Jasymca 2.0 - Symbolic Calculator for Java

Helmut Dersch

March 15, 2009

Abstract

Jasymca is an interactive System for solving math problems. It supports arbitrary precision numbers and symbolic variables. Scalars, vectors, and matrices can be built from all datatypes and used in calculations. From the pseudoinverse of symbolic matrices over trigonometric simplifications to symbolic solutions of integrals and systems of equations, the main functionalities of CAS-programs are provided. Additionaly, high performance numerical routines from LAPACK and a plotmodule are implemented. The user interface can be selected from either an Octave/Matlab/SciLab-like language or a GNU-Maxima style. Three versions of Jasymca are provided which cover almost any computer platform: A Midlet version for portable devices like cellphones or PDAs, a java application for desktop PCs, laptops and workstation, and an applet which can be integrated in webpages. Jasymca is free software covered by the GNU public license.

Contents

1 Introduction

Jasymca has been developed for teaching mathematics, especially to facilitate a fast and easy entrance to computer mathematics. One of the main obstacles are pocket calculators, which prevent many students from using computers for math. Pocket calculaters are cheap and portable, while CAS-programs are often expensive and always require at least a laptop to run. Jasymca is free software and runs on almost any system equipped with a microprocessor: from mobile phones and pdas to windows/linux/macos computers, even on game consoles or internet routers.

Jasymca 2.0 is based on Jasymca 1.01 [1] with significant extensions and improvements. Apart from the new grammar (Jasymca 1.01 was Maxima [7] - oriented), matrix and plotfunktionen were added, as well as the parser and compiler completely rewritten. The user interface defaults to a style reminiscent of Octave [4], Matlab [5] and SciLab [6], without copying each detail. Users of either of these programs should have no problems using Jasymca, and, for the sake of teaching, vice versa. The main extension to these programs is the seamless integration of symbolic calculations, which do not require special commands.

The user interface can be switched to GNU-Maxima-style, which is more conveniant for some problems, see chapter 3 for details.

Chapter 2 of this document is a tutorial with examples and exercises, partially taken from our introductory Computer-Mathcourse for engineering students. Most examples can be solved by the applet at the Jasymca-homepage [8], and do not require any installation. The next chapter briefly explains the alternative Maxima-mode. An overview and reference to all commands, functions and options follows (chapter 4), and the last chapter (5) deals with technicalities of installing Jasymca on computers and mobile devices.

2 Working with Jasymca

Your computer must be equipped with a recent Java-Version ($\geq 1,5$). No further installation is required: Just visit the Jasymca-homepage [8] to start the program. Of course, you can also use a local installation on any suitable system, see chapter 5 for an installation guide.

Jasymca starts up in Octave-mode. Commands are entered using the keyboard in the textinput field in the lower part of the window. The results are displayed and saved in the upper large textarea. The zoom-function in the main menu adjusts font size. The buttons `<` and `>` recall earlier commands (scrolling the command history). The same operation is acieved by typing the arrow keys of the keyboard, and is an importand aid for efficient working.

Typing `demoEN` starts a 5-minutes demonstration of Jasymca's capabilities. In the following examples and exercises the verbatim response of Jasymca is displayed; to repeat the results you have to copy the text after the prompt (`>>`) into the textinputfield and press *enter*.

2.1 Numbers

Numbers are entered in the usual Computer format: with optional decimalpoint and decimal exponent following the letter `e` (or `E`). The numbers 5364 and $-1.723478265342 \cdot 10^{12}$ should be entered like:

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

Most of the time these will be stored as floating point data (double, IEEE standard 754). They are rounded to 5 significant digits for display, but for calculations the full precision of this format is always preserved (15-16 decimal digits). By switching the format (`format long`) all significant places are displayed.

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

As an extension Jasymca offers the command `format Base Number`, which is used to display numbers in a system with arbitrary `Base` with any `Number` of significant digits. To display numbers with 15 digits in the binary system we type:

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

Using `format short` returns the display mode to default (short decimal). It should be emphasized, that none of the format commands influences the *internal* representation and accuracy of floating point numbers.

Numbers, which are entered without decimal point and exponent, and which are larger than 10^{15} are stored as exact rational datatype. These numbers are internally represented as quotient of two variable length integers (java datatype `BigInteger`), which allows you to perform calculations without any rounding errors. In the first case of the following example a floating point number is generated, in the second case an exact rational:

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

Each floating point number `Z` can be converted to an exact number using the command `rat(Z)`. The conversion is accomplished by continued fraction expansion with an accuracy determined by the variable `ratepsilon` (default: 10^{-8}).

```
>> rat(0.333333333333333333)
ans = 1/3
```

Operations between exact and floating point numbers always lead to the promotion of floating point numbers. Calculations can be performed without rounding errors by “rationalizing” just the first number.

```
>> 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
```

Conversely, the command `float(Z)` converts numbers into floating point format. Both commands also work for composite datatypes, like polynomials and matrices, whose coefficients are transformed in one step. Irrational function values of exact numbers and constants like `pi` remain unevaluated until the `float`-command is issued.

