Jasymca 2.0 - Symbolischer Rechner für Java

Helmut Dersch

15. März 2009

Zusammenfassung

Jasymca ist ein interaktives System zur Lösung mathematischer Aufgaben. Es werden beliebig genaue Zahlen und symbolische Variablen unterstützt. Skalare, Vektoren und Matrizen können aus allen Datentypen gebildet und bearbeitet werden. Von der Pseudoinversen einer symbolischen Matrix über trigonometrische Vereinfachungen bis zur symbolischen Lösung von Integralen oder Gleichungssystemen werden die wesentlichen Funktionen der gängigen CAS-Programme bereitgestellt. Daneben sind effiziente numerische Routinen des LAPACK sowie eine Plotfunktion implementiert. Die Bedienung läßt sich dynamisch umschalten zwischen einer Octave/Matlab/Scilab-artigen Sprache und einer an GNU-Maxima ausgerichteten Version. Jasymca ist in verschiedenen Versionen auf praktisch jeder Computer-Platform lauffähig: Als Midlet auf Mobiltelefonen und PDAs, als Java-Applikation auf fast jedem PC, oder auch als Applet in Webseiten integrierbar. Jasymca ist freie Software lizensiert gemäß GPL.

Inhaltsverzeichnis

1 Einleitung **3**

2 Arbeiten mit Jasymca **4**

2.1 Zahlen 4
2.2 Operatoren und Funktionen 7
2.3 Variablen 8
2.4 Vektoren und Matrizen (1) 11
2.5 Plotten 14
2.6 Polynome (1) 17

2.7 Dateien . . . 18
2.8 Programmierung . . . 19
2.8.1 Funktionen . . . 19
2.8.2 Verzweigungen . . . 20
2.8.3 Schleifen . . . 20
2.8.4 Sprungbefehle . . . 21
2.9 Vektoren und Matrizen (2) . . . 22
2.9.1 LAPACK . . . 24
2.10 Symbolische Variable . . . 27
2.11 Polynome (2) und gebrochen rationale Funktionen . . . 27
2.11.1 Nullstellen . . . 28
2.11.2 Quadratfreie Zerlegung . . . 29
2.11.3 Division, größter gemeinsamer Teiler . . . 29
2.11.4 Real- und Imaginärteil . . . 29
2.12 Symbolische Umformungen . . . 30
2.12.1 Substitution . . . 30
2.12.2 Vereinfachungen und Zusammenfassungen . . . 31
2.13 Gleichungen . . . 32
2.13.1 Lineare Gleichungssysteme . . . 32
2.13.2 Nichtlineare Gleichungen . . . 33
2.13.3 Nichtlineare Gleichungssysteme . . . 34
2.14 Infinitesimalrechnung . . . 35
2.14.1 Differentialquotient . . . 35
2.14.2 Taylorpolynom . . . 35
2.14.3 Unbestimmtes Integral . . . 36
2.14.4 Numerische Integration . . . 38
2.14.5 Differentialgleichungen . . . 38

3 Maxima-Modus **39**

4 Referenz **41**
4.1 Allgemeines . . . 41
4.2 Kommandos . . . 41
4.3 Operatoren . . . 42
4.3.1 Definieren . . . 42
4.3.2 Rechnen . . . 42
4.3.3 Vergleichen, Logik . . . 43
4.4 Programmierung . . . 44

4.5 Funktionen 45
4.5.1 Skalare 45
4.5.2 Skalare Funktionen 46
4.5.3 Vektoren und Matrizen 47
4.5.4 Polynome 48
4.5.5 Gleichungen und Ausdrücke 48
4.5.6 Infinitesimalrechnung 49
4.5.7 Plotten 49

5 Installation **50**

6 Lizenz **51**

1 Einleitung

Jasymca ist für die Mathematikausbildung entwickelt worden mit dem Ziel, den Zugang zur Computermathematik zu erleichtern. Damit soll vor allem das Hindernis der Taschenrechner überwunden werden, das viele Studenten zu lange davon abhält, sich das effiziente und professionelle mathematische Arbeiten an der Workstation anzueignen. Man kann gar nicht früh genug anfangen, alle Aufgaben dort zu bearbeiten. Hinderlich sind natürlich teure kommerzielle Programme, und die mangelnde Verfügbarkeit unterwegs. Jasymca kann hier den Einstieg liefern, denn es ist frei und läuft auf fast allem, was mit einem Prozessor bestückt ist, vom Mobiltelefon über PDA, Windows/Linux/MacOS-PC bis zu Spielkonsolen oder Internetroutern.

Jasymca 2.0 ist eine wesentliche Weiterentwicklung von Jasymca 1.01 [1]. Neben der neuen Grammatik (Jasymca 1.01 war Maxima [7]-orientiert) wurden vor allem Matrix- und Plotfunktionen hinzugefügt, sowie der Parser und Compiler völlig umgeschrieben. Voreingestellt ist jetzt eine Benutzeroberfläche, die sich an die Programme Octave [4], Matlab [5] und SciLab [6] anlehnt, ohne mit diesen jedoch in allen Details identisch zu sein. Ziel war es, daß Benutzer, die mit einem der genannten System vertraut sind, auch problemlos mit Jasymca zurechtkommen, und im Sinne der Ausbildung, natürlich vor allem auch umgekehrt. Zusätzlich sollte das Arbeiten mit symbolischen Größen vereinfacht werden, das in Jasymca nahtlos integriert ist.

Daneben kann man nach wie vor eine an Maxima [7] orientierte Bedienung verwenden, die bei einigen Aufgaben vorteilhaft ist, siehe dazu das Kapitel 3.

Kapitel 2 dieses Dokuments stellt eine Benutzeranleitung mit Beispielen und Übungsaufgaben dar, die teilweise auf einen Computer Mathematik Kurs für Ingenieurstudenten zurückgeht, und die als Einstieg durchgearbeitet werden kann. Es kann ohne weitere Installation mit dem Applet auf der Jasymcas-Homepage [8] gearbeitet werden. Anschließend wird der alternative Maxima-Modus kurz erläutert. Eine Übersicht aller Befehle und Optionen erfolgt in Kapitel 4, das als Referenz zum praktischen Arbeiten benutzt werden kann. Die Installation auf dem eigenen Rechner, Mobiltelefon, oder PDA wird dann in Kapitel 5 erklärt.

2 Arbeiten mit Jasymca

Ihr Rechner muß mit einer aktuellen Java-Version ($\geq 1,5$) ausgerüstet sein. Besuchen Sie einfach die Jasymca-Homepage [8], um das Programm zu starten. Natürlich können Sie im folgenden auch irgendeine der im Kapitel 5 vorgestellten Installationen verwenden.

Jasymca startet im Octave-Modus. Befehle werden mit der Tastatur im unteren Texteingabefeld eingegeben, die Ergebnisse im großen Textfeld darüber angezeigt und gespeichert. Mit der Zoom-Funktion im Hauptmenü lässt sich die Schriftgröße anpassen. Mit den Tasten < und > lassen sich frühere Eingaben wiederholen (Blättern im Verlaufsspeicher). Derselbe Effekt wird durch die Pfeiltasten der Tastatur erzielt, und ist eine wichtige Bedienhilfe für das effiziente Arbeiten. Durch Eingabe von `demoDE` wird eine 5-minütige Vorführung von Jasymca gestartet. In den folgenden Beispielen wird jeweils die protokollierte Jasymca-Ausgabe angegeben; zum nachrechnen bitte den Text nach dem Promptzeichen (>>) in das Texteingabefeld von Jasymca kopieren und auf *Eingabe* drücken.

2.1 Zahlen

Zahlen werden im üblichen Computerformat eingegeben: mit optionalem Dezimal*punkt* und gegebenenfalls mit dezimalem Exponenten nach dem Buchstaben `e` (oder `E`). Die Zahlen 5364 und $-1,723478265342 \cdot 10^{12}$ gibt man also so ein:

```
>> 5364
ans = 5364
>> -1.723478265342e12
ans = -1.7235E12
```

Zumeist werden diese intern als Gleitkomma-Datentypen angelegt (Double nach IEEE Standard 754). Angezeigt werden sie gerunded auf 5 Stellen, gerechnet wird aber mit der vollen Auflösung dieses Formats (15-16 Dezimalstellen). Durch Umschalten des Formats (`format long`) werden alle signifikanten Stellen angezeigt.

```
>> format long
>> -1.723478265342e12
ans = -1.723478265342E12
```

Als Erweiterung bietet Jasymca den Befehl `format Basis Anzahl`, mit dem die Anzeige in ein Zahlensystem mit beliebiger `Basis` und mit beliebiger `Anzahl` signifikanter Stellen erfolgt. So wird eine Zahl mit 15 Stellen im Dualsystem dargestellt:

```
>> format 2 15
>> -1.723478265342e12
ans = -1.1001000101001E40
```

Mit `format short` wird die kurze Anzeige im Dezimalsystem wiederhergestellt. Zu beachten ist, daß die `format`-Anweisung in keinem Fall die *interne* Genauigkeit oder Darstellung der Zahlen beeinflußt.

Zahlen, die ohne Dezimalpunkt und Exponent eingegeben werden, und größer als 10^{15} sind, werden als exakte rationale Zahlen gespeichert. Diese werden intern als Bruch zweier Ganzzahlen variabler Länge (Java Datentyp `BigInteger`) dargestellt, die das Rechnen ohne jegliche Rundungsfehler ermöglicht. Im folgenden Fall erzeugt die erste Eingabe eine Gleitkommazahl, die zweite eine exakte Zahl.

```
>> 10000000000000001.
ans = 1.0E16
>> 10000000000000001
ans = 10000000000000001
```

Jede Gleitkommazahl `Z` kann in eine exakte Zahl durch den Befehl `rat(Z)` verwandelt werden. Die Umwandlung erfolgt durch Kettenbruch-Entwicklung mit einer durch die Variable `ratepsilon` festgelegten Genauigkeit (voreingestellt: 10^{-8}).

```
>> rat(0.33333333333333333)
ans = 1/3
```

Operationen zwischen exakten und Gleitkommazahlen führen immer zu einer Umwandlung der Gleitkommazahl. Man kann also eine Rechnung rundungsfrei gestalten, indem man die erste Zahl „rationalisiert“.

```
>> 1/21/525/21/5*7*175*63*15-1
ans = -4.4409E-16
>> rat(1)/21/525/21/5*7*175*63*15-1
ans = 0
```

Umgekehrt werden durch den Befehl `float(Z)` Umwandlungen in das Gleitkommaformat bewirkt. Beide Befehle funktionieren auch mit zusammengesetzten Datentypen, etwa Polynomen oder Matrizen, deren Koeffizienten damit in einem Schritt transformiert werden. Irrationale Funktionswerte exakter Zahlen und Konstanten wie `pi` werden ebenfalls erst durch den `float`-Befehl in eine numerische Größe verwandelt.

```
>> sqrt(2)
ans = 1.4142
>> sqrt(rat(2))
ans = sqrt(2)
>> float(ans)
ans = 1.4142
```

Nützlich ist der exakte Datentyp vor allem bei instabilen Problemen, etwa beim Lösen von linearen Gleichungssystemen mit schlecht konditionierter Matrix. Ein extremes Beispiel sind die Hilbertmatrizen:

```
>> det( hilb(20)*invhilb(20) )
ans = 1           % richtig
>> det( float(hilb(20))*float(invhilb(20)) )
ans = 1.6713E151  % ziemlich falsch
```

Imaginäre Zahlen werden duch ein (ohne Leerzeichen) folgendes `i` oder `j` gekennzeichnet. Das funktioniert auch, wenn die vordefinierten Konstanten `i` oder `j` überschrieben wurden.

```
>> 2+3i
ans = 2+3i
```

2.2 Operatoren und Funktionen

Die Grundrechenarten werden mit den üblichen Symbolen (`+ - * /`) gebildet. Zur Exponentiation wird der Akzent (`^`) benutzt. Punkt- geht vor Strichrechnung, beliebige Reihenfolgen lassen sich durch runde Klammern erzwingen.

Übung 1 (Zahlen und Operatoren)

Berechnen Sie folgende mathematischen Ausdrücke:

$$3{,}23 \cdot \frac{14 - 2^5}{15 - (3^3 - 2^3)} \qquad 4{,}5 \cdot 10^{-23} : 0{,}0000013$$

$$17{,}4^{(3-2{,}13^{1{,}2})^{0{,}16}} \qquad \frac{17{,}23 \cdot 10^4}{1{,}12 - \frac{17{,}23 \cdot 10^4}{1{,}12 - \frac{17{,}23 \cdot 10^4}{1{,}12}}}$$

Lösung:

```
>> 3.23*(14-2^5)/(15-(3^3-2^3))
ans = 14.535
>> 4.5e-23/0.0000013
ans = 3.4615E-17
>> 17.4^((3-2.13^1.2)^0.16)
ans = 13.125
>> 17.23e4/(1.12-17.23e4/(1.12-17.23e4/1.12))
ans = 76919
```

Neben den Operatoren zum Rechnen können Vergleiche zwischen Zahlen (`< > >= <= == ~=`) angestellt werden, sowie Boolesche Verküpfungen gebildet werden (`& | ~`). Als wahr gilt die Zahl 1, falsch ist 0.

```
>> 1+eps>1
ans = 1
>> 1+eps/2>1      % So ist eps definiert
ans = 0
>> A=1;B=1;C=1;   %  Das Semikolon unterdrückt die Ausgabe
>> !(A&B)|(B&C) == (C~=A)
ans = 1
```

Die gebräuchlichsten implementierten Funktionen sind die Wurzel (`sqrt(x)`), die trigonometrischen Funktionen (`sin(x), cos(x), tan(x)`) mit Umkehrungen (`atan(x), atan2(y,x)`), sowie die hyperbolischen Funktionen (`exp(x), log(x)`). Eine Vielzahl weiterer Funktionen sind verfügbar, eine Liste befindet sich im Kapitel 4. Daneben sind Funktionen für Ganzzahlen implementiert, die auch mit beliebig großen Zahlen funktionieren: `primes(Z)` zerlegt `Z` in Primfaktoren, `factorial(Z)` berechnet die Fakultät. Modulare Division mit `divide` wird im Kapitel Polynome besprochen.

Beispiele: Funktionen

```
>> log(sqrt(854))         % Natürlicher Logarithmus
ans = 3.375
>> 0.5*log(854)
ans = 3.375
>> float(sin(pi/2))       % Argumente im Bogenmaß
ans = 1
>> gammaln(1234)          % log( gamma( x ) )
ans = 7547
>> primes(1000000000000000001)
ans = [ 101  9901  999999000001 ]
>> factorial(35)
ans = 1.0333E40
>> factorial(rat(35))     % So wird es exakt.
ans = 10333147966386144929666651337523200000000
```

2.3 Variablen

Variablen werden durch Angabe eines Namens und Wertes im Format `Name=Wert` angelegt. Der Name kann jede Buchstabenfolge sein. Bis auf das erste Zeichen dürfen auch Zahlen im Namen verwendet werden. Der Wert ist eine Zahl oder ein beliebiger Ausdruck aus Operatoren, Zahlen, Variablen, etc.

```
>> x=24+3i
x = 24+3i
```

Einige Variablen sind vordefiniert (z.B. `pi`). Das letzte Ergebnis ist als Variable `ans` abrufbar. Alle Variablen lassen sich mit dem Befehl `who` anzeigen. Einzelne Variablen lassen sich mit dem Kommando `clear Variable` löschen.

Es lassen sich auch Variablen bilden, deren Argumente Funktionen sind. Dazu muß der Funktionsname allerdings mit einem $-Zeichen vorangestellt werden. Diese Variablen können dann wie die Funktion eingesetzt werden. Wem z.B. die eingebaute Funktion `realpart(x)` einen zu langen Namen hat, kann sie auf den Namen der entsprechenden Matlab-Funktion kürzen:

```
>> real=$realpart
$realpart
>> real(24+3i)
ans = 24
```

Übung 2 (Variablen)

Rechnen Sie mehrere Temperaturen ($0°C, 20°C, 30°C, 50°C$) von Celsius in Fahrenheit-Grade um. Die Formel zur Berechnung von Fahrenheit-Graden lautet:

$$T/°F = 9/5 \cdot T/°C + 32$$

Benutzen Sie Variablen für die Temperaturen.
Lösung:

```
>> TC=20
TC = 20
>> TF=9/5*TC+32
TF = 68
>> TC=30
TC = 30
>> TF=9/5*TC+32   %  Wiederholen mittels Pfeiltasten
TF = 86
```

Übung 3 (Variablen)

Berechnen Sie die Oberfläche Ihres Körpers aus der Größe h und dem Gewicht W nach der DuBoisschen Formel:

$$A = h^{0,725}(cm) \cdot W^{0,425}(kg) \cdot 71,84 \cdot 10^{-4}(m^2)$$

Benutzen Sie Variablen h und W.
Lösung:

```
>> h=182; W=71;
>> A=h^0.725*W^0.425*71.84e-4
A = 1.9129
```

Übung 4 (Variablen)

Berechnen Sie einige Folgenglieder der rekursiv definierten Folge

$$x_0 = 1 \qquad\qquad x_{n+1} = \frac{1}{2}(x_n + \frac{3}{x_n})$$

Wann ist die Folge bis auf `2*eps` an den Grenzwert $\sqrt{3}$ herangerückt?
Lösung:

```
>> x=1
x = 1
>> x=1/2*(x+3/x); x-sqrt(3)   % n= 1
ans = 0.26795
>> x=1/2*(x+3/x); x-sqrt(3)   % n= 2
ans = 1.7949E-2
>> x=1/2*(x+3/x); x-sqrt(3)   % n= 3
ans = 9.205E-5
>> x=1/2*(x+3/x); x-sqrt(3)   % n= 4
ans = 2.4459E-9
>> x=1/2*(x+3/x); x-sqrt(3)   % n= 5
ans = 0                       % Bingo
```

2.4 Vektoren und Matrizen (1)

Diese Datentypen werden entweder zur Darstellung mehrdimensionaler Größen, oder zum gleichzeitigen Bearbeiten vieler Einzelwerte, etwa bei statistischen Problemen, verwendet. In diesem Kapitel wird nur dieser letzte Aspekt besprochen. Die Probleme der linearen Algebra und eigentlichen Vektorrechnung kommen später im Kapitel 2.9.

Vektoren werden durch eckige Klammern gekennzeichnet. Die Elemente werden als durch Kommas getrennte Liste eingefügt. Wenn die Unterscheidung eindeutig ist, können die Kommas weggelassen werden, was im zweiten Beispiel allerdings mißlingt:

```
>> x=[1,-2,3,-4]
x = [ 1  -2  3  -4 ]
>> x=[1 - 2  3 -4]       % Vorsicht: 1-2=-1
x = [ -1  3  -4 ]
```

Mit dem Doppelpunkt und der Funktion `linspace` lassen sich Zahlenbereiche als Vektoren definieren.

```
>> y=1:10               % 1 bis 10, Abstand 1
y = [ 1  2  3  4  5  6  7  8  9  10 ]
>> y=1:0.1:1.5          % 1 bis 1.5, Abstand 0.1
y = [ 1  1.1  1.2  1.3  1.4  1.5 ]
>> y=linspace(0,2,5)    % 5 von 0 bis 2.5, äquidistant.
y = [ 0  0.5  1  1.5  2 ]
```

Die Anzahl der Elemente im Vektor `x` ist mit der Funktion `length(x)` feststellbar, einzelne Elemente werden durch Nachstellung des Index `k` wie `x(k)` abgerufen. Dabei ist `k` eine Zahl im Bereich 1 bis einschließlich `length(x)`. Eine besondere Rolle spielt der Doppelpunkt („Magic Colon“). Als Index verwendet, werden alle Elemente als neuer Vektor bezeichnet. Zusätzlich können Bereiche indiziert werden.

```
>> y(2)                 % einzelnes Element
ans = 0.5
>> y(:)                 % magic colon
ans = [ 0  0.5  1  1.5  2 ]
>> y(2:3)               %  Index zwischen 2 und 3
ans = [ 0.5  1 ]
```

```
>> y(2:length(y))      %  Alles ab Index 2
ans = [ 0.5  1  1.5  2 ]
>> y([1,3,4])          %   Indices 1,3 und 4
ans = [ 0  1  1.5 ]
>> y([1,3,4]) = 9      %   Einfügen
ans = [ 9  0.5  9  9  2 ]
>> y([1,3,4]) = [1,2,3] % Einfügen
ans = [ 1  0.5  2  3  2 ]
```

Für Matrizen gilt ähnliches, wobei eben zwei Indices für Reihe (erster Index) und Spalte (zweiter Index) verwendet werden. Reihen werden bei der Eingabe durch ein Semikolon oder ein Zeilenende getrennt.

```
>> M=[1:3 ; 4:6 ; 7:9]
M =
  1  2  3
  4  5  6
  7  8  9
>> M([1 3],:)
ans =
  1  2  3
  7  8  9
>> C=M<4
C =
  1  1  1
  0  0  0
  0  0  0
```

Auf Vektoren und Matrizen können die Operatoren des Kapitel 2.2 angewandt werden. Wenn eine elementweise (skalare) Ausführung gewünscht wird, muß jedoch bei den Punktrechenarten (`* / ^`) ein Punkt vorangestellt werden, um die Operation von der ganz anderen Version der linearen Algebra (siehe Kapitel 2.9) mit gleichem Namen zu unterscheiden. Weitere nützliche und häufig verwendete Funktionen sind `sum(Vektor)` und `prod(Vektor)`, die Summe bzw. Produkt der Elemente eines Vektors berechnen.

Übung 5 (Vektoren)

Arbeiten mit Vektoren: Berechnen Sie

$$\sum_{k=1}^{k=n} k \qquad \sum_{k=1}^{k=n} k^2 \qquad \sum_{k=1}^{k=n} k^3$$

für $n = 10, n = 100, n = 10000$.
Lösung:

```
>> n=10; k=1:n;
>> sum(k), sum(k.*k), sum(k.^3)  % Punkt beachten!
ans = 55
ans = 385
ans = 3025
>> n=100; k=1:n;
>> sum(k), sum(k.*k), sum(k.^3)
ans = 5050
ans = 3.3835E5
ans = 2.5503E7
>> n=10000; k=1:n;
>> sum(k), sum(k.*k), sum(k.^3)
ans = 5.0005E7
ans = 3.3338E11
ans = 2.5005E15
```

Übung 6 (Vektoren)

Rechnen Sie die Liste der Celsiustemperaturen von $-30°C, -29°C, \ldots, 80°C$ in Fahrenheitgrade um. Benutzen Sie dazu Vektoren.
Lösung:

```
>> TC=-30:80;
>> TF=9/5*TC+32
TF = [ -22  -20.2  -18.4  -16.6  -14.8  -13  -11.2  -9.4 ..
```

2.5 Plotten

Daten lassen sich mit der `plot(x,y)`-Funktion graphisch darstellen, wobei x und y Vektoren gleicher Länge sind, die wiederum Koordinaten der zu zeichnenden Punkte sind. In einem dritten Argument `plot(x,y,Option)` kann eine Plotoption (Farbe, Symbole) spezifiziert werden, siehe Übungen 7ff. Die Graphik lässt sich anschließend dekorieren (Achsenbeschriftung, Titel etc), und als Datei zur Verwendung in einem Textdokument abspeichern. Die Speicheroption ist nur in der Applikation, nicht im Applet oder Midlet, verfügbar. Mit `hold` (oder `hold on`) kann eine Graphik fixiert werden, so daß der nächste Plotbefehl in diesselbe Grafik erfolgt. Mit erneutem `hold` (oder `hold off`) wird die Grafik gelöscht.

Doppelt oder einfach logarithmische Plots sind mit den Funktionen `loglog`, `linlog` und `loglin` verfügbar.

Übung 7 (Plotten)

Plotten Sie die Funktion $y = \frac{1}{1+2x^2}$ im Bereich $x = 0,01 \ldots 100$ linear und doppeltlogarithmisch.
Lösung:

```
>> x=0.01:0.01:100; y=1./(1+0.5*x.*x); plot(x,y)
>> x=0.01:0.01:100; y=1./(1+0.5*x.*x); loglog(x,y)
```

Übung 8 (Plotten)

Darstellung von Lissajous-Figuren: Erzeugen Sie aus dem Vektor `t=0:0.1:4*pi;` die trigonometrischen Ausdrücke `x=sin(0.5*t+1);` und `y=cos(1.5*t);`. Der Plot x gegen y ist eine *Lissajous*-Figur. Erzeugen Sie durch Variation der Konstanten `0.5,1,1.5` in den Definitionen von x und y andere Lissajous-Figuren.
Teillösung:

```
>> t=0:0.1:4*pi;
>> x=sin(0.5*t+1);
>> y=cos(1.5*t);
>> plot(x,y)
```

Übung 9 (Plotten)

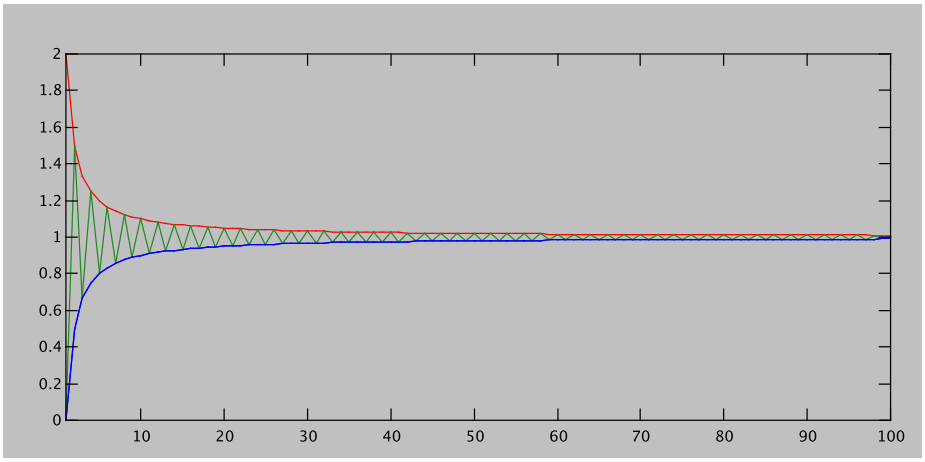
Berechnen Sie die ersten 100 Glieder der Folgen

$$x_n = \frac{n-1}{n}; x_n = \frac{n+1}{n}; x_n = \frac{n+(-1)^n}{n}$$

Plotten Sie x_n gegen n mit dem Befehl `plot(n, xn)`. Variieren Sie die Plotoptionen (Farben, Symbole).

Lösung:

```
>> n=1:100;
>> x1=(n-1)./n; x2=(n+1)./n; x3=(n+(-1).^n)./n;
>> plot(n,x1)        % Standard: blau, Linien
>> hold
Current plot held.
>> plot(n,x2,'r')    % Farbe:  r,g,b,c,m,y,w.
>> plot(n,x3,'g')
```



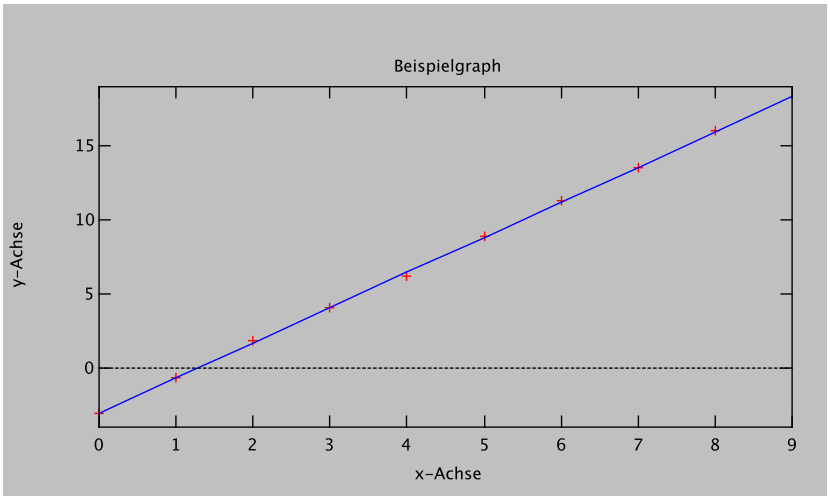
Übung 10 (Plotten)

Plotten Sie die Datenpunkte folgender Tabelle mit dem `plot`-Befehl als farbige Symbole. Berechnen Sie die Ausgleichsgerade mit der Funktion `polyfit`. Plotten Sie im selben Bild mit anderer Farbe die Ausgleichsgerade, die Sie als Lösung ermittelt haben. Fügen Sie Beschriftungen der Achsen und einen Titel ein, und speichern Sie den Plot in einem Format, das Sie in einer Textverarbeitung einfügen können.

x	0	1	2	3	4	5	6	7	8	9
y	-3.1	-0.7	1.8	4.1	6.2	8.9	11.3	13.5	16	18.3

Lösung:

```
>> x=0:9;
>> y=[-3.1,-0.7,1.8,4.1,6.2,8.9,11.3,13.5,16,18.3];
>> plot(x,y,"+r")    % Symbol: o,x,+,*
>> hold
Current plot held.
>> plot(x,polyval(polyfit(x,y,1),x)) % Ausgleichsgerade
>> xlabel("x-Achse")
>> ylabel("y-Achse")
>> title("Beispielgraph")
>> print("graph.eps") % nicht mit Applet oder Midlet!
```



2.6 Polynome (1)

Jasymca kann Polynome mit symbolischen Variablen verarbeiten. In diesem Kapitel wird jedoch die in Matlab/Oktave/SciLab vorgesehene Verwendung von Vektoren als Liste der Koeffizienten behandelt. Ein Polynom $n-$ten Grades wird dabei durch einen Vektor mit $n+1$ Elementen repräsentiert, wobei das Vektorelement mit Index 1 den Koeffizienten zur höchsten Potenz im Polynom ist. Mit `poly(x)` wird ein Polynom in Normalform berechnet, dessen Nullstellen die Elemente des Vektors `x` sind, `polyval(a,x)` berechnet Funktionswerte des Polynoms mit Koeffizienten `a` im Punkt `x`, `roots(a)` berechnet alle Nullstellen, und `polyfit(x,y,n)` berechnet die Koeffizienten des Polynoms *n*-ten Grades, dessen Graph durch die Punkte mit Koordinaten x und y verläuft. x und y sind gleichgroße Vektoren. Wenn deren Länge größer als $n+1$ ist, wird nach der Methode der kleinsten Fehlerquadrate das am besten passende Polynom bestimmt. So ist die Ausgleichsgerade in Übung 8 entstanden.

Übung 11

Von einem Polynom 4.ten Grades sind die Nullstellen $(-4,-2,2,4)$ und der y-Achsenabschnitt (-64) bekannt. Berechnen Sie die Koeffizienten.
2 Lösungen:

```
>> a=poly([-4,-2,2,4])
a = [ 1  0  -20  0  64 ]
>> a = -64/polyval(a,0) * a
a =
  -1   0    20   0    -64
>> a = polyfit([-4,-2,2,4,0],[0,0,0,0,-64],4)
a =
  -1   0    20   0    -64
```

Übung 12

Auf einem Gitter mit x,y-Koordinaten befinden sich folgende Grauwerte eines digitalen Bildes:

y\x	1	2	3	4
1	98	110	122	136
2	91	112	131	141
3	73	118	145	190
4	43	129	170	230

Welchen Grauwert erwarten Sie an Position $x = 2,35; y = 2,74$? Berechnen Sie die bikubische Interpolation.

Lösung:

```
>> Z=[98,110,122,136;
> 91,112,131,141;
> 73,118,145,190;
> 43,129,170,230];
>> i=1; p=polyfit(1:4,Z(i,:),3); z(i)=polyval(p,2.35);
>> i=2; p=polyfit(1:4,Z(i,:),3); z(i)=polyval(p,2.35);
>> i=3; p=polyfit(1:4,Z(i,:),3); z(i)=polyval(p,2.35);
>> i=4; p=polyfit(1:4,Z(i,:),3); z(i)=polyval(p,2.35);
>> p=polyfit(1:4,z,3); zp=polyval(p,2.74)
zp = 124.82
```

2.7 Dateien

Lesen und Schreiben von Dateien ist mit der Applikation einfacher, da Applets bei den üblichen Sicherheitseinstellungen keinen Zugriff auf die Festplatte besitzen. Zum Arbeiten mit Dateien stehen Menü-Funktionen zur Verfügung:

File : Open Script

Mit dieser Option wird ein Script in Jasymca eingelesen. Das Script muß eine Textdatei („Plain Text“) sein, die mit jedem gängigen Texteditor erstellt werden kann (z.B. Notepad, TextWrangler, Kwrite für die Plattformen Windows, MacOS, Linux). Der Inhalt dieser Datei wird verarbeitet, als wäre er im Texteingabefeld eingetippt worden, besteht also aus einer Liste von Jasymca-Kommandos.

Damit lassen sich Daten, etwa von einer automatischen Datenerfassung, zur Verarbeitung in Jasymca einlesen, oder aber selbstprogrammierte Funktionen laden, siehe Kapitel 2.8. Praktisch ist auch eine Liste von Natur- oder Materialkonstanten, die man zur Lösung seiner Aufgaben häufig benötigt.

Es ist auch möglich, Dateien von der Texteingabe aus zu laden. Dazu muß der Dateiname die Endung „.m“ besitzen (also z.B. „Daten.m“). Außerdem muß die Datei in einem Verzeichnis im Suchpfad von Jasymca liegen (siehe unten). Man gibt dann den Namen *ohne* die Endung „.m“ in der Texteingabe ein (also hier: `Daten`).

File : Save History

Mit diesem Befehl wird ein Protokoll der Sitzung abgelegt, und zwar alle Einträge im Verlaufsspeicher. Diese können später mit „Open Script“ wieder gelesen werden.

File : Add Path

Der Suchpfad legt fest, in welchen Verzeichnissen Jasymca nach Dateien sucht, wenn diese geladen werden sollen. Mit `path` kann die Liste dieser Verzeichnisse angezeigt werden. Es wird nicht in Unterverzeichnissen gesucht. Soll ein Verzeichnis zum Suchpfad hinzugefügt werden, so kann dies durch die genannte Menüoption erfolgen, oder aber durch das Kommando `addpath('Pfad')`. Der Name des Verzeichnisses muß in Anführungszeichen stehen.

```
>> path
m:.
>> addpath('/Users/dersch/jasymca')
>> path
/Users/dersch/jasymca:m:.
```

2.8 Programmierung

2.8.1 Funktionen

Programme können interaktiv erstellt und ausgeführt werden. Die Programmierung einer Funktion erfolgt wie in diesem Beispiel der Funktion `ttwo(x)`, die ihr Argument mit 2 multipliziert. Nach der Definition kann sie wie jede eingebaute Jasymca-Funktion verwendet werden.

```
>> function y=ttwo(x) y=2*x; end
>> ttwo(3.123)
ans = 6.246
```

Nach dem Schlüsselwort `function` kommt der Prototyp der Funktion mit einer Rückgabevariablen `y`. Dadurch wird das in anderen Programmiersprachen übliche `return y` ersetzt.

Wenn solche Funktionen wiederverwendet werden sollen, kann man sie in einer Textdatei im Suchpfad abspeichern. Der Name der Datei muß der Name der Funktion erweitert um die Endung „.m“ sein, also in unserem Fall `ttwo.m`. Dann kann in späteren Sitzungen direkt `ttwo` als Funktion verwendet werden, die Programmdefinition wird automatisch gesucht und geladen. Viele der installierten Jasymca-Funktionen sind als solche m-Dateien angelegt.

2.8.2 Verzweigungen

`if x A end`

Abhängig von einer Bedingung `x` wird eine oder mehrere Anweisung(en) `A` ausgeführt. Dabei muß `x` `1` oder `0` ergeben. Der Fall, das die Bedingung falsch (also 0) ist, kann zu einer zweiten Anweisung `B` führen:

`if x A else B end`

Als Beispiel die Heavyside Funktion:

```
>> function y=H(x)
>    if (x>=0)
>      y=1;
>    else
>      y=0;
>    end
>  end
>> H(-2)
y = 0
>> H(0)
y = 1
```

2.8.3 Schleifen

Schleifen mit Bedingung `x` und Anweisung(en) `A`:

`while x A end`

Die `while`-Schleife wird so oft wiederholt, bis die Bedingung `x` falsch (`0`) wird.

```
>> x=1;y=1;
>> while(x<10) y=x+y; x++; end
>> y
y = 46
```

Schleifen mit Zähler `z` und Anweisung(en) `A`:

```
for  z = Vektor   A   end
```

Bei der `for`-Schleife wird der Zähler `z` mit einem `Vektor` initialisiert. Tatsächlich nimmt der Zähler mit jedem Durchlauf den Wert des nächsten Elements von `Vektor` an.

```
>> x=1;y=1;
>> for(x=1:0.1:100) y=x^2+y; end
>> y
y = 3.3383E6
```

2.8.4 Sprungbefehle

`return, continue, break`

Mit `return` wird eine Funktion vorzeitig verlassen. `continue` und `break` werden in Schleifen verwendet: `continue` überspringt den Rest der Schleife zurück zur Bedingung, die Schleife wird also wiederholt. Mit `break` verläßt man vorzeitig die Schleife.

```
>> x=1;
>> while( 1 )
>    if(x>1000)
>       break;
>    end
>    x++;
> end
>> x
x = 1001
```

2.9 Vektoren und Matrizen (2)

Verschiedene Standardmatrizen können ohne Eingabe der Elemente durch Funktionen berechnet werden: `ones(n,m)`, `zeros(n,m)`, `rand(n,m)` ergeben Matrizen mit Elementen 1, 0 oder Zufallszahlen zwischen 0 und 1. `eye(n,m)` hat 1 auf der Diagonalen, sonst 0, und mit `hilb(n)` kann die Hilbert-Matrix *n*-ten Grades bestimmt werden.

```
>> A=rand(1,3)
A =
  0.33138  0.94928  0.56824
>> B=hilb(4)
B =
  1    1/2  1/3  1/4
  1/2  1/3  1/4  1/5
  1/3  1/4  1/5  1/6
  1/4  1/5  1/6  1/7
```

Zur Bearbeitung stehen die Funktionen `diag(x)` (extrahiert die Diagonale), `det(x)` (Determinante), `eig(x)` (Eigenwerte), `inv(x)` (Inverse), `pinv(x)` (Pseudoinverse) zur Verfügung. Die Adjunkte wird durch den Operator `'` berechnet.

```
>> det(hilb(4))
ans = 1/6048000
>> M=[2 3 1; 4 4 5; 2 9 3];
>> M'
ans =
  2  4  2
  3  4  9
  1  5  3
>> eig(M)
ans = [ 11.531  -3.593  1.062 ]
>> inv(M)
ans =
  0.75         0            -0.25
  4.5455E-2    -9.0909E-2   0.13636
  -0.63636     0.27273      9.0909E-2
```

Die nichttrivialen Funktionen beruhen alle auf der LU bzw LR-Zerlegung, die als Funktion ebenfalls abrufbar ist durch `lu(x)`. Diese Funktion hat zwei oder

drei Rückgabewerte, daher müssen in der entsprechenden Gleichung auf der linken Seite zwei oder drei Variablen in einem Vektor zur Verfügung gestellt werden.

```
>> M=[2 3 1; 4 4 5; 2 9 3]
>> [l,u,p]=lu(M)         % 2 oder 3 Rückgabewerte
l =                      % Linke Dreiecksmatrix (permutiert)
  0.5      0.14286  1
  1        0        0
  0.5      1        0
u =                      % Rechte, obere Dreiecksmatrix
  4        4        5
  0        7        0.5
  0        0        -1.5714
p =                      % Permutationsmatrix
  0  0  1
  1  0  0
  0  1  0
```

Wenn keine Punkte vorangestellt sind, werden mit den Symbolen der Grundrechenarten die korrespondierenden Matrix und Vektoroperationen ausgelöst, also Vektor- und Matrixmultiplikation im Falle von `*`.

```
>> x=[2,1,4]; y=[3,5,6];
>> x.*y      %  Mit Punkt
ans = [ 6  5  24 ]
>> x*y       %  Ohne Punkt
ans = 35
```

Ist eines der Argumente ein Skalar, so erfolgt die Operation mit jedem Element des Vektors/der Matrix einzeln.

```
>> x=[2,1,4];
>> x+3
ans = [ 5  4  7 ]
```

Matrixdivision entspricht der Multiplikation mit der Pseudoinversen. Mit dem Operator \ ist die Linksdivsion möglich, die als Transponierte des Quotienten der Transponierten berechnet wird. Damit ist z.B. die Lösung eines linearen Gleichungssystems möglich:

```
>> M=[2 3 1; 4 4 5; 2 9 3];
>> b=[0;3;1];
>> x=M\b      % Lösung von M*x = b
x =
  -0.25
  -0.13636
  0.90909
>> M*x        % Kontrolle
ans =
  0
  3
  1
```

Lineare Gleichungssysteme können (und sollten) direkt mit dem Befehl `linsolve(A,b)` gelöst werden, der im Zusammenhang mit Gleichungen (Kapitel 2.13.1 ausführlicher besprochen wird.

2.9.1 LAPACK

Die Applikation Jasymca (nicht das Applet und das Midlet) enthält JLAPACK [9], die Java-Portierung der LAPACK [10]-Routinen mit umfangreicheren und besseren Algorithmen für Matrixberechnungen. Allerdings sind diese auf Matrizen mit reellen Koeffizienten im Gleitkommaformat beschränkt. Aufgerufen werden die LAPACK Routinen durch:

svd(*A*) Singulärwertzerlegung von *A* (1 oder 3 Rückgabewerte).

```
>> A=[2 3 1; 4 4 5; 2 9 3];
>> svd(A)
ans = [ 12.263  3.697  0.9705 ]
```

qr(*A*) QR-Zerlegung von *A* (2 Rückgabewerte).

```
>> A=[2 3 1; 4 4 5; 2 9 3];
>> [q,r]=qr(A)
q =
  -0.40825    -5.3149E-2  -0.91132
  -0.8165     -0.4252     0.39057
  -0.40825    0.90354     0.13019
```

```
r =
  -4.899   -8.165   -5.7155
  0        6.2716   0.53149
  0        0        1.4321
```

linsolve2(*A, b*) Löst $A \cdot x = b$ (1 Rückgabewert). Beispiel in Kapitel 2.13.1.

linlstsq(*A, b*) Löst $A \cdot x = b$, überbestimmt (1 Rückgabewert). Für ein Beispiel siehe Kasten „Vergleich der Jasymca und LAPACK Routinen“.

eigen(*A*) Eigenwerte von A (1 Rückgabewert).

```
>> A=[2 3 1; 4 4 5; 2 9 3];
>> eigen(A)
ans = [ 11.531  1.062  -3.593 ]
```

Vergleich der Jasymca und LAPACK Routinen

Wir berechnen eine Ausgleichskurve 4. Grades:

```
>> x=[1:6],y=x+1
x = [ 1  2  3  4  5  6 ]
y = [ 2  3  4  5  6  7 ]
>> polyfit(x,y,4)
p =
  5.1958E-14   -9.6634E-13  -2.4727E-12  1   1
```

Die Koeffizienten `p(1),p(2),p(3)` sollten natürlich verschwinden, da `x` und `y` eine perfekte Gerade bilden. Dies ist ein numerisch instabiles Problem, und es lässt sich leicht so erweitern, daß die Jasymca-Routinen völlig versagen. Im zweiten Versuch verwenden wir die Lapack-Routine `linlstsq`:

```
>> x=[1:6],y=x+1;
>> l=length(x);n=4;
>> X=(x'*ones(1,n+1)).^(ones(l,1)*(n:-1:0))
>> linlstsq(X,y')
ans =
  -1.6288E-18
  -7.0249E-17
  1.0653E-15
  1
  1
```

Die Koeffizienten `p(1),p(2),p(3)` sind jetzt erheblich kleiner. Dieses besondere Problem lässt sich übrigens mit den Jasymca-Routinen exakt lösen, indem man mit exakten Zahlen arbeitet, wobei jetzt keine Rundungsfehler entstehen können:

```
>> x=rat([1:6]);y=x+1;
>> polyfit(x,y,4)
p =
  0  0  0  1  1
```

2.10 Symbolische Variable

Im Gegensatz zu den Vorbildern Octave und Matlab ist das Rechnen mit symbolischen Größen von Grund auf in Jasymca integriert, und nicht, wie dort, nachträglich unter Verwendung besonderer Befehle hinzugefügt. Das bedeutet, wenn Operationen für symbolische und nichtsymbolische algebraische Objekte erlaubt sind, so wird auch immer derselbe Befehl oder Operator verwendet. In der Benutzung wird also kein Unterschied zwische symbolischen und nichtsymbolischen Größen gemacht. Es tritt nur gelegentlich eine Warnung auf, wenn eine Operation nur für einen der Datentypen möglich ist.

Symbolische Variable sind zu unterscheiden von den bisher besprochenen Variablen. Während letztere nur als Adresse für ein Objekt im Speicher (dem „Environment“) dienen, sind symbolische Variable selbst Teil algebraischer Objekte. Wenn also `x` eine normale Variable ist, so wird bei der Eingabe von `x` das dazu passende Objekt im Speicher gesucht und `x` durch dieses ersetzt. Ist `x` dagegen eine symbolische Variable, so erzeugt die Eingabe von `x` ein Polynom ersten Grades mit der Variablen `x` und den Koeffizienten 1 und 0.

Im Octave-Modus muß eine symbolische Variable `x` vor Verwendung mit dem Befehl `syms x` als solche deklariert werden. Der schon früher eingeführte Befehl `clear x` löscht sie wieder.

```
>> x=3;             % Nichtsymbolische Variable
>> xˆ2+3-2*sin(x)   % Dient als Platzhalter für ’3’
ans = 11.718
>> syms x           % Symbolische Variable
>> xˆ2+3-2*sin(x)   % Erzeugt eine Funktion
ans = -2*sin(x)+(xˆ2+3)
```

2.11 Polynome (2) und gebrochen rationale Funktionen

Früher haben wir gesehen, daß ein Polynom durch seinen Koeffizientenvektor dargestellt werden kann. Mit einer Variablen `x` läßt sich jetzt einfach ein symbolisches Polynom `p` aufbauen. Umgekehrt können die Koeffizienten des symbolischen Polynoms durch die Funktion `coeff(p, x, Exponent)` extrahiert werden. Mit dem Befehl `allroots(p)` werden die Nullstellen berechnet.

```
>> a=[3 2 5 7 4];   % Koeffizienten
>> syms x
>> y=polyval(a,x)   % Symbolisches Polynom
```

```
y = 3*x^4+2*x^3+5*x^2+7*x+4
>> coeff(y,x,3)     % Ein einzelner Koeffizient
ans = 2
>> b=coeff(y,x,4:-1:0)  % oder alle auf einmal
b = [ 3  2  5  7  4 ]
>> allroots(y)      % liefert dasselbe wie roots(a)
ans = [ 0.363-1.374i  0.363+1.374i
          -0.697-0.418i  -0.697+0.418i ]
```

Bis hier liefert das symbolische Rechnen keinen Vorteil, sondern ist nur eine andere Schreibweise. Die Vorteile ergeben sich, wenn ein Polynom mehrere Variablen, bzw. nichtkonstante Koeffizienten besitzt. Dieser Fall kann nur mit symbolischen Variablen behandelt werden. Beachten Sie, wie das Polynom y im folgenden Beispiel automatisch ausmultipliziert wird, und in eine kanonische Form gebracht wird. Dabei werden die Variablen umgekehrt alphabetisch sortiert, d.h. `z` ist Hauptvariable gegenüber `x`. Die Koeffizienten können dann für jede Variable getrennt extrahiert werden.

```
>> syms x,z
>> y=(x-3)*(x-1)*(z-2)*(z+1)
y = (x^2-4*x+3)*z^2+(-x^2+4*x-3)*z+(-2*x^2+8*x-6)
>> coeff(y,x,2)
ans = z^2-z-2
>> coeff(y,z,2)
ans = x^2-4*x+3
```

2.11.1 Nullstellen

Der schon beschriebene Befehl `allroots` funktioniert auch bei variablen Koeffizienten, allerdings dann nur, wenn der Grad des Polynoms `p` in der Hauptvariablen kleiner als 3 ist, oder es biquadratisch ist. Werden Nullstellen anderer Variablen `x` gesucht, wird die universellere `solve(p,x)` Funktion verwendet, die später noch ausführlicher betrachtet wird.

```
>> syms x,z
>> y = x*z^2-3*x*z+(2*x+1);
>> allroots(y)
ans = [ sqrt((1/4*x-1)/x)+3/2  -sqrt((1/4*x-1)/x)+3/2 ]
>> solve(y,x)
ans = -1/(z^2-3*z+2)
```

2.11.2 Quadratfreie Zerlegung

Eine Zerlegung des Polynoms `p` in lineare, quadratische, kubische usw Faktoren wird mit `sqfr(p)` erzielt. Rückgabewert ist ein Vektor der Faktoren sortiert nach aufsteigenden Exponenten.

```
>> syms x
>> y=(x-1)^3*(x-2)^2*(x-3)*(x-4)
y = x^7-14*x^6+80*x^5-242*x^4+419*x^3-416*x^2+220*x-48
>> z=sqfr(y)
z = [ x^2-7*x+12  x-2  x-1 ]
```

2.11.3 Division, größter gemeinsamer Teiler

Die Division zweier Polynome `p` und `q` in ein Polynom und einen Rest wird mit `divide(p,q)` erzielt. Bei Polynomen mit mehreren Variablen kann die Variable angegeben werden, nach der die Division erfolgen soll. Der größte gemeinsame Teiler zweier Ausdrücke wird mit `gcd(p,q)` berechnet. Beide Funktionen arbeiten auch mit Zahlen als Argument.

```
>> divide(122344,7623)
ans = [ 16  376 ]
>> divide(2+i,3+2*i)
ans = [ 1/2  1/2 ]
>> syms x,z
>> divide(x^3*z-1,x*z-x,x)
ans = [ x^2*z/(z-1)  -1 ]
>> divide(x^3*z-1,x*z-x,z)
ans = [ x^2  x^3-1 ]
>> gcd(32897397,24552502)
ans = 377
>> gcd(z*x^5-z,x^2-2*x+1)
ans = x-1
```

2.11.4 Real- und Imaginärteil

Mit `realpart(Ausdruck)` und `imagpart(Ausdruck)` können komplexe Ausdrücke zerlegt werden. Sind symbolische Variablen vorhanden, so werden diese als reellwertig angenommen.

```
>> syms x
>> y=(3+i*x)/(2-i*x)
y = (-x+3i)/(x+2i)
>> realpart(y)
ans = (-x^2+6)/(x^2+4)
>> imagpart(y)
ans = 5*x/(x^2+4)
```

2.12 Symbolische Umformungen

2.12.1 Substitution

Mit `subst(a,b,c)` können beliebige Ausdrücke für symbolische Variable substituiert werden. Dabei wird `a` für `b` in `c` eingesetzt. Dies ist ein leistungsfähiger Befehl mit vielen Anwendungsmöglichkeiten. Zunächst kann man ihn verwenden, um Zahlen für Variablen einzusetzen, hier 3 für x in der Formel $2\sqrt{x}\cdot e^{-x^2}$.

```
>> syms x
>> a=2*sqrt(x)*exp(-x^2);
>> subst(3,x,a)
ans = 4.275E-4
```

Man kann aber auch komplexere Ausdrücke für eine Variable einsetzen, hier z^3+2 für x in x^3+2x^2+x+7.

```
>> syms x,z
>> p=x^3+2*x^2+x+7;
>> subst(z^3+2,x,p)
ans = z^9+8*z^6+21*z^3+25
```

Schließlich kann auch `b` selber ein komplexer Ausdruck (im Beispiel z^2+1) sein, den Jasymca dann versucht, in `c` (im Beispiel $\frac{z\cdot x^3}{\sqrt{z^2+1}}$) zu identifizieren und entsprechend durch `a` (hier: y) zu ersetzen. Dazu wird mit dem Gleichungslöser die Beziehung $a=b$ nach der in `b` verborgenen Variablen (hier z) aufgelöst, und die Lösung dann in `c` verwendet. Das gelingt natürlich nicht immer, oder es kann mehrfache Lösungen geben, die dann als Vektor angegeben werden.

```
>> syms x,y,z
>> c=x^3*z/sqrt(z^2+1);
>> d=subst(y,z^2+1,c)
```

```
d = [ x^3*sqrt(y-1)/sqrt(sqrt(y-1)^2+1)
     -x^3*sqrt(y-1)/sqrt(sqrt(y-1)^2+1) ]
>> d=trigrat(d)
d = [ x^3*sqrt(y-1)/sqrt(y)
     -x^3*sqrt(y-1)/sqrt(y) ]
```

2.12.2 Vereinfachungen und Zusammenfassungen

Der Befehl `trigrat`(*Ausdruck*) wendet eine Serie von Algorithmen auf einen *Ausdruck* an:

- Zunächst werden alle Zahlen in exakte Zahlen überführt.
- Trigonometrische Ausdrücke werden in Exponentialform umgewandelt.
- Additionstheoreme der Exponential- und Logarithmusfunktion werden angewendet.
- Wurzelausdrücke werden, wenn möglich, ausgeführt.
- Komplexe Exponentialausdrücke werden zurück in trigonometrische Funktionen verwandelt.

Anschließend muß häufig mit `float(Ausdruck)` wieder in Gleitkommazahlen umgeformt werden.

```
>> syms x
>> trigrat(sin(x)^2+cos(x)^2)
ans = 1
>> b=sin(x)^2+sin(x+2*pi/3)^2+sin(x+4*pi/3)^2;
>> trigrat(b)
ans = 3/2
>> trigrat(i/2*log(x+i*pi))
ans = 1/4*i*log(x^2+pi^2)+(1/2*atan(x/pi)-1/4*pi)
>> trigrat(sin((x+y)/2)*cos((x-y)/2))
ans = 1/2*sin(y)+1/2*sin(x)
>> trigrat(sqrt(4*y^2+4*x*y-4*y+x^2-2*x+1))
ans = y+(1/2*x-1/2)
```

Mit `trigexpand(Ausdruck)` werden trigonometrische Ausdrücke in Exponentialform umgewandelt. Dies ist der erste Schritt der `trigrat` Funktion.

```
>> syms x
>> trigexp(i*tan(i*x))
ans = (-exp(2*x)+1)/(exp(2*x)+1)
>> trigexp(atan(1-x^2))
ans = -1/2*i*log((-x^2+(1-1*i))/(x^2+(-1-1*i)))
```

2.13 Gleichungen

2.13.1 Lineare Gleichungssysteme

Zur Lösung linearer Gleichungssysteme stehen die Befehle `linsolve(A,b)` und `linsolve2(A,b)` (LAPACK, nicht im Applet und Midlet) zur Verfügung. In beiden Fällen ist A die quadratische Matrix des Gleichungssystems, und b ein Zeilen- oder Spaltenvektor, gelöst wird dann die Matrixgleichung $A \cdot z = b$ nach z.

```
>> A=[2 3 1; 4 4 5; 2 9 3];
>> b=[0;3;1];
>> linsolve(A,b)
ans =
  -0.25
  -0.13636
  0.90909
>> linsolve2(A,b) % Nicht Applet oder Midlet
ans =
  -0.25
  -0.13636
  0.90909
```

Für größere numerische Matrizen sollte die LAPACK-Variante verwendet werden. Die Jasymca-Version kann allerdings auch mit exakten und symbolischen Ausdrücken arbeiten. Um Rundungsfehler zu vermeiden, sollte dann immer mit exakten Größen operiert werden:

```
>> syms x,y
>> A=[x,1,-2,-2,0;1 2 3*y 4 5;1 2 2 0 1;9 1 6 0 -1;0 0 1 0]
A =
  x    1    -2   -2   0   % symbolisches Matrixelement
  1    2    3*y  4    5   % symbolisches Matrixelement
  1    2    2    0    1
```

```
  9    1    6    0    -1
  0    0    1    0    0
>> b = [1 -2  3  2  4 ];
>> trigrat( linsolve( rat(A), b) )

ans =
  (-6*y-13/2)/(x+8)
  (20*y+(-9*x-151/3))/(x+8)
  4
  ((-3*x+10)*y+(-49/4*x-367/6))/(x+8)
  (-34*y+(13*x+403/6))/(x+8)
```

2.13.2 Nichtlineare Gleichungen

Unter Gleichung wird im folgenden immer die Gleichung `Ausdruck = 0` verstanden. Mit dem `solve(Ausdruck, x)` Befehl wird eine Gleichung nach einer symbolischen Variablen `x` aufgelöst. Ist `Ausdruck` eine gebrochene Funktion, wird `Zähler = 0` gelöst. Die mit Jasymca lösbaren Probleme lassen sich anhand der internen Lösungsstrategie angeben:

1. Zunächst wird abgezählt, wie oft `x` in `Ausdruck` vorkommt, und zwar als freie Variable und als Variable innerhalb eingebetteter Funktionen. Beispiel: In $x^3 \cdot \sin(x) + 2x^2 - \sqrt{x-1}$ kommt `x` dreimal vor: als freie Variable, in $\sin(x)$ und in $\sqrt{x-1}$.

2. Wenn die Variable einmal vorkommt, handelt es sich um einen Polynom-Ausdruck, der zunächst nach *seiner* Variablen, z.B. `z`, gelöst wird. Dies gelingt bis zum Grad $n = 2$, sowie bei mehrfach quadratischen Ausdrücken immer, bei anderen nur, wenn die Koeffizienten konstant sind. Die Lösung wird im nächsten Schritt dann nach der gesuchten Variablen `x` aufgelöst. Wenn also die Gleichung $\sin^2(x) - 2\sin(x) + 1 = 0$ zu lösen ist, so wird die Gleichung $z^2 - 2z + 1 = 0$ nach z und dann $\sin(x) = z$ nach x aufgelöst. Zunächst Beispiele mit einer freien Variablen:

   ```
   >> syms x,b
   >> solve(x^2-1,x)
   ans = [ 1  -1 ]
   >> solve(x^2-2*x*b+b^2,x)
   ans = b
   ```

Und ein Beispiel mit einer Funktionsvariablen (hier $exp(j \cdot x)$):

```
>> syms x
>> float( solve(sin(x)^2+2*cos(x)-0.5,x) )
ans = [ 1.438i  -1.438i  -1.7975  1.7975 ]
```

3. Wenn die Variable zweimal vorkommt, wird nur *ein* Fall weiter verfolgt, nämlich der, wenn die Variable `x` frei und zusätzlich als Wurzel auftritt. Dann wird der Wurzelausdruck isoliert, die Gleichung quadriert, und wie oben gelöst. In diesem Fall entstehen zusätzliche falsche Lösungen, die später manuell aussortiert werden müssen.

```
>> syms x
>> y=x^2+3*x-17*sqrt(3*x^2+12);
>> solve(y,x)
ans = [ -32.501  26.528
    -1.3931E-2-2.0055i  -1.3931E-2+2.0055i ]
```

4. In allen anderen Fälle gibt Jasymca auf.

2.13.3 Nichtlineare Gleichungssysteme

Mit `algsys([Ausdrücke],[Symbolische Variable])` können gekoppelte Gleichungssysteme gelöst werden. Zunächst werden lineare Gleichungen nach dem Gauss-System mit Pivot aufgelöst, und dann die restlichen Gleichungen eine nach der anderen wie `solve()` oben aufgelöst, und die Lösung in den restlichen Gleichungen eliminiert. Dabei wird (außer dem Pivot) keinerlei Intelligenz auf die Strategie des Einsetzens verwendet, so daß dieser Algorithmus nur für sehr einfache Systeme funktioniert. Die Lösung wird als Vektor von Lösungsvektoren angegeben, jede Variable in der Form eines Linearfaktors. Im ersten Beispiel unten gibt es eine Lösung mit `xs=-2/3, a2=3/4, a0=2, a1=0`, im zweiten Beispiel zwei Lösungen.

```
>> syms xs,a0,a1,a2
>> algsys([2-a0,a1-0,a2*xs^2+a1*xs+a0-3-xs,
>                2*a2*xs+a1+1],[a2,a1,a0,xs])
ans = [ [ xs+2/3  a2-3/4  a0-2  a1 ] ]
>> syms a,xs
```

```
>> algsys([a*xs+3*a-(3-xs^2),a+2*xs],[a,xs])
ans = [ [ -sqrt(6)+(xs+3)  2*sqrt(6)+(a-6) ]
      [ sqrt(6)+(xs+3)  -2*sqrt(6)+(a-6) ] ]
>> float(ans)
ans = [[ xs+0.55051  a-1.101 ] [ xs+5.4495  a-10.899 ]]
```

2.14 Infinitesimalrechnung

2.14.1 Differentialquotient

`diff(Funktion,x)` differenziert den Ausdruck `Funktion` nach der symbolischen Variablen `x`. Wenn `x` nicht angegeben wird, wird die Haupvariable von `Funktion` verwendet. Selbst programmierte Funktionen können ebenfalls differenziert werden.

```
>> syms a,x
>> diff(a*x^3)
ans = 3*a*x^2
>> diff(a*x^3,a)
ans = x^3
>> diff(3*sqrt(exp(x)+2),x)
ans = 1.5*exp(x)/sqrt(exp(x)+2)
>> diff(sin(x))      % Ohne Angabe einer Variable
ans = 1              % wird nach z=sin(x) differenziert!
>> diff(sin(x),x)    % So wird es richtig.
ans = cos(x)
>> function y=ttwo(x) y=2*x; end
>> diff(ttwo(sin(x)),x)
ans = 2*cos(x)
```

2.14.2 Taylorpolynom

`taylor(Funktion, x, x0, n)` berechnet das Taylorpolynom in der symbolischen Variablen `x` zum Grad `n` im Arbeitspunkt `x0`.

```
>> syms x
>> taylor(log(x),x,1,1)
ans = x-1
>> rat( taylor(exp(x),x,0,6))
```

```
ans = 1/720*x^6+1/120*x^5+1/24*x^4+1/6*x^3+1/2*x^2+x+1
>> float( taylor(x^3*sin(2*x+pi/4),x,pi/8,2))
ans = 1.057*x^2-0.36751*x+4.1881E-2
```

2.14.3 Unbestimmtes Integral

`integrate(Funktion, x)` integriert den Ausdruck `Funktion` in der symbolischen Variablen `x`. Intern verwendet Jasymca dabei folgende Strategie:

1. Die vordefinierten Funktionen sind bekannt, und Polynome trivial zu berechnen:

   ```
   >> syms x
   >> integrate(x^2+x-3,x)
   ans = 0.33333*x^3+0.5*x^2-3*x
   >> integrate(sin(x),x)
   ans = -cos(x)
   ```

2. Wenn `Funktion` ein gebrochen rationaler Ausdruck ist, d.h. Quotient zweier Polynome, deren Koeffizienten unabhängig von `x` sind, wählen wir den Standardansatz: Aufspalten in Polynom, dann quadratfreien Teil nach Horowitz [11], und schließlich Integration der Partialbrüche. Diese werden mittels `trigrat` in reelle Ausdrücke verwandelt.

   ```
   >> syms x
   >> y=(x^3+2*x^2-x+1)/((x+i)*(x-i)*(x+3))
   y = (x^3+2*x^2-x+1)/(x^3+3*x^2+x+3)
   >> integrate(y,x)
   ans = -1/4*log(x^2+1)+(-1/2*log(x+3)+(-1/2*atan(x)+x))
   >> diff(ans,x)           % Probe
   ans = (x^3+2*x^2-x+1)/(x^3+3*x^2+x+3)
   ```

3. Ausdrücke vom Typ $g(f(x)) \cdot f'(x)$ und $\frac{f'(x)}{f(x)}$ werden detektiert:

   ```
   >> syms x
   >> integrate(x*exp(-2*x^2),x)
   ans = -0.25*exp(-2*x^2)
   >> integrate(exp(x)/(3+exp(x)),x)
   ans = log(exp(x)+3)
   ```

4. Substitutionen vom Typ $(a \cdot x + b)$ werden immer gefunden:

```
>> syms x
>> integrate(3*sin(2*x-4),x)
ans = -1.5*cos(2*x-4)
```

5. Produkte $Polynom(x) \cdot f(x)$ werden durch partielle Integration berechnet. Dies löst alle Falle, bei denen f eine der Funktionen *exp*, *sin* , *cos* , *log* , *atan* ist.

```
>> syms x
>> integrate(x^3*exp(-2*x),x)
ans = (-0.5*x^3-0.75*x^2-0.75*x-0.375)*exp(-2*x)
>> integrate(x^2*log(x),x)
ans = 0.33333*x^3*log(x)-0.11111*x^3
```

6. Alle trigonometrischen und Exponentialfunktionen werden in normalisierte Exponentialfunktionen umgewandelt. Solange die Argumente in den Funktionen erster Ordnung in der Variablen sind, löst das prinzipiell alle Fälle, bei denen die `Funktion` eine rationale Funktion trigonometrischer und exponentieller Funktionen ist. Praktisch funktionieren die meisten Polynome, gebrochen rationale Funktionen führen jedoch oft zu Ausdrücken, bei denen Jasymca mit der abschließenden Zusammenfassung überfordert ist.

```
>> syms x
>> integrate(sin(x)*cos(3*x)^2,x)
ans = -3.5714E-2*cos(7*x)+(5.0E-2*cos(5*x)-0.5*cos(x))
>> integrate(1/(sin(3*x)+1),x)
ans = -2/3*cos(3/2*x)/(sin(3/2*x)+cos(3/2*x))
>> integrate(1/(1+exp(2*x)),x)
ans = -1/2*log(exp(2*x)+1)+x
```

7. Der spezielle und häufige Fall $\sqrt{ax^2+bx+c}$ ist implementiert:

```
>> syms x
>> integrate(sqrt(x^2-1),x)
ans = 0.5*x*sqrt(x^2-1)-0.5*log(2*sqrt(x^2-1)+2*x)
```

8. Natürlich kann man auch clevere Substitutionen selbst mittels `subst` einführen. Wenn alles versagt, integriere man numerisch mittels `quad` oder `romberg` (s.u.).

Für die symbolische Variable gilt das im letzten Kapitel gesagte: Sie kann weggelassen werden, wenn es die freie Hauptvariable der `Funktion` ist. Im Zweifel läßt sich jede Integration schnell mittels Differentiation prüfen.

2.14.4 Numerische Integration

Zur numerischen Integration stehen zwei Routinen zur Verfügung, die jedoch beide nicht besonders leistungsfähig sind. `quad('Ausdruck',ll,ul)` ist der Octave/Matlab Integrationsroutine nachgebildet, allerdings sehr vereinfacht. Es werden nach dem Simpson-Verfahren 200 Stützstellen im festen Abstand verwendet. Diese Funktion nutzt nicht die symbolischen Methoden; der Ausdruck muß als String (also in Anführungszeichen) mit dem Variablennamen `x` eingegeben werden, und zwar so, daß `x` als Vektor verwendet werden kann. Im Romberg-Verfahren wird eine Variable spezifiziert, die zu integrierende Funktion wird als symbolischer Ausdruck eingegeben. Die maximale Zahl der Iterationen ist durch die Konstante `rombergit` (Voreinstellung: 11) und die relative Toleranz `rombergtol` (Voreinstellung: 10^{-4}) festgelegt.

```
>> quad('exp(-x.^2)',0,5)
s = 0.88623
>> syms x
>> romberg(exp(-x^2),x,0,5)
ans = 0.88623
```

2.14.5 Differentialgleichungen

`ode(Ausdruck,y,x)` löst die lineare Differentialgleichung erster Ordnung $y' = f(x) \cdot y + g(x)$. Dabei ist `Ausdruck` die komplette rechte Seite der Differentialgleichung, `x` und `y` sind symbolische Variable. Freie Konstanten werden als `C` angezeigt.

```
>> ode(x,y,x)
ans = 0.5*x^2+C
>> syms k
>> ode(-k*y,y,x)
```

```
ans = C*exp(-k*x)
>> ode(y*tan(x)+cos(x),y,x)
ans = (0.5*cos(x)*sin(x)+(0.5*x+C))/cos(x)
```

3 Maxima-Modus

Zwischen Maxima- und Octave-Modus kann in einer laufenden Sitzung mit der Menüoption *Run - Octave Mode* und *Run - Maxima Mode* umgeschaltet werden. Die Variablen bleiben erhalten, können aber bei Bedarf mit der Option *Run - Clear Environment* gelöscht werden. Aufstarten erfolgt normalerweise im Octave-Modus, es kann aber auch im Maxima-Modus gestartet werden, siehe dazu das Kapitel 5.

Die beiden Benutzeroberflächen unterscheiden sich nicht nur in einigen Bezeichnungen für Operatoren und Funktionen (siehe Kapitel 4). Im Maxima-Modus müssen symbolische Variablen nicht extra mit `syms` deklariert werden, sondern jede auf einer rechten Seite einer Gleichung auftretende unbekannte Zeichenkette wird automatisch als symbolische Variable eingestuft. Zusätzlich werden alle Rückgabewerte fortlaufend in Variablen mit Bezeichnung `d1,d2,d3,..` gespeichert, und bleiben immer abrufbar. Dadurch wird dieser Modus besonders geeignet für Arbeiten mit symbolischen Größen, da man selten Variablen anlegen muß. Für die Bearbeitung großer Zahlenmengen ist das natürlich weniger geeignet, da man schnell den Speicher füllt: jedes Zwischenergebnis bleibt für die Dauer der Sitzung im Speicher.

Exemplarisch einige Beispiele für das Arbeiten im Maxima-Modus. Zu beachten ist, daß hier jeder Befehl mit Semikolon abzuschließen ist. Es können also Befehle über mehrere Zeilen eingegeben werden.

```
c1) [2,3,4]+[4,5,6];     % Semikolon nicht vergessen
    d1 = [ 6  8  10 ]    % Ausgabespeicher d1
(c2) d1[2]*d1[3];        % Index in eckigen Klammern
    d2 = 80
(c3) x^2+3 = y*x-2;      % Automatische Symbole
    d3 = -x*y+(x^2+5)    % Gleichungen sind Differenzen
(c4) choose(n,k):=n!/((n-k)!*k!);  %Definition einer Funktion
choose
(c5) sum(1/k^2,k,1,1000);% 2.te Variante des Summenbefehls
    d4 = 1.6439
```

```
(c6) allroots( (x-1)^3*(x-2)^2*(x-3)*(x-4) );
    d5 = [ 4  3  2  2  1  1  1 ]
(c7) allroots(x^2+1);
    d6 = [ 1*i  -1*i ]
c8) a:x+2-y;           % Zuweisungen mit Doppelpunkt ':'
    a = -y+(x+2)
```

Das Arbeiten mit m-Dateien wird dagegen nicht unterstützt, daher sind alle mit diesem Mechanismus programmierten Funktionen im Maxima-Modus nicht verfügbar. Das Arbeiten mit Vektoren und Matrizen ist ebenfalls weniger komfortabel: Es sind zwar die internen Funktionen (Determinante, Inverse, LAPACK-Funktionen usw) verfügbar, aber nicht die komfortablen Operatoren zur Indexierung, Extraktion und zum Einfügen („Magic Colon“).

Das Arbeiten mit Dateien erfolgt über den `Loadfile` und `Save`-Befehl.

4 Referenz

4.1 Allgemeines

Eine Texteingabe wird durch Zeilenende (Oktave-Modus) bzw Semikolon und Zeilenende (Maxima-Modus) abgeschlossen. Eine Eingabe kann aus mehreren Anweisungen bestehen, die durch Komma oder Semikolon getrennt sind. Im Octave-Modus wird dadurch zusätzlich gesteuert, ob das Ergebnis einer Operation in der Konsole ausgedruckt wird (nicht mit Semikolon). Kommentare lassen sich in beiden Moden durch % oder # kennzeichnen, die entsprechende Zeile wird von diesem Buchstaben an ignoriert. Bei Variablen wird zwischen Groß-und Kleinschreibung unterschieden, nicht jedoch bei den Namen der vorprogrammierten Funktionen. Lange Berechnungen oder Endlosschleifen in eigenen Programmen lassen sich unterbrechen mit der Menuoption *Run - Interrupt* oder durch Eingabe von `Strg-c`. Das funktioniert jedoch nicht bei eingebauten Funktionen.

In den folgenden Tabellen werden Funktionsargumente durch unterschiedliche Schrifttypen und Bezeichnungen entsprechend dieser Tabelle näher charakterisiert.

Beispiel	Schrift	Beschreibung
`long`	Schreibmaschine	konstanter Text, der wörtlich so eingegeben werden muß.
Name	schräg	Text (String), immer in Anführungszeichen (einfache ' oder zweifache ")
Var	kursiv	Algebraische Variable. Beliebiger Ausdruck mit oder ohne Symbolen, Skalar, Vektor oder Matrix.
Vektor	kursiv	Vektor, meist auch symbolisch möglich (außer Lapack)
Matrix	kursiv	Matrix, meist auch symbolisch möglich (außer Lapack)
Sym	kursiv	symbolische Variable
ANZAHL	Kapitälchen	Ganzzahl, oft > 0 erforderlich.

4.2 Kommandos

Kommandos werden für Einstellungen verwendet. Anders als bei Funktionen können hier die Optionen ohne Klammern gesetzt werden. Nur ein Kommando kann in ei-

ner Texteingabe erfolgen.

Name	Option	Funktion	Mod	Ref

format	`short`
	`long`
	BASIS STELLEN
		Anzeigeformat für Gleitkommazahlen: 5 Stellen, alle Stellen oder zur BASIS mit STELLEN	M,O	

syms	Sym_1 [,Sym_2,...]	Deklariert Sym_1,... als symbolische Variable	O	
clear	Var_1 [,Var_2,...]	Löscht die Variablen Var_1,...	M,O	
who		Anzeige aller Variablen	M,O	
path		Anzeige des Suchpfades	M,O	
addpath	*Pfad*	Erweitern des Suchpfades	M,O	
hold		Graphik fixieren/lösen	M,O	
hold	`on`	Graphik fixieren	M,O	
hold	`off`	Graphik lösen	M,O	

4.3 Operatoren

4.3.1 Definieren

Größe	Bsp. Octave	Bsp. Maxima
Zahl	`-3.214e-12`	`-3.214e-12`
Vektor	`[1 2 3 4]`	`[1,2,3,4]`
Matrix	`[1 2 ; 3 4]`	`matrix([1,2],[3,4])`
Vektorelement	`x(2)`	`x[2]`
Matrixelement	`x(2,1)`	`x[2,1]`
Zahlenbereich	`2:5`	
Variable	`x = 2.321`	`x : 2.321;`

4.3.2 Rechnen

Die folgende Tabelle gibt die Operatoren mit je einem Beispiel im Octave-Modus (O-Mode) und Maxima-Modus (M-Mode) an. Operationen werden entsprechend ihres *Rangs* abgearbeitet, wenn dieser gleich ist in der *Reihenfolge*.

Funktion	O-Mode	M-Mode	Rang	Reihenfolge
Addition	`x + y`	`x + y`	4	links-rechts
Subtraktion	`x - y`	`x - y`	4	links-rechts
Skalare Multiplikation	`x .* y`	`x * y`	3	links-rechts
Vektor/Matrix Multiplikation	`x * y`	`x . y`	3	links-rechts
Skalare Division	`x ./ y`	`x / y`	3	links-rechts
Matrix Division (rechts)	`x / y`		3	links-rechts
Matrix Division (links)	`x \ y`		3	links-rechts
Skalare Exponentiation	`x .^ y`	`x ^ y`	1	links-rechts
Vektor/Matrix Exponentiation	`x ^ y`	`x 'y`	1	links-rechts
Zahlenbereich	`x:y:z`		5	links-rechts
Zuweisung	`x = z`	`x : y`	10	rechts-links
Zuweisung	`x += z`		10	rechts-links
Zuweisung	`x -= z`		10	rechts-links
Zuweisung	`x /= z`		10	rechts-links
Zuweisung	`x *= z`		10	rechts-links
Preincrement	`++x`	`++x`	10	links-rechts
Predecrement	`--x`	`--x`	10	links-rechts
Postincrement	`x++`	`x++`	10	rechts-links
Postdecrement	`x--`	`x--`	10	rechts-links
Adjunkte, konjugiert komplex	`x'`		1	rechts-links
Fakultät		`x!`	1	links-rechts

4.3.3 Vergleichen, Logik

Funktion	O-Mode	M-Mode	Rang	Reihenfolge
Kleiner	`x < y`	`x < y`	6	links-rechts
Kleiner oder gleich	`x <= y`	`x <= y`	6	links-rechts
Größer	`x > y`	`x > y`	6	links-rechts
Größer oder gleich	`x >= y`	`x >= y`	6	links-rechts
Gleich	`x == y`	`x == y`	6	links-rechts
Nicht gleich	`x ~= y`	`x ~= y`	6	links-rechts
Und	`x & y`	`x & y`	7	links-rechts
Oder	`x | y`	`x | y`	9	links-rechts
Nicht	`~x`	`~x`	8	links-rechts

4.4 Programmierung

In beiden Moden sind die gleichen Konstrukte verfügbar, wobei jedoch die Grammatik etwas verschieden ist:

Oktave Modus

Verzweigungen	`if x==0 xp=1; end`
	`if x==0 xp=1; else xp=2; end`
Schleifen	`while x<17 x=x+1; end`
	`for i=0:100 x=x+i; end`
Sprungbefehle	`return, continue, break`
Funktion	`function y=ttwo(x) y=2*x; end`
Führe aus	`eval('x=24')`
Textausgabe	`printf('Ergebnis=%f', x)`
Fehlermeldung	`error('Fehler')`

Maxima Modus

Verzweigungen	`if x==0 then ( xp:1 );`
	`if x==0 then ( xp:1 ) else ( xp:2 );`
Schleifen	`while x<17 do ( x:x+1 );`
	`for i:0 step 1 thru 100 do ( x:x+i );`
Sprungbefehle	`return, continue, break`
Funktion	`ttwo(x) := 2*x;`
Block	`block( [x,y], x:1, y:1, z:z+x+y );`
Textausgabe	`printf('Ergebnis=%f', x);`
Fehlermeldung	`error('Fehler');`

4.5 Funktionen

4.5.1 Skalare

Name(Argumente)	Funktion	Mod	Ref
float(*Var*)	*Var* als Gleitkommazahl	M,O	
rat(*Var*)	*Var* als exakte Zahl	M,O	
realpart(*Var*)	Realteil von *Var*	M,O	
imagpart(*Var*)	Imaginärteil von *Var*	M,O	
abs(*Var*)	Absolutbetrag von *Var*	M,O	
sign(*Var*)	Vorzeichen von *Var*	M,O	
conj(*Var*)	*Var* konjugiert komplex	M,O	
angle(*Var*)	Hauptwert des Arguments von *var*	M,O	
cfs(*Var* [,Var_T])	Kettenbruchentwicklung von *Var* mit Genauigkeit Var_T	M,O	
primes(VAR)	Primzahlzerlegung von VAR	M,O	

4.5.2 Skalare Funktionen

Name(Argumente)	Funktion	Mod	Ref
sqrt(*Var*)	Quadratwurzel	M,O	
exp(*Var*)	Exponentialfunktion	M,O	
log(*Var*)	Natürlicher Logarithmus	M,O	
sinh(*Var*)	Hperbolischer Sinus	O	
cosh(*Var*)	Hperbolischer Cosinus	O	
asinh(*Var*)	Hyperbolischer Areasinus	O	
acosh(*Var*)	Hperbolischer Areacosinus	O	
sech(*Var*)	Hperbolischer Sekans	O	
csch(*Var*)	Hperbolischer Cosekans	O	
asech(*Var*)	Hperbolischer Areasekans	O	
acsch(*Var*)	Hperbolischer Areacosekans	O	
sin(*Var*)	Sinus (Radian)	M,O	
cos(*Var*)	Cosinus (Radian)	M,O	
tan(*Var*)	Tangens (Radian)	M,O	
asin(*Var*)	Arcussinus (Radian)	M,O	
acos(*Var*)	Arcuscosinus (Radian)	M,O	
atan(*Var*)	Arcustangens (Radian)	M,O	
atan2(Var_1, Var_2)	Arcustangens (Radian)	M,O	
sec(*Var*)	Sekans (Radian)	O	
csc(*Var*)	Cosekans (Radian)	O	
asec(*Var*)	Arcussekans (Radian)	O	
acsc(*Var*)	Arcuscosekans (Radian)	O	
factorial(N)	Fakultät $N!$	M,O	
nchoosek(N,K)	Binomialkoeffizient $\binom{N}{K}$	M,O	
gamma(*Var*)	Gammafunktion	M,O	
gammaln(*Var*)	Logarithmus der Gammafunktion	M,O	

4.5.3 Vektoren und Matrizen

Name(Argumente)	Funktion	Mod	Ref
linspace(*Var*$_1$,*Var*$_2$,ANZAHL)	Vektor mit ANZAHL Zahlen von *Var*$_1$ bis *Var*$_2$	O	
length(*Vektor*)	Anzahl der Elemente des *Vektor*	M,O	
zeros(ZEILEN[,SPALTEN])	Matrix aus Nullen	M,O	
ones(ZEILEN[,SPALTEN])	Matrix aus Einsen	M,O	
eye(ZEILEN[,SPALTEN]])	Matrix mit Einserdiagonale	M,O	
rand(ZEILEN[,SPALTEN])	Matrix aus Zufallszahlen	M,O	
hilb(RANG)	Hilbertmatrix	M,O	
invhilb(RANG)	Inverse Hilbertmatrix	O	
size(*Matrix*)	Anzahl der Reihen und Spalten	M,O	
sum(*Var*)	Wenn *Var* ein Vektor ist: Summe der Elemente, wenn *Var* Matrix ist: Summe der Spalten.	M,O	
find(*Var*)	Index nichtverschwindender Elemente	M,O	
max(*Var*)	Größtes Element in *Var*	M,O	
min(*Var*)	Kleinstes Element in *Var*	M,O	
diag(*Var*,[VERSATZ])	Wenn *Var* ein Vektor ist: Matrix, die *Var* als Diagonale besitzt, wenn *Var* Matrix ist: Diagonale als Vektor.	M,O	
det(*Matrix*)	Determinante	M,O	
eig(*Matrix*)	Eigenwerte	M,O	
inv(*Matrix*)	Inverse	M,O	
pinv(*Matrix*)	Pseudoinverse	M,O	
lu(*Matrix*)	LU-Zerlegung	M,O	
svd(*Matrix*)	Singulärwertzerlegung (Lapack)	M,O	
qr(*Matrix*)	QR-Zerlegung (Lapack)	M,O	
eigen(*Matrix*)	Eigenwerte (Lapack)	M,O	

4.5.4 Polynome

Name(Argumente)	Funktion	Mod	Ref
poly(*Vektor*)	Koeffizienten des Polynoms mit Nullstellen *Vektor*	O	
polyval(*Vektor, Var*)	Funktionswert des Polynoms mit Koeffizienten *Vektor* im Punkt *Var*	O	
polyfit($Vektor_x$,$Vektor_y$,GRAD)	Fitted Polynom mit GRAD an die Punkte mit Koordinaten $Vektor_x$ und $Vektor_y$	O	
roots(*Vektor*)	Nullstellen des Polynoms mit Koeffizienten *Vektor*	M,O	
coeff(*Var*,*Sym*,POTENZ)	Koeffizient von *Sym* zur POTENZ in *Var*	M,O	
divide(Var_1, Var_2)	Division $\frac{Var_1}{Var_2}$ mit Rest	M,O	
gcd(Var_1, Var_2)	Größter gemeinsamer Teiler	M,O	
sqfr(*Var*)	Quadratfreie Zerlegung	M,O	
allroots(*Var*)	Nullstellen	M,O	

4.5.5 Gleichungen und Ausdrücke

Name(Argumente)	Funktion	Mod	Ref
subst(Var_x,Var_y,Var_z)	Ersetze Var_x für Var_y in Var_z	M,O	
trigrat(*Var*)	Trigonometrische und andere Vereinfachungen	M,O	
trigexp(*Var*)	Trigonometrische Erweiterung	M,O	
solve(*Var, Sym*)	Löst $Var = 0$ nach *Sym* auf.	M,O	
algsys(*[*Var_1, $Var_2, \ldots$*]*,*[*Sym_1, $Sym_2, \ldots$*]*)	Löst das Gleichungssystem $Var_1 = 0, Var_2 = 0, \ldots$ nach $Sym_1, Sym_2, \ldots$ auf.	M,O	
linsolve(*Matrix, Vektor*)	Löst $Matrix \cdot x = Vektor$ nach x	M,O	
linsolve2(*Matrix, Vektor*)	Löst $Matrix \cdot x = Vektor$ nach x (Lapack)	M,O	
linlstsq(*Matrix, Vektor*)	Löst $Matrix \cdot x = Vektor$ nach x, überbestimmt (Lapack)	M,O	

4.5.6 Infinitesimalrechnung

Name(Argumente)	Funktion	Mod	Ref
sum(*Var*,*Sym*, ANFANGSWERT, ENDWERT)	$\sum_{Sym=Anfangswert}^{Sym=Endwert} Var$	M,O	
lsum(*Var*,*Sym*,*Vektor*)	$\sum_{Sym \in Vektor} Var$	M,O	
diff(*Var*[,*Sym*])	$\frac{dVar}{dSym}$	M,O	
integrate(*Var*[,*Sym*])	$\int Var(Sym) dSym$	M,O	
romberg(*Var*$_f$,*Sym*, *Var*$_a$, *Var*$_b$)	$\int_{Var_a}^{Var_b} Var_f dSym$	M,O	
quad(*Ausdruck*, *Var*$_a$, *Var*$_b$)	$\int_{Var_a}^{Var_b} Ausdruck dx$	O	
taylor(*Var*$_f$, *Sym*, *Var*$_p$, GRAD)	Taylorpolynom der Funktion *Var*$_f$ zur Variablen *Sym* im Arbeitspunkt *Var*$_p$ der Ordnung GRAD	M,O	
ode(*Var*, *Sym*$_y$, *Sym*$_x$)	Löst die lineare Differentialgleichung $y' = f(x) \cdot y + g(x)$. Dabei ist *Var* die rechte Seite der Gleichung.	M,O	
fzero(*Ausdruck*,*Var*$_a$, *Var*$_b$)	Findet Nullstelle zwischen *Var*$_a$ und *Var*$_b$	O	

4.5.7 Plotten

Name(Argumente)	Funktion	Mod	Ref
plot(*x*,*y* [,*Option*])	Plotted *x* gegen *y* mit *Option* *x* und *y* sind Vektoren gleicher Länge. *Option* ist eine Zeichenkette (String), in der die Farbe (eine von `r,g,b,y,m,c,w,k`) und das Symbol (eines von `+,*,o,x`) gesetzt werden. Voreinstellung: blau, Linien.	M,O	
loglog(*x*,*y*[,*Option*])	Doppeltlogarithmischer Plot	M,O	
linlog(*x*,*y*[,*Option*])	Einfach logarithmischer Plot	M,O	
loglin(*x*,*y*[,*Option*])	Einfach logarithmischer Plot	M,O	
print(*Name*)	Schreibt Grafik als eps-Datei	M,O	

5 Installation

Applet

Das Applet kann in jedem Java-fähigen (Version ≥ 1.5) Internetbrowser direkt von der Homepage [8] gestartet werden; es ist keine Installation erforderlich. Wenn Sie eine lokale Kopie anlegen wollen, kopieren Sie das Verzeichnis `Applet`) aus der Distribution auf Ihren Computer, und öffnen Sie dann `index.html` mit Ihrem Internetbrowser. Alternativ kann man die Datei `index.html` doppelklicken. Dabei erhalten Sie einen lokalen Zugang zur Online-Referenz, die über den Internet-Browser navigiert werden kann. Die übrigen Links auf der Seite `index.html` sind dann allerdings leer.

Applikation

Die Applikation erlaubt den Zugriff auf LAPACK sowie das Dateisystem. Kopieren Sie das Verzeichnis
`Applikation` auf Ihren Computer. Es enthält alle Dateien, die zum Aufstarten des Programms erforderlich sind. Natürlich muß eine aktuelle Version (≥ 1.5) der Java-Runtime [12] installiert sein. Neben Festplatte kann Jasymca auch auf Wechselmedien (USB-Stick, Speicherkarte) oder auf nur-lesbaren Medien (CD-Rom) installiert werden.

Zum Aufstarten des Programms reicht es auf den meisten Computern, den Programmicon `jasymca.jar` in `Applikation` anzuklicken. Wenn das nicht geht, muß ein Kommandofenster geöffnet werden, und im Verzeichnis `Applikation` folgender Befehl eingegeben werden: `java -jar jasymca.jar`. Soll Jasymca statt im Octave- im Maxima-Modus aufstarten, so ist dies mit
`java -jar jasymca.jar ui=Maxima` möglich.

Beim Aufstarten sucht Jasymca eine Datei mit Namen `Jasymca.Octave.rc` bzw. `Jasymca.Maxima.rc` je nach Modus. Wenn die Umgebungsvariable `JASYMCA_RC` gesetzt ist, sucht Jasymca stattdessen unter diesem Namen, erweiter um `.Octave.rc` oder `.Maxima.rc` . Man kann diese Datei als Skriptdatei anlegen, um oft benutzte Definitionen zu laden.

Midlet

Die Midlet-Version ist i.W. kompatibel mit dem Applet inklusive der Möglichkeit zur graphischen Darstellung von Funktionen. Die Benutzeroberfläche entspricht

dabei dem von FnattLavME [2], und kann im Handbuch zu diesem Programm nachgelesen werden.

Die Installation auf mobile Geräte unterscheidet sich stark zwischen den einzelnen Systemen. Das Mobiltelefon oder der PDA muß folgende Java-Installation vorweisen: CLDC 1.1, MIDP 2.0, JSR-75. Für die meisten Systeme ist dies vom Hersteller verfügbar, oder bereits vorinstalliert. Das Midlet besteht aus den Dateien `Jasymca.jar` und `Jasymca.jad`, beide im Verzeichnis `Midlet`. Es kann über eine Internetverbindung, oder eine lokale Verbindung (USB/Bluetooth) von Ihrem PC aus installiert werden. Beachten Sie dazu die Anleitungen Ihres Mobilgerätes. Detaillierte Anleitungen für eine Installation auf Palm Tungsten E und Nokia 6230i finden Sie hier [3].

6 Lizenz

Jasymca ist ein freies und quelloffenes Programm. Sie könen es verteilen und modifizieren unter der Bedingung der GNU General Public License. Der Lizenztext ist separat in der Distribution enthalten.

Dieses Dokument ist urheberrechtlich geschützt. Es darf nur für private Zwecke kopiert werden.

DasProgramm Jasymca verwendet Module und Teile anderer Programme, die mit ihren Lizensen hier aufgelistet sind. Den vollen Lizenztext enthalten die ebenfalls enthaltenen Quelltexte des Programms.

- Die Dateien BigInteger.java, Random.java, and MPN.java sind leicht modifizierte Versionen des GNU-Classpath [14] Projektes.

  ```
  Copyright (C) 1998, 1999, 2000, 2001, 2002, 2003,
  2005  Free Software Foundation, Inc.

  GNU Classpath is free software; you can
  redistribute it and/or modify it under the terms
  of the GNU General Public License as published by
  the Free Software Foundation; either version 2, or
  (at your option) any later version.
  ```

- Die Dateien JMath.java und SFun.java sind aus freien Quellen zusammengestellt von Visual Numerics Inc [15].

Copyright (c) 1999 Visual Numerics Inc. All Rights Reserved.

Permission to use, copy, modify, and distribute this software is freely granted by Visual Numerics, Inc., provided that the copyright notice above and the following warranty disclaimer are preserved in human readable form.

- EPS Graphics Library:

Copyright (c) 2006-2007, Thomas Abeel Project: http://sourceforge.net/projects/epsgraphics/ The EPS Graphics Library is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

- Pzeros.java:

derived from pzeros.f <http://netlib.org/numeralgo/na10>

All the software contained in this library is protected by copyright. Permission to use, copy, modify, and distribute this software for purpose without fee is hereby granted, provided that this entire notice is included in all copies of any software which is or includes a copy or modification of this software and in all copies of the supporting documentation for such software.

THIS SOFTWARE IS BEING PROVIDED "AS IS", WITHOUT ANY EXPRESS OR IMPLIED WARRANTY. IN NO EVENT, NEITHER THE AUTHORS, NOR THE PUBLISHER, NOR ANY MEMBER OF THE EDITORIAL BOARD OF THE JOURNAL

"NUMERICAL ALGORITHMS", NOR ITS EDITOR-IN-CHIEF, BE LIABLE FOR ANY ERROR IN THE SOFTWARE, ANY MISUSE OF IT OR ANY DAMAGE ARISING OUT OF ITS USE. THE ENTIRE RISK OF USING THE SOFTWARE LIES WITH THE PARTY DOING SO.

ANY USE OF THE SOFTWARE CONSTITUTES ACCEPTANCE OF THE TERMS OF THE ABOVE STATEMENT.

AUTHOR:

DARIO ANDREA BINI
UNIVERSITY OF PISA, ITALY
E-MAIL: bini@dm.unipi.it

REFERENCE:

- NUMERICAL COMPUTATION OF POLYNOMIAL ZEROS BY MEANS OF ABERTH'S METHOD NUMERICAL ALGORITHMS, 13 (1996), PP. 179-200

SOFTWARE REVISION DATE:

JUNE, 1996

- BLAS, LAPACK und JLAPACK:

Copyright (c) 1992-2008 The University of Tennessee. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions

Literatur

[1] www.hs-furtwangen.de/ dersch

[2] www.hs-furtwangen.de/ dersch/FnattLabME/FnattLabME2.pdf

[3] www.hs-furtwangen.de/ dersch/Jasymca/Jasymca.pdf

[4] www.octave.org/

[5] www.mathworks.com

[6] www.scilab.org/

[7] http://maxima.sourceforge.net/

[8] www.hs-furtwangen.de/ dersch/jasymca2

[9] http://www.netlib.org/java/f2j/

[10] www.netlib.org/lapack/

[11] mathworld.wolfram.com/HorowitzReduction.html

[12] http://java.sun.com

[13] http://java.sun.com/j2me/

[14] www.gnu.org/software/classpath/

[15] www.vni.com/