 2.236067977, 2.645751311]

Mit diesen Beispielen zeigt sich wieder einmal, wie nützlich Variablen sind.

3.3 Mengen

Mengen erkennt man immer an den umgebenden geschweiften Klammern. Auf den ersten Blick hat es den Anschein, es wären Mengen und Listen dasselbe. Das ist aber nicht der Fall, da bei Mengen mehrfach auftetende Elemente automatisch eliminiert werden, wie es z.B. bei Lösungen von Gleichungen erwünscht ist.

- `{2,3,5,7,2}`
  $\{2, 3, 5, 7\}$

Außerdem wird die Reihenfolge der Elemente von MuPAD scheinbar willkürlich festgelegt und ist daher nicht vorhersehbar.

Etwas kompliziert ist es, wenn man eine solche Menge in eine Liste umwandeln möchte, die für viele mathematische Zwecke besser geeignet sind. Dazu speichert man die Lösungsmenge einer Gleichung in einer Variablen

- `lsg := solve(x^3−x^2−5∗x+5=0,x)`
  $\{2, 3, 5, 7\}$

und wendet dann einen etwas komplizierteren `map`-Befehl an. Der dabei verwendete Befehl `op` wandelt zunächst die Menge in eine Folge, um die dann Listenklammern gesetzt werden.

- `lst:=[op(lsg)]`
  $\left[\sqrt{5}, -\sqrt{5}, 1\right]$

Mit dieser Liste können dann z.B. schnell alle Lösungen als Dezimalzahlen angegeben werden oder weitere Folgeberechnungen durchgeführt werden.

- `map(lst,float)`
  $[2.236067977, -2.236067977, 1.0]$

4 Gleichungen

4.1 Normale Gleichungen

Der Standardbefehl zum Lösen einer Gleichung heißt `solve`.

- `solve( x^2+3∗x+2=0,x )`
  $\{-2, -1\}$

Gleichungen bereiten MuPAD überhaupt keine Probleme. Selbst wenn weitere Variable auftreten, arbeitet MuPAD zuverlässig. Wer z.B. die pq-Formel vergessen hat, gibt ganz einfach die zugehörige Gleichung ein.

- `solve( x^2+p*x+q=0,x )`
  $\left\{-\frac{p}{2}-\frac{\sqrt{p^2-4\,q}}{2}, \frac{\sqrt{p^2-4\,q}}{2}-\frac{p}{2}\right\}$

Na gut, die Schreibweise ist vielleicht etwas gewöhnungsbedürftig. Es handelt sich aber eindeutig um die pq-Formel.

Wenn nur eine einzige Variable in der Gleichung auftritt, reicht auch die Kurzform.

- `solve( x^2+3*x+2=0 )`
  $\{[x=-1], [x=-2]\}$

Zum einfachen Lösen von Gleichungen reicht diese Kurzform völlig aus. Sollen die Ergebnisse allerdings noch weiterverarbeitet werden, ist die ausführliche Form unbedingt vorzuziehen, da nur die Zahlenwerte als Menge und nicht noch der Variablennamen ausgegeben wird.

Allerdings kann `solve` deutlich mehr, als uns Mathematiklehrern auf dem Gymnasium lieb ist. Die Gleichung $x^2+1=0$ sollte eigentlich keine Lösungen haben, MuPAD behauptet aber dagegen etwas ganz anderes.

- `solve( x^2+1=0,x )`
  $\{-\mathrm{i}, \mathrm{i}\}$

Iiiiiii ... komplexe Zahlen. Es gibt also wohl noch ein paar Zahlen mehr als die reellen Zahlen, die uns die Mathematiklehrer bis jetzt verschwiegen haben. Das ist aber auch ganz gut so und soll so bleiben. Ein paar Geheimnisse muss es ja schließlich noch geben...

Der folgende Zusatz unterdrückt diese auf der Schule unerwünschten Lösungen. Doch muss jetzt unbedingt die Gleichungsvariable angegeben werden, um eine vernünftige Ausgabe zu erhalten.

- `solve( x^2+1=0,x,Real )`
  $\emptyset$

Eine leere Lösungsmenge, es geht doch ... Allerdings hat auch der überaus mächtige `solve`-Befehl seine Tücken. Mit der folgenden Gleichung kann er nun rein gar nichts anfangen.

- `solve( x^5-3*x-1=0,x )`
  $\mathrm{RootOf}(\mathrm{X1}^5-3\,\mathrm{X1}-1, \mathrm{X1})$

Dabei ist schon jedem Schüler eines Grundkurses völlig klar, dass es mindestens eine Lösung geben muss. Doch die mathematischen Methoden zur exakten Bestimmung von Lösungen können z.B. bei höhergradigen Polynomen versagen. In solchen Fällen muss man sich dann mit Näherungslösungen zufrieden geben. Das Attribut `AllRealRoots` versorgt uns nur mit reellen Lösungen.

- `numeric::solve( x^5-3*x-1=0,x,AllRealRoots )`
  $\{-1.214648042, -0.3347341428, 1.388791982\}$

4.2 Gleichungssysteme

Der `solve`-Befehl eignet sich auch zum Lösen von Gleichungssystemen. Dabei werden die durch Kommata getrennten Gleichungen sowie die Variablen jeweils mit geschweiften Klammern umgeben.

- `solve( {2*x+y+z=1,2*x-2*y-z=-7,4*x+y+3*z=1},{x,y,z} )`
  $\{[x = -1, y = 2, z = 1]\}$

Auch hier gibt es wieder eine Kurzform ohne Angabe der Gleichungsvariablen. Hier kann sie schon eher eingesetzt werden, da das Ergebnis relativ selten direkt weiterverarbeitet wird.

- `solve( {2*x+3*y+z=3,4*x-4*y+2*z=1,3*x-5*y+z=-1} )`
  $\left\{\left[x = 0, y = \frac{1}{2}, z = \frac{3}{2}\right]\right\}$

Gibt es keine Lösungen, wird uns dies unmissverständlich mitgeteilt.

- `solve( {x+y-z=4,4*x-2*y-2*z=3,5*x-4*y-2*z=0} )`
  $\emptyset$

Ebenso kann man leicht erkennen, wenn unendlich viele Lösungen existieren.

- `solve( {x-2*y-z=-2,3*x-6*y-3*z=-6,2*x-4*y-2*z=-4} )`
  $\{[x = 2\,y + z - 2]\}$

5 Funktionen

5.1 Deklaration von Funktionen

Die moderne Deklaration von Funktion in MuPAD ist zwar etwas gewöhnungsbedürftig, bietet aber viele Vorteile. Es gibt durchaus andere (auch einfachere) Möglichkeiten, die aber nicht alle Fähigkeiten dieser erst seit der Version 3.1.1. existierenden Deklaration bieten. Die eigentliche Funktionsdeklaration hat die Form

Funktionsvariable	Zuordnungspfeil	Funktionsterm
`x`	`-->`	`x^2`

- `f := x --> x^2`
  $x \mapsto x^2$

Davor befindet sich nur ein zusätzlicher Ausdruck, der diese Deklaration in der Variablen f speichert. Diese komplizierte Deklaration soll ja schließlich nicht immer wieder erneut eingegeben werden müssen.

Nun können ganz einfach Funktionswerte berechnet werden.

- `f(x), f(−2), f(3), f(a)`
  $x^2,\ 4,\ 9,\ a^2$

Wenn man sich noch mit der Erzeugung von Folgen auskennt, kann man auch ganz leicht Wertetabellen erstellen.

- `f(x) $ x=−3..3`
  $9,\ 4,\ 1,\ 0,\ 1,\ 4,\ 9$

Natürlich kann auch die Schrittweite für die Wertetabelle geändert werden.

- `f(x) $ x=−3..3 step 0.5`
  $9,\ 6.25,\ 4.0,\ 2.25,\ 1.0,\ 0.25,\ 0.0,\ 0.25,\ 1.0,\ 2.25,\ 4.0,\ 6.25\ 9.0$

Leider verliert man bei der einfachen Aufzählung von Funktionswerten leicht den Überblick. Besser wäre es, wenn x- und y-Koordinaten zusammenstehen würden. Wäre es so genehm?

- `[x,f(x)] $ x=−3..3`
  $[-3, 9],\ [-2, 4],\ [-1, 1],\ [0, 0],\ [1, 1],\ [2, 4],\ [3, 9]$

5.2 Beispiel einer Kurvendiskussion

Schauen wir uns doch einmal eine etwas *komplizierte* Funktion an und führen eine stark verkürzte Kurvendiskussion durch.

- `f := x−−>x^3−9∗x`
  $x \mapsto x^3 - 9\,x$

Wir beginnen mit den Nullstellen.

- `solve( f(x)=0,x )`
  $\{-3, 0, 3\}$

Das Verhalten für betragsgroße x (Verhalten im Unendlichen) ist mit dem `limit`-Befehl kein Problem.

- `limit(f(x),x=−infinity), limit(f(x),x=infinity)`
  $-\infty,\ \infty$

Auch die Ableitungen bereiten überhaupt keine Probleme. Dazu verwendet man einfach die übliche Schreibweise.

- `f'(x); f''(x); f'''(x)`
  $3\,x^2 - 9$
  $6\,x$
  6

Zur Berechnung der Extrempunkte muss man nach dem notwendigen Kriterium zunächst die erste Ableitung gleich 0 setzen.

- `e := solve(f'(x)=0,x)`
  $\left\{-\sqrt{3}, \sqrt{3}\right\}$

Da die Ergebnisse in Form einer Menge vorliegen, wandeln wir sie schnell in eine sortierte Liste um, damit nicht irgendwelche doppelt auftretenden Ergebnisse verloren gehen. Dabei hilft eine durchdachte Wahl von Variablennamen bei all den vielen Untersuchungen, die bei einer vollständigen Kurvendiskussion auftreten. Also x bzw. y für x- bzw. y-Koordinaten und den Zusatz n, e bzw. w für Nullstellen, Extrempunkte und Wendepunkte. Dann können auch gleiche Zahlen, die wie hier in unterschiedlichen Zusammenhängen auftreten, kein Chaos erzeugen.

- `xe := sort( [op(e)] )`
  $\left[-\sqrt{3}, \sqrt{3}\right]$

Ein hinreichendes Kriterium besagt nun, das wir diese Kandidaten in die zweite Ableitung einsetzen können und mit etwas Glück schon die Extrempunkte bestimmt haben.

- `map(xe,f'')`
  $\left[-6\sqrt{3}, 6\sqrt{3}\right]$

In der Tat sind alle Werte der zweiten Ableitung ungleich 0 und damit die Existenz der Extrempunkte gesichert. Nun fehlen zum mathematischen Erfolgserlebnis nur noch die y-Koordinaten dieser Extrempunkte. Der `map`-Befehl bringt uns sofort ans Ziel.

- `ye := map(xe,f)`
  $\left[6\sqrt{3}, -6\sqrt{3}\right]$

Allerdings sind jetzt die x- und y-Koordinaten der Extrempunkte voneinander getrennt. Ein etwas komplizierter Befehl beseitigt auch dieses Problem (wenn es denn überhaupt eins ist). Mit dem `zip`-Befehl können die Elemente zweier Listen paarweise miteinander verknüpft werden, wobei gleichzeitig eine bestimmte Operation ausgeführt wird. Hier sollen die Elemente der beiden Listen mit den x- bzw. y-Koordinaten als eine Liste von Punkten zusammengeführt werden.

- `op( zip(xe,ye,(x,y)->[x,y]) )`
  $\left[-\sqrt{3}, 6\sqrt{3}\right], \left[\sqrt{3}, -6\sqrt{3}\right]$

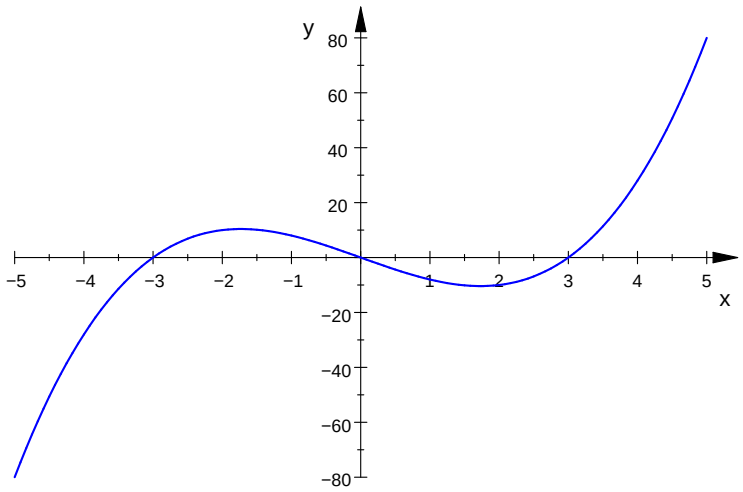
Wenn du das versteht, ist das vorzüglich. Wenn nicht, ist das allerdings auch nicht weiter schlimm. Du musst eigentlich nur wissen, das `xe` die Liste mit den x-Koordinaten und `ye` die Liste mit den y-Koordinaten ist. Das die Funktion als dritter Parameter jeweils aus den beiden Koordinaten eine neue Liste bildet, sei dann nur am Rande erwähnt. Und wem das alles immer noch reichlich mysteriös ist, muss eben auf diesen Komfort verzichten. Für die Kurvendiskussion ist dies nicht unbedingt erforderlich.

6 Zeichnen von Funktionen

6.1 Der plotfunc2d-Befehl

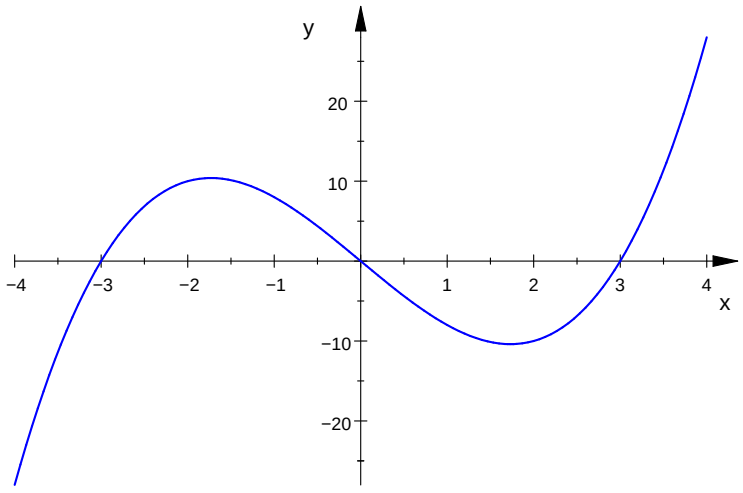
Die einfachste Möglichkeit, Funktiongraphen darzustellen, bietet der Befehl `plotfunc2d`. Er kann mit einem einzigen Argument, nämlich der Funktion aufgerufen werden.

- `plotfunc2d( f(x) )`



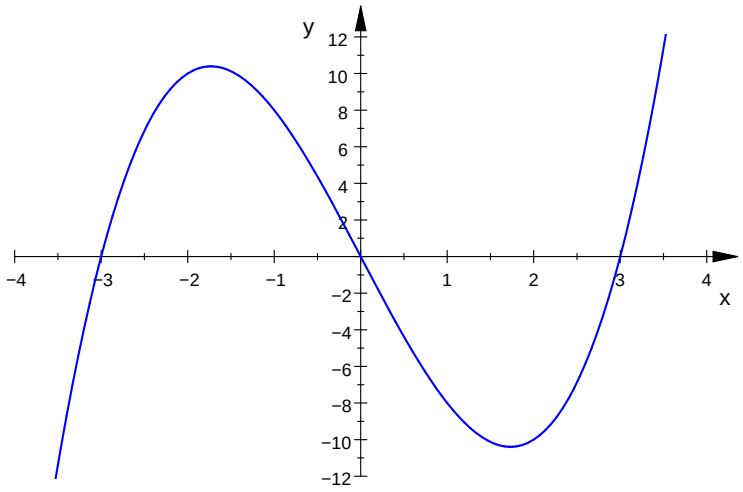
Mit diesem Befehl wird eine Funktion immer automatisch im Bereich von -5 bis 5 dargestellt. Falls ein anderer Bereich gewünscht wird, muss dies explizit angegeben werden.

- `plotfunc2d( f(x),x=−4..4 )`



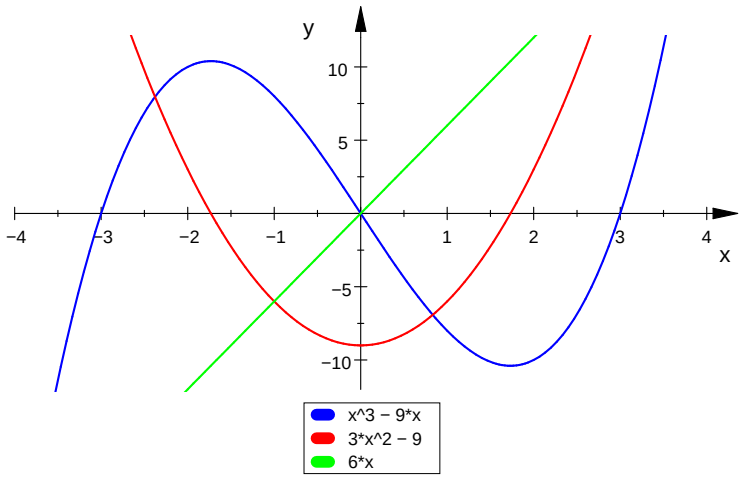
Als Wertebereich wird laut Voreinstellung immer der gesamte auftretende Bereich aller y-Werte gewählt. Dies ist in vielen Fällen nicht gewünscht und kann mit dem Parameter `YRange` abgeändert werden.

- `plotfunc2d( f(x),x=-4..4,YRange=-12..12 )`



Sollen mehrere Funktionen in ein Koordinatensystem eingezeichnet werden, müssen diese einfach nur nacheinander angegeben werden. Im folgenden Beispiel werden zusätzlich zu der Funktion noch die erste und zweite Ableitung gezeichnet.

- `plotfunc2d( f(x),f'(x),f''(x), x=-4..4,YRange=-12..12 )`



6.2 Attribute für plotfunc2d

Für die Darstellung von Funktionen gibt es eine Vielzahl von zusätzlichen Attributen, die die Darstellung beeinflussen. Einige sollen hier angegeben werden.

6.2.1 LineWidth

Dieser Parameter bestimmt die Dicke von Linienobjekten. Der Wert sollte als absolute physikalische Länge mit Längeneinheiten angegeben werden, z.B. `LineWidth=1.5*unit::mm`. Zahlen ohne Längeneinheit werden als mm interpretiert.

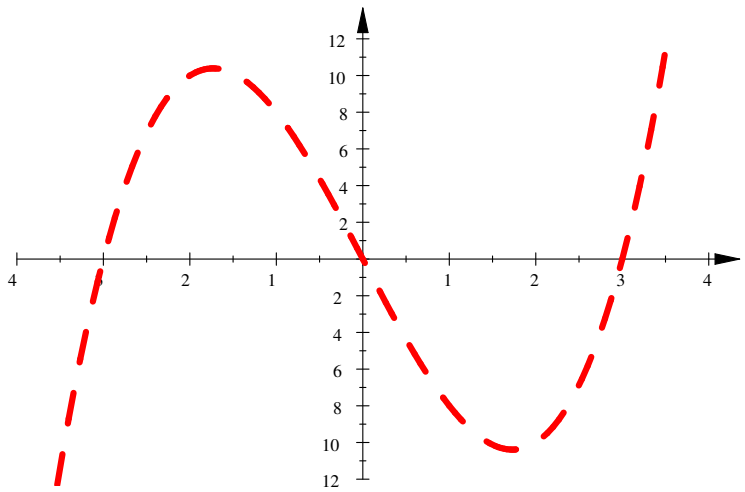
6.2.2 LineColor

Dieser Parameter legt die Farbe fest, in der Linienobjekte gezeichnet werden, z.B. `RGB::Red`.

6.2.3 LineStyle

Hiermit wird festgelegt, ob Linien durchgehend (`Solid`), gestrichelt (`Dashed`) oder gepunktet (`Dotted`) gezeichnet werden sollen.

- ```
  plotfunc2d( f(x),x=-4..4,YRange=-12..12,
      LineWidth=1.0, LineColor=RGB::Red,
      LineStyle=Dashed )
  ```

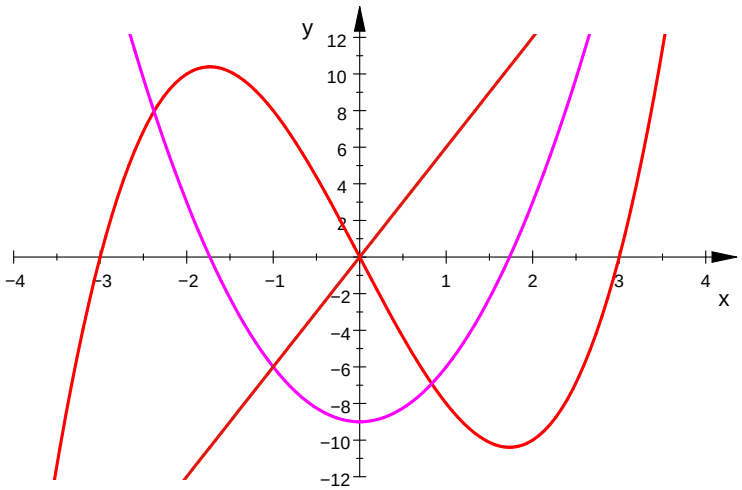


Allerdings sieht man schon an diesem Beispiel, das viele Programme gestrichelte oder gepunktete Linien nicht immer korrekt wiedergeben, wenn diese dazu auch noch etwas dicker gezeichnet werden.

6.2.4 Colors

Werden mehrere Funktionen gleichzeitig gezeichnet, kann mit `Colors` eine Liste von Farben festgelegt werden, die für die einzelnen Objekte benutzt werden. Zusätzlich unterdrücken wir die Legende, da uns die Bedeutung der einzelnen Graphen völlig klar sind.
  ```

- `plotfunc2d( f(x),f'(x),f''(x), x=-4..4,YRange=-12..12,`
  `Colors=[RGB::Red,RGB::Magenta,RGB::CadmiumRedDeep],`
  `LineWidth=0.5, LegendVisible=FALSE )`



Mit der Eingabe des Befehles `RGB::ColorNames()` liefert MuPAD eine Liste mit allen vordefinierten Namen von Farben.

- `RGB::ColorNames()`

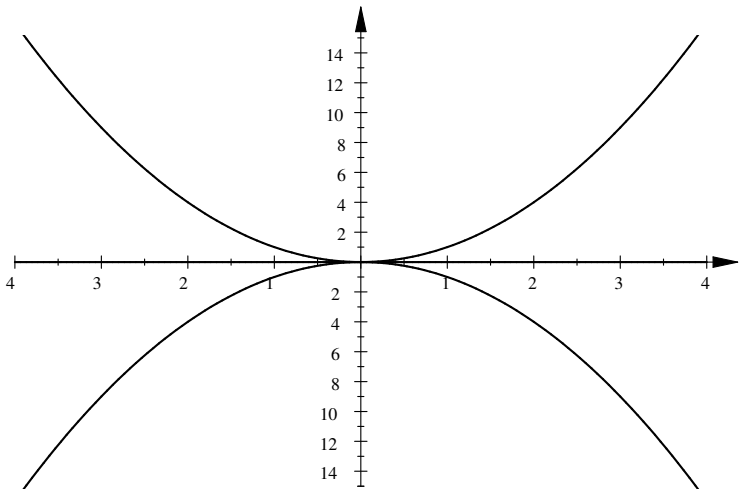
Da dieser Befehl z.Z. eine Liste von 287 unterschiedlichen Farben ausgibt, sei hier auf die Ausgabe verzichtet.

6.3 Funktionenscharen

Häufig treten in der Mathematik Funktionenscharen auf. In den folgenden Beispielen werden einige Funktionen der Funktionenschar $k \cdot x^2$ gezeichnet, die als Folge von Funktionen eingegeben wird. Auf die Ausgabe einer Legende wird in allen Beispielen verzichtet.

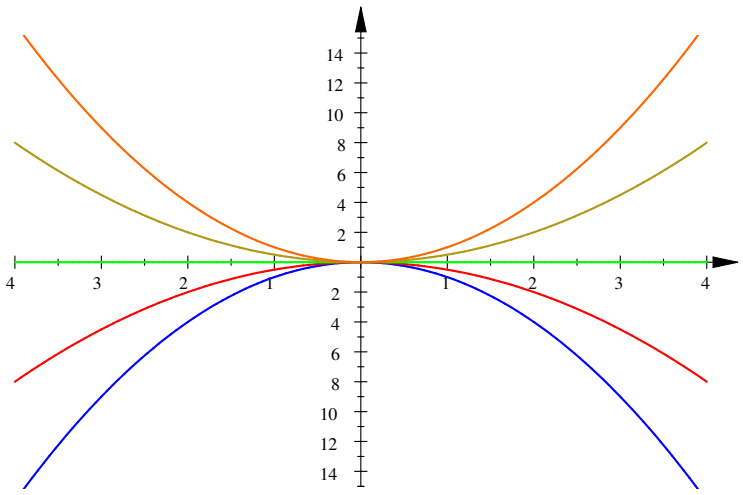
Im ersten Beispiel werden die Funktionen für die Parameter $k = -1$, $k = 0$ und $k = +1$ gezeichnet.

- `plotfunc2d(k*x^2 $ k=-1..1,`
  `x=-4..4, YRange=-15..15,`
  `LineColor=RGB::Black, LegendVisible=FALSE)`



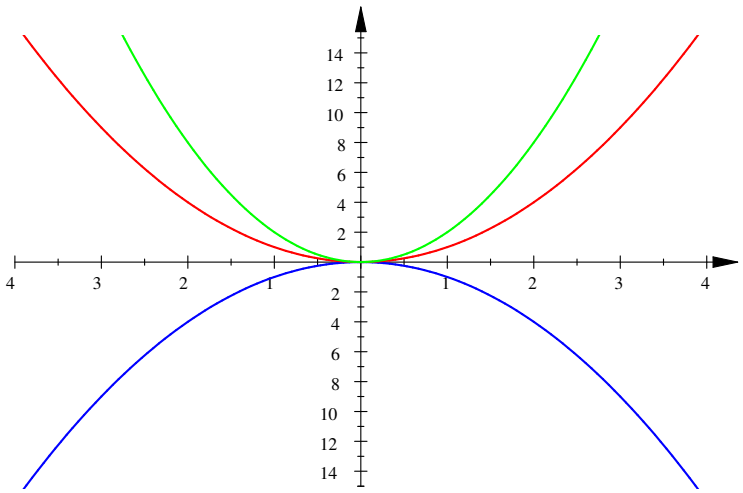
Dabei kann aber auch die Schrittweite beliebig verändert werden. Im nächsten Beispiel ändern sich die Werte für k jeweils um $0,5$.

- ```
  plotfunc2d(k*x^2 $ k=-1..1 step 0.5,
      x=-4..4, YRange=-15..15,
      LegendVisible=FALSE)
  ```



Weiterhin ist es möglich, unregelmäßig auftretende Parameter auszuwählen. Dies ist z.B. hier sinnvoll, da der Graph für $k = 0$ doch etwas aus dem Rahmen fällt. Im nachfolgenden Beispiel werden die Funktionen für $k = -1$, $k = +1$ und $k = +2$ gezeichnet.

- ```
  plotfunc2d(k*x^2 $ k in [-1,1,2],
      x=-4..4, YRange=-15..15,
      LegendVisible=FALSE)
  ```



6.4 Attribute für das Koordinatensystem

Für die Gestaltung der Graphiken stehen unzählige Attribute zur Verfügung. Die kleine Auswahl in den nachfolgenden Tabellen hilft meistens schon, wirklich ausdrucksstarke Graphiken zu erzeugen.

6.4.1 Koordinatenlinien

Auf Wunsch kann das Koordinatensystem mit Koordinatenlinien als Raster ausgegeben werden. Diese Koordinatenlinien befinden sich immer an den gewählten Markierungen.

GridVisible	Sichtbarkeit aller Koordinatenlinien als Raster Werte: `TRUE`, `FALSE` Voreinstellung: `FALSE` Varianten: `XGridVisible`,`YGridVisible`
GridLineWidth	Breite der Koordinatenlinien Voreinstellung: `0.1*unit::mm`
GridLineStyle	Darstellungsstil der Koordinatenlinien Werte: `Solid`, `Dashed`, `Dotted` Voreinstellung: `Solid`
GridLineColor	Farbe der Koordinatenlinien Werte: alle Farben Voreinstellung: `RGB::Grey75`

6.4.2 Koordinatenachsen

Sehr viele Befehle gibt es für die Gestaltung der Koordinatenachsen. Alle hier aufgeführten Befehle besitzen drei Varianten. Der Hauptbefehl gilt für beide Koordinatenachsen gleichzeitig.

Dagegen gelten Befehle, die mit einem `X` beginnen immer nur für die *x*-Achse und entsprechend Befehle mit einem `Y` immer nur für die *y*-Achse.

AxesVisible	Sichtbarkeit der Achsen Werte: `TRUE`, `FALSE` Voreinstellung: `TRUE` Varianten: `XAxisVisible`, `YAxisVisible`
AxesTitles	Achsentitel Werte: [string,string] Voreinstellung: ["x","y"] Varianten: `XAxisTitle`, `YAxisTitle`
AxesTitleAlignment	Ausrichtung der Achsentitel Werte: `Begin`, `Center` `End` Voreinstellung: `End` Varianten: `XAxisTitleAlignment`, `XAxisTitleAlignment`
YAxisTitleOrientation	Orientierung des Titels der y-Achse Werte: `Horizontal`, `Vertical` Voreinstellung: `Horizontal`
AxesLineColor	Farbe der Achsen Werte: alle Farben Voreinstellung: `RGB::Black`
AxesLineWidth	Breite der Achsen Voreinstellung: `0.18*unit::mm`
TicksNumber	Anzahl der beschrifteten Skalenmarkierungen Werte: `None`, `Low`, `Normal`, `High` Voreinstellung: `Normal` Varianten: `XTicksNumber`, `YTicksNumber`
TicksVisible	Sichtbarkeit der Skalenmarkierungen Werte: `TRUE`, `FALSE` Voreinstellung: `TRUE` Varianten: `XTicksVisible`, `YTicksVisible`
TicksLabelsVisible	Sichtbarkeit der zugehörigen Beschriftungen Werte: `TRUE`, `FALSE` Voreinstellung: `TRUE` Varianten: `XTicksLabelsVisible`, `YTicksLabelsVisible`
TicksAt	benutzerdefinierte Skalenmarkierungen Werte: `[tick1,tick2,...]` Voreinstellung: automatischh Varianten: `XTicksAt`, `YTicksAt`

Sollen die Markierungen mit einem der drei `TicksAt`-Befehle selbst gesetzt werden, muss die automatische Markierung zunächst ausgeschaltet werden. Als Liste kann dann entweder eine Liste von reellen Zahlen oder eine erweiterte Liste mit Paaren der Form `Wert=Beschriftung`

verwendet werden. In den folgenden zwei Beispielen werden benutzerdefinierte Markierungen für die x-Achse verwendet.

- `plotfunc2d(f(x),XTicksNumber=None,XTicksAt=[1,3,4])`
- `plotfunc2d(f(x),XTicksNumber=None,XTicksAt=[1="eins",3="drei",4="vier"])`

6.4.3 Skalierung

Ein weiterer wichtiger Parameter beeinflusst die Skalierung.

Scaling	Skalierung der Graphikausgabe Werte: `Automatic`, `Constrained`, `Unconstrained` Voreinstellung: `Automatic`

- Mit `Constrained` wird die Graphikausgabe gemäß den Koordinaten skaliert. Damit erscheinen Kreise auch als Kreise.
- Mit `Unconstrained` wird die Graphikausgabe in den einzelnen Koordinatenrichtungen so skaliert, dass das Bild optimal in den Ausgabebereich passt. Damit können allerdings Kreise u.U. als Ellipsen erscheinen.
- Mit der Voreinstellung `Automatic` überlassen wir die Entscheidung völlig MuPAD