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

The exact datatype is useful especially for unstable problems, like solving systems of linear equations with ill-conditioned matrix. The Hilbert-matrix is an extreme example:

```
>> det( hilb(20)*invhilb(20) )
ans = 1           % correct
>> det( float(hilb(20))*float(invhilb(20)) )
ans = 1.6713E151  % slightly wrong
```

Imaginary numbers are marked with an immediately following `i` or `j`. This will work even if the predefined variables `i` and `j` have been overwritten.

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

2.2 Operators and Functions

The basic arithmetic operations are marked with the usual symbols (`+ - * /`) . Exponention is performed with the accent character (ˆ). Multiplication and division precede addition and subtraction; any order of evaluation can be forced by parenthesis.

Exercise 1 (Numbers and Operators)

Calculate the following mathematical expressions:

$$3.23 \cdot \frac{14-2^5}{15-(3^3-2^3)} \qquad 4.5 \cdot 10^{-23} : 0.0000013$$

$$17.4^{(3-2.13^{1.2})^{0.16}} \qquad \frac{17.23 \cdot 10^4}{1.12 - \frac{17.23 \cdot 10^4}{1.12 - \frac{17.23 \cdot 10^4}{1.12}}}$$

Solution:

```
>> 3.23*(14-2ˆ5)/(15-(3ˆ3-2ˆ3))
ans = 14.535
>> 4.5e-23/0.0000013
ans = 3.4615E-17
>> 17.4ˆ((3-2.13ˆ1.2)ˆ0.16)
ans = 13.125
>> 17.23e4/(1.12-17.23e4/(1.12-17.23e4/1.12))
ans = 76919
```

In addition to these arithmetic operators Jasymca provides operators for comparing numbers (`< > >= <= == ˜=`), and for boolean functions (`& | ˜`). Logical *true* is the number 1, *false* is 0.

```
>> 1+eps>1
ans = 1
>> 1+eps/2>1      % defines eps
ans = 0
>> A=1;B=1;C=1;   % semikolon suppresses output.
>> !(A&B)|(B&C) == (C˜=A)
ans = 1
```

The most common implemented functions are the squareroot (`sqrt(x)`), the trigonometric functions (`sin(x), cos(x), tan(x)`) and inverses (`atan(x), atan2(y,x)`), and the hyperbolic functions (`exp(x), log(x)`). A large number of additional functions are available, see the list in chapter 4. Some functions are specific to integers, and also work with arbitrary large numbers: `primes(Z)` expands `Z` into primefactors, `factorial(Z)` calculates the factorial function. Modular division is provided by `divide` and treated later in the context of polynomials.

Example: Functions

```
>> log(sqrt(854))          % natural logarithm
ans = 3.375
>> 0.5*log(854)
ans = 3.375
>> float(sin(pi/2))        % argument in radian
ans = 1
>> gammaln(1234)           % log( gamma( x ) )
ans = 7547
>> primes(1000000000000000001)
ans = [ 101  9901  999999000001 ]
>> factorial(35)
ans = 1.0333E40
>> factorial(rat(35))      % to make it exact.
ans = 10333147966386144929666651337523200000000
```

2.3 Variables

Variables are declared by supplying a name and value in the format `name=value`. The name can be any charactersequence. With the exception of the first character it may also contain numbers. The value is any number or expression.

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

Some variables are predefined (like `pi`). The last previous result of a calculation is stored in the variable `ans`. All variables are displayed by the command `who`. Single variables can be deleted by entering `clear variable`.

It is possible to define variables whose value is a function. In this case the function's name must be preceded by the character `$` to suppress evaluation. These variables can be used like the function they stand for. For example, who dislikes the builtin function `realpart(x)`'s name can shorten it to the Matlab-version:

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

Exercise 2 (Variables)

Convert several temperatures ($0°C, 20°C, 30°C, 50°C$) from Celsius to Fahrenheit-scale. The formula for transformations to Fahrenheit-degrees reads:

$$T/°F = 9/5 \cdot T/°C + 32$$

Use variables for the temperatures.
Solution:

```
>> TC=20
TC = 20
>> TF=9/5*TC+32
TF = 68
>> TC=30
TC = 30
>> TF=9/5*TC+32   %  Repeat with arrow key
TF = 86
```

Exercise 3 (Variables)

Calculate the skin surface of your body from height h und weight W using DuBois' formula:

$$A = h^{0.725}(cm) \cdot W^{0.425}(kg) \cdot 71.84 \cdot 10^{-4}(m^2)$$

Use variables h and W.
Solution:

```
>> h=182; W=71;
>> A=hˆ0.725*Wˆ0.425*71.84e-4
A = 1.9129
```

Exercise 4 (Variables)

Calculate some elements of the recursively defined sequence

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

When does this sequence approach its limit $\sqrt{3}$ to within `2*eps`?
Solution:

```
>> 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 Vectors und Matrices (1)

These datatypes are either used for multidimensional objects, or for simultaneous calculations on large numbers of data, e.g. for statistical problems. In this chapter we discuss this latter aspect. Linear algebra and the usual vector calculations are treated in chapter 2.9.

Vectors are marked with square brackets. The elements are entered as comma-separated list. The commas may be left if the elements can be distiguished in a unique manner, which however fails in the second example below:

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

Colon and the function `linspace` are used to define ranges of numbers as vectors.

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

The number of elements in a vector `x` is calculated with the function `length(x)`, individual elements are extracted by providing the index `k` like `x(k)`. This index `k` must be a number in the range 1 to (including) `length(x)`. The colon operator plays a special role: Used as index, all elements of the vector are returned. Additionally, ranges of numbers can be used as index.

```
>> y(2)                % single element
ans = 0.5
>> y(:)                % magic colon
ans = [ 0  0.5  1  1.5  2 ]
>> y(2:3)              %  index between 2 and 3
ans = [ 0.5  1 ]
>> y(2:length(y))      %  all from index 2
ans = [ 0.5  1  1.5  2 ]
>> y([1,3,4])          %  indices 1,3 and 4
```

```
ans = [ 0  1  1.5 ]
>> y([1,3,4]) = 9      %  insert
ans = [ 9  0.5  9  9  2 ]
>> y([1,3,4]) = [1,2,3] % insert
ans = [ 1  0.5  2  3  2 ]
```

Matrices are handled in a similar way, only with two indices for rownumber (first index) and columnnumber (second index). Rows are separated by either a semicolon or a linefeed during input.

```
>> 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
```

The operators of chapter 2.2 may be applied to vectors and matrices. If scalar, per-element operation is desired, some operators (`* / ^`) must be preceded by a point to distinguish them from the quite different linear-algebra versions of these operations (see chapter 2.9). Further useful functions are `sum(vector)` and `prod(vector)` which return the sum and product of the vectors elements.

Exercise 5 (Vectors)

Working with vectors: Calculate

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

for $n = 10, n = 100, n = 10000$.
Solution:

```
>> n=10; k=1:n;
>> sum(k), sum(k.*k), sum(k.^3)  % point!
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
```

Exercise 6 (Vectors)

Transform the list of Celsiustemperatures $-30°C, -29°C, \ldots, 80°C$ into Fahrenheitdegrees. Use vectors.
Solution:

```
>> 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 Plotting

Data may be graphed using the `plot(x,y)`-function, x and y being equalsized vectors, which denote the coordinates of the datapoints to be plotted. A third optional argument `plot(x,y,option)` specifies plotoptions like colors and symbols, see exercise 7ff. The graphic may then be decorated (axis, title) and exported as encapsulated-postscript file for inclusion in textdocuments. This export option is not available in the applet and midlet versions of Jasymca. With `hold` (or `hold on`) the graphic gets locked so that subsequent plotcommands use the same window. Repeating `hold` (or `hold off`) deletes the graphic.

Logarithmic and semilogarithmic plots are provided with the functions `loglog`, `linlog` and `loglin`.

Exercise 7 (Plotting)

Plot the function $y = \frac{1}{1+2x^2}$ in the range $x = 0.01 \ldots 100$ linear and logarithmic.
Solution:

```
>> 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)
```

Exercise 8 (Plotting)

Display of Lissajous-figures: From the vector `t=0:0.1:4*pi;` create the trigonometric expressions `x=sin(0.5*t+1);` und `y=cos(1.5*t);`. The plot x vs. y is called *Lissajous*-figure. Create different figures by variating the constants `0.5,1,1.5` in the definition.
Partial solution:

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

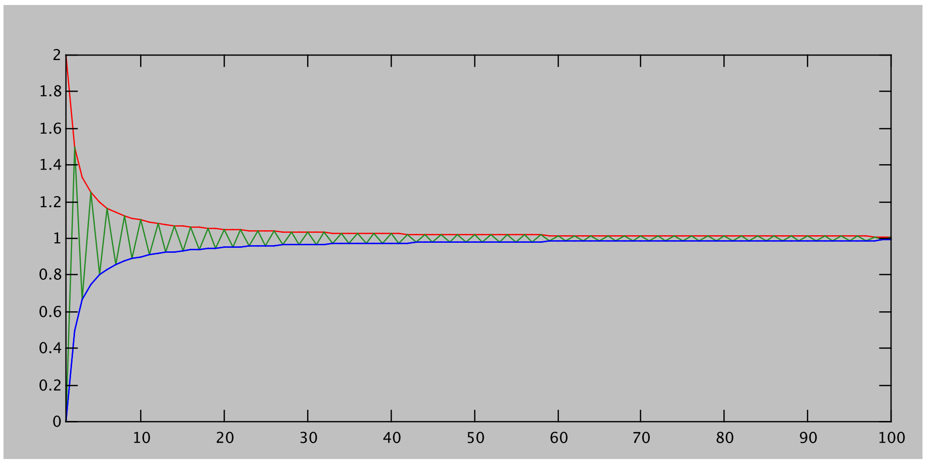
Exercise 9 (Plotting)

Calculate the first 100 elements of the sequences

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

Plot x_n versus n using the command `plot(n, xn)`. Variate the plotoptions (colors, symbols).
Solution:

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



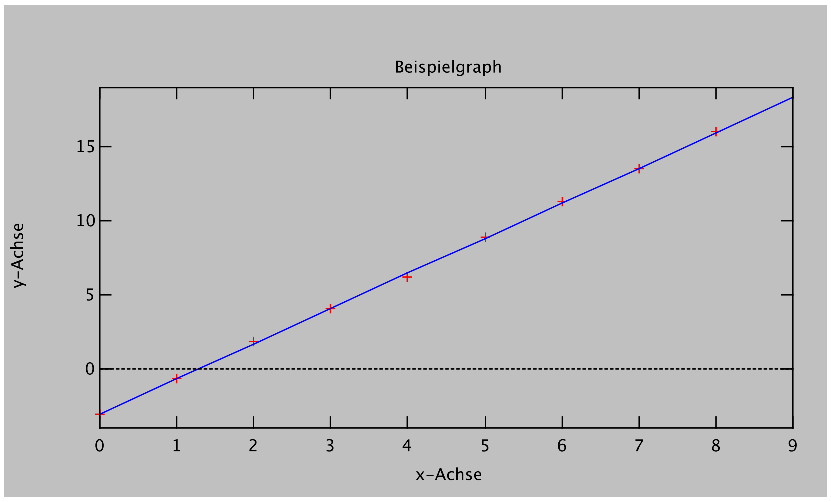
Exercise 10 (Plotting)

Plot the datapoints of the following table using `plot` and colored symbols. Calculate the linear regression using `polyfit`, and plot the regression line in the same graph. Add title and labels, and export the graphic to a file suitable for inclusion in a text document.

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

Solution:

```
>> 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)) % Regression
>> xlabel("x-Achse")
>> ylabel("y-Achse")
>> title("Beispielgraph")
>> print("graph.eps") % not Applet or Midlet!
```



2.6 Polynomials (1)

Jasymca can handle polynomials with symbolic variables. In this chapter, however, we work with the Matlab/Octave/Scilab-approach of using vectors as list of polynomial coefficients: A polynomial of degree n is represented by a vector having $n+1$ elements, the element with index 1 being the coefficient of the highest exponent in the polynomial. With `poly(x)` a normal polynomial is created, whose zeros are the elements of `x`, `polyval(a,x)` returns functionvalues of the polynomial with coefficients `a` in the point `x`, `roots(a)` calculates the zeros, and `polyfit(x,y,n)` calculates the coefficients of the polynomial of degree n, whose graph passes through the points x ynd y. If their number is larger than $n+1$ a least square estimate is performed. The regression analysis in exercise 8 was performed using this method.

Exercise 11

The roots of a 4th-degree polynomial are $-4,-2,2,4$ and it intersects the y-axis at ($y=-64$). Calculate its coefficients:
2 Solutions:

```
>> 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
```

Exercise 12

On a grid with x, y-Koordinaten we have the following greyvalues of a digital image:

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

Which greyvalue do you expect at position $x = 2,35; y = 2,74$? Calculate the bicubic interpolation.

Solution:

```
>> 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 Files

Reading and writing files is simplest with the Jasymca application, since file access by applets and midlets is often restricted due to security measures. Menu-options are provided for working with files:

File : Open Script

A scriptfile is read into Jasymca. The scriptfile must be plain text, which can be generated with common texteditors (e.g. Notepad, TextWrangler, KWrite for the platforms windows, macos, linux). The content of the scriptfile is treated as if it were entered in the textinputfield of Jasymca, i.e. it must be a list of valid commands.

This can be used to read in data from a data aquisition system for analysis, or to load user programs (see chapter 2.8). Quite useful are lists of constants or material data which are often required for exercises.

It is possible to load files from the textinputfield without menu: In this case the file must end with ".m" (e.g. "data.m"), and reside within Jasymcas searchpath (see below). Entering the filename *without* the ending ".m" (in this example:`data`) loads the data.

File : Save History

A protocol of the running session is created with all entries in the command history. This can be loaded in subsequent sessions using "Open Script".

File : Add Path

The search path specifies the directories which Jasymca searches when loading files. The command `path` displays the list of these directories. No subdirectories are entered. If a directory is to be added to the searchpath, this can be done either using this menuoption, or the command `addpath('path')`. The name `path` must be quoted.

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

2.8 Programming

2.8.1 Functions

Programs can be created and run interactively. Programming a function is demonstrated in the following example of a function `ttwo(x)`, which multiplies its argument by 2. After the definition it can be used like any other Jasymca function.

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

Following the keyword `function` is the prototype with a return variable `y`. This replaces the construct `return y` of other programming languages.

If functions are to be reused later, they should be written to a textfile and saved somewhere in Jasymcas searchpath. The filename must be the function name extended by ".m", in the present example `ttwo.m`. In subsequent sessions the function `ttwo` can be used without separately loading the file. Several installed functions of Jasymca are provided using this mechanism.

2.8.2 Branches

`if  x  A  end`
Depending on the condition `x` one or several statements `A` are executed. The condition `x` must be an arbitrary expression, which evaluates to either 0 or 1. The *false*-case (i.e. x=0) can lead to another branch `B`:
`if  x   A   else   B   end`
As an example the Heavyside funkcion:

```
>> 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 Loops

Loops with condition `x` and statement(s) `A`:
`while   x   A   end`
The `while`-loop is repeated until `x` becomes false (`0`).

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

Loops with counter `z` and statement(s) `A`:

```
for  z = vector   A   end
```

In the `for`-loop the counter is formally initialized by a `vector`. In each execution of the loop the counter takes on the value of the next element of `vector`.

```
>> x=1;y=1;
>> for(x=1:0.1:100) y=xˆ2+y; end
>> y
y = 3.3383E6
```

2.8.4 Jumps

```
return, continue, break
```

A function may be prematurely left using `return`. `continue` and `break` are used in loops: `continue` jumps back to the start of the loop, and begins another cycle. `break` permanently leaves the loop.

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

2.9 Vectors and Matrices (2)

Several standardmatrices are created by means of functions without specifying individual elements: `ones(n,m)`, `zeros(n,m)`, `rand(n,m)` return matrices with elements 1, 0 or random numbers between 0 and 1. `eye(n,m)` has diagonalelements 1, else 0, and `hilb(n)` creates the n-th degree Hilbert-matrix.

```
>> 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
```

The following functions are provided for matrix calculations: `diag(x)` (extracts diagonal elements), `det(x)` (determinante), `eig(x)` (eigenvalues), `inv(x)` (inverse), `pinv(x)` (pseudoinverse). The adjunct matrix is created using the operator `’`.

```
>> 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
```

The nontrivial functions are all based on the LU-decomposition, which is also accessible as a function call `lu(x)`. It has 2 or 3 return values, therefor the left side of the equation must provide multiple variables, see example below:

```
>> M=[2 3 1; 4 4 5; 2 9 3]
>> [l,u,p]=lu(M)           % 2 or 3 return values
l =                        % left triangular matrix (perm.)
  0.5      0.14286  1
  1        0        0
  0.5      1        0
u =                        % right upper triangular matrix
  4        4        5
  0        7        0.5
  0        0        -1.5714
p =                        % permutation matrix
  0  0  1
  1  0  0
  0  1  0
```

Without preceding point the arithmetic operators function as matrix operators, e.g. `*` corresponds to matrix and vector multiplication.

```
>> x=[2,1,4]; y=[3,5,6];
>> x.*y      %  with point
ans = [ 6  5  24 ]
>> x*y       %  without point
ans = 35
```

If one of the arguments is a scalar datatype, the operation is repeated for each element of the other argument:

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

Matrix division corresponds to multiplication by the pseudoinverse. Using the operator `\` leads to left-division, which can be used to solve systems of linear equations:

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

Systems of linear equations can (and should) be solved directly with the function `linsolve(A,b)` which will be discussed in chapter 2.13.1.

2.9.1 LAPACK

The application jasymca (not the applet or midlet) contain JLAPACK [9], the Java-port of the LAPACK [10]-routines with extended and better algorithms for matrix calculations. However, these are limited to matrices with real coefficients in floating point format. The LAPACK routines are accessed by the following functions:

svd(*A*) Singular value decomposition of *A* (1 or 3 returnvalues).

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

qr(*A*) QR-decomposition of *A* (2 returnvalues).

```
>> 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*) Solves $A \cdot x = b$ (1 returnvalue). Example in chapter 2.13.1.

linlstsq(A, b) Solves $A \cdot x = b$, overdetermined (1 return value). For an example see insert “Comparison of LAPACK and Jasymca Routines”.

eigen(A) Eigenvalues of A (1 returnvalue).

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

Comparison of LAPACK and Jasymca Routines

We calculate the 4-th degree regression polynomial for the following x,y-data:

```
>> 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
```

The coefficients `p(1),p(2),p(3)` should vanish since `x` and `y` represent a perfect straight line. This is an unstable problem, and it can be easily extended to make Jasymca completely fail. In our second attempt we use the 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
```

The coefficients `p(1),p(2),p(3)` are now significantly smaller. This particular problem can be solved exactly using Jasymca-routines and exact numbers, which avoids any rounding errors:

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

2.10 Symbolic Variables

In contrast to the examples Octave and Matlab, Jasymca integrates numeric and symbolic datatypes at the core of the program; symbolic math is not treated as an add-on. This means that with few exceptions most operations accept any mixture of numeric and symbolic arguments using the same commands and commandsyntax.

Symbolic variables should not be confused with variables as discussed until now. These latter variables serve as address for an object in memory (the "environment"), while symbolic variables are algebraic objects on their own. That means if `x` is a conventional variable, entering `x` in the textinputfield makes Jasymca search in the environment for the corresponding object, which then replaces `x`. If however `x` is a symbolic variable, the same action will lead to the creation of a first-degree polynomial with variable `x` and coefficients 1 and 0.

In Octave-mode, each symbolic variable `x` must be declared as symbolic by entering `syms x` before using it. The command `clear x` deletes the symbolic (actually any) variable `x`.

```
>> x=3;              % nonsymbolic variable
>> x^2+3-2*sin(x)    % placeholder for '3'
ans = 11.718
>> syms x            % symbolic variable
>> x^2+3-2*sin(x)    % create function
ans = -2*sin(x)+(x^2+3)
```

2.11 Polynomials (2) and Rational Functions

We have learnt that polynomials may be represented by the vector of their coefficient. Using a symbolic variable `x` we will now create a symbolic polynomial `p`. Conversely, we can extract the coefficients from a symbolic polynomial using the function `coeff(p, x, exponent)`. The command `allroots(p)` returns the zeros.

```
>> a=[3 2 5 7 4];    % coefficients
>> syms x
>> y=polyval(a,x)    % symbolic polynomial
y = 3*x^4+2*x^3+5*x^2+7*x+4
>> coeff(y,x,3)      % get one coefficient
ans = 2
```

```
>> b=coeff(y,x,4:-1:0)  % or all at once
b = [ 3  2  5  7  4 ]
>> allroots(y)      % same as roots(a)
ans = [ 0.363-1.374i  0.363+1.374i
          -0.697-0.418i  -0.697+0.418i ]
```

Up to this point there is little advantage of using symbolic calculations, it is just another way of specifying a problem. The main benefit of symbolic calculations emerges when we are dealing with more than one symbolic variable, or, meaning essentially the same, when our polynomial has nonconstant coefficients. This case can be treated efficiently only with symbolic variables. Notice in the example below how the polynomial y is automatically multiplied through, and brought into a canonical form. In this form the symbolic variables are sorted alphabetically, i.e. `z` is main variable compared to `x`. The coefficients can be calculated for each variable separately.

```
>> 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 Roots

The command `allroots` functions with variable coefficients also, but only, if the polynomials degree in the main variable is smaller than 3, or it is biquadratic. If roots of other variables `x` are searched, one should use the more general `solve(p,x)`, which will be discussed in more detail later.

```
>> 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 Squarefree Decomposition

The decomposition of `p` in linear, quadratic, cubic etc factors is accomplished by `sqfr(p)`. Returned is a vector of factors sorted in ascending order of the exponents.

```
>> 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, Greatest Common Denominator

The division of two polynomials `p` and `q` in one polynomial and remainder is calculated using `divide(p,q)`. If the polynomials have more than one variable, an optional variable can be specified, which will be used for division. `gcd(p,q)` returns the greatest common denominator of two expressions. Both functions also work with numbers as arguments.

```
>> 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- and Imaginary Part

`realpart(expression)` and `imagpart(expression)` is used to decompose complex expressions. Symbolic variables in these expressions are assumed to be real-valued.

```
>> 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 Symbolic Transformations

2.12.1 Substitution

Parts of an expression may be replaced by other expressions using `subst(a,b,c)`: `a` is substituted for `b` in `c`. This is a powerful function with many uses.

First, it may be used to insert numbers for variables, in the example 3 for x in der formula $2\sqrt{x}\cdot e^{-x^2}$.

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

Second, one can replace a symbolic variable by a complex term. The expression is automatically updated to the canonical format. In the following example z^3+2 is inserted for 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
```

Finally, the term `b` itself may be a complex expression (in the example z^2+1). Jasymca then tries to identify this expression in `c` (example: $\frac{z\cdot x^3}{\sqrt{z^2+1}}$). This is accomplished by solving the equation $a=b$ for the symbolic variable in `b` (example: z), and inserting the solution in `c`. This does not always succeed, or there may be several solutions, which are returned as a vector.

```
>> 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 Simplifying and Collecting Expressions

The function `trigrat`(*expression*) applies a series of algorithms to *expression*.

- All numbers are transformed to exact format.
- Trigonometric functions are expanded to complex exponentials.
- Addition theorems for the exponentials are applied.
- Square roots are calculated and collected.
- Complex exponentials are backtransformed to trigonometric functions.

It is often required to apply `float(expression)` to the final result.

```
>> 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)
```

`trigexpand(expression)` expands trigonometric expressions to complex exponentials. It is the first step of the function `trigrat` above.

```
>> 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 Equations

2.13.1 Systems of Linear Equations

Solving systems of linear equations is accomplished by either the function `linsolve(A,b)` (all versions of Jasymca), or `linsolve2(A,b)` (LAPACK, not in applet and midlet). In both cases A is the quadratic matrix of the system of equations, and b a (row or column) vector representing the right-hand-side of the equations. The equations may be written as $A \cdot z = b$ and we solve for 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) % not Applet or Midlet
ans =
  -0.25
  -0.13636
  0.90909
```

For large numeric matrices one should use the LAPACK-version if available. The Jasymca version can also handle matrices containing exact or symbolic elements. To avoid rounding errors in these cases it is advisable to work with exact numbers if possible:

```
>> 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   % symbolic element
  1    2    3*y  4    5   % symbolic element
  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 Nonlinear Equations

“Equation” in the following means the equation `expression = 0`. Equations are solved for a symbolic variable `x` by the function `solve(expression, x)`. If `expression` is a quotient, then `nominator = 0` is solved. Jasymca uses the following strategy to solve equations:

1. First, all occurances of the variable `x` in `expression` are counted, both as free variable and embedded inside functions. Example: In $x^3 \cdot \sin(x) + 2x^2 - \sqrt{x-1}$ `x` occurs three times: as free variable, in $\sin(x)$ and in $\sqrt{x-1}$.

2. If this count is one, then we are dealing with a polynomic equation, which is solved for the polynomial’s main variable, e.g. `z`. This works always, if the polynomial’s degree is 2 or of it is biquadratic, otherwise only, if the coefficients are constant. In the next step the solution is solved for the desired variable `x`. As an example: Jasymca has to solve $\sin^2(x) - 2\sin(x) + 1 = 0$ for x. It first solves $z^2 - 2z + 1 = 0$ for z and then $\sin(x) = z$ for x. Examples with free variables:

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

   An example with functionvariable ($exp(j \cdot x)$):

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

3. If count is 2, only one case is further considered: The variable occurs free and inside squareroot. This squareroot is then isolated, the equation squared

and solved. This case leads to additional false solutions, which have to be sorted out manually.

```
>> 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 all other cases Jasymca gives up.

2.13.3 Systems of Nonlinear Equations

Coupled systems of equations can be solved by the function `algsys([expressions],[symbolic variables])`. First, all linear equations are solved using the Gauss-method, then each equation is fed through `solve()` and the solution used to eliminate one variable in all other expressions. The equations are treated in the order they are supplied. This method only works for simple systems. The solution is provided as vector of solutionvectors, each individual solution in as linear factor: In the first example below there is one solution with `xs=-2/3, a2=3/4, a0=2, a1=0`, the second example has two solutions.

```
>> 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 Calculus

2.14.1 Differentiation

`diff(function,x)` differentiates `function` with respect to the symbolic variable `x`. The main variable of `function` is used if `x` is not provided. Functions defined by user programs can often be handled as well.

```
>> 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))      % no variable specified
ans = 1              % use z=sin(x) as variable
>> diff(sin(x),x)    % more reasonable
ans = cos(x)
>> function y=ttwo(x) y=2*x; end
>> diff(ttwo(sin(x)),x)
ans = 2*cos(x)
```

2.14.2 Taylorpolynomial

`taylor(function, x, x0, n)` calculates the `n`-th Taylorpolynomial in the symbolic variable `x` at the point `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 Indefinite Integral

`integrate(function, x)` integrates expression `function` with respect to the symbolic variable `x`. Jasymca uses the following strategy:

1. Integrals of builtin-functions and all polynomials are provided:

   ```
   >> 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. If `function` is rational (i.e. quotient of two polynomials, whose coefficients do not depend on `x`)) we use the standard approach: Separate a polynomial part, then separate a square free part using Horowitz' [11] method, and finally integrate the rest using partial fractions. The final terms are collected to avoid complex expressions.

```
>> 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)           % control
ans = (x^3+2*x^2-x+1)/(x^3+3*x^2+x+3)
```

3. Expressions of type $g(f(x)) \cdot f'(x)$ and $\frac{f'(x)}{f(x)}$ are detected:

```
>> 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. Substitutions of type $(a \cdot x + b)$ are applied:

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

5. Products $polynomial(x) \cdot f(x)$ are fed through partial integration. This solves all cases where f is one of *exp, sin , cos , log , atan*.

```
>> 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. All trig and exp-functions are normalized. This solves any expression, which is the product of any number and any type of exponentials and trigonometric functions, and some cases of rational expressions of trig- and exp-functions

   ```
   >> 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))
   ```

7. The special case $\sqrt{ax^2+bx+c}$ is implemented:

   ```
   >> 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. Clever substitutions may be supplied manually through `subst()`. If all fails, integrate numerically using `quad` or `romberg`.

The symbolic variable may be omitted if it is the main variable of `expression`. Integrations can be quickly verified using `diff()` on the result.

2.14.4 Numerical Integration

Two routines are supplied for numerical integration, which are both not very sophisticated. `quad('expression',ll,ul)` is modelled after the Octave/-Matlab integration function, but much simpler. Simpson's method is applied with a fixed number of nodes. This function uses the "eval"-method rather than symbolic variables. The function has to be supplied as quoted string, and must be compatible with vector arguments. Finally, the variablename must be `x`. `romberg` uses symbolic function definitions, and a symbolic variable has to be supplied. The maximum number of iterations is set by the variable `rombergit` (default 11) and accuracy by `rombergtol` (default: 10^{-4}).

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

2.14.5 Differential Equations

`ode(expression,y,x)` solves the linear first-order differential equation $y' = f(x) \cdot y + g(x)$. `expression` is the complete right-hand-side of the equation, `x` and `y` are symbolic variables. Free constants in the solution are marked `C`.

```
>> 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-Mode

The userinterface can be switched to Maxima-mode during a running session by selecting the menuoption *Run - Maxima Mode* and back to Octave-mode by selecting *Run - Octave Mode*. All variables keep their values; they can be cleared if required by clicking *Run - Clear Environment*. Normally Jasymca starts up in Octave-mode; see chapter 5 for instructions on how to start up in Maxima-mode.

Not only do the two user-interfaces differ in grammar and semantics regarding some operators and functions (see chapter 4 for an overview): In Maxima-mode, symbolic variables need not be declared as such by `syms`. Jasymca automatically creates symbolic variables for each unknown character sequence. Return values are automatically assigned to variables named `d1,d2,d3,..`, and remain accessible throughout the session. These two features make Maxima-mode favourable for work with symbolic expressions, since few variables need to be declared. It is less suitable for large amounts of data since each intermediate result is saved, and fills up memory.

Some random examples follow. Commands in Maxima-mode must be concluded with a semicolon and may extend beyond several lines, i.e. hitting “return” without the semicolon leaves Jasymca waiting for more input.

```
c1) [2,3,4]+[4,5,6];     % semicolon!
    d1 = [ 6  8  10 ]    % automatic variable d1
(c2) d1[2]*d1[3];        % index in brackets
    d2 = 80
```

```
(c3) xˆ2+3 = y*x-2;      % automatic symbolic vars
    d3 = -x*y+(xˆ2+5)    % equations are differences
(c4) choose(n,k):=n!/((n-k)!*k!);  % defining functions
choose
(c5) sum(1/kˆ2,k,1,1000);% 2. version of sum
    d4 = 1.6439
(c6) allroots( (x-1)ˆ3*(x-2)ˆ2*(x-3)*(x-4) );
    d5 = [ 4  3  2  2  1  1  1 ]
(c7) allroots(xˆ2+1);
    d6 = [ 1*i  -1*i ]
c8) a:x+2-y;           % assignment with ':'
    a = -y+(x+2)
```

Working with m-files is not supported, therefor all functions loaded by this mechanism do not work in Maxima-mode. Working with vectors and matrices is also less comfortable: While the builtin functions (determinante, inverse, LAPACK-functions, etc) are available, the conveniant methods for indexing, extraction and insertion (colon operator) are not supported.

Working with files is accomplished using the `loadfile` und `save`- functions, or the menu-fileoptions.

4 Reference

4.1 General

Textinput is concluded with lineend (linefeed or carriage return character) in Octave-mode, and semicolon together with lineend in Maxima-mode. One textinput may consist of several statements separated by commas or semicolons. In Octave-mode the concluding character determines, whether the command's output is displayed in the textconsole (not with semicolon). Comments are marked in both modes by the character `%` or #; the respective line is ignored starting at this letter. Variable names are case-sensitive, not, however, the names of builtin functions. Long calculations and infinite loops in programs can be interrupted using the menu-option *Run - Interrupt* or by typing `Ctrl-c`. This does not work, however, with builtin functions.

The following list summarizes how argument data types in the subsequent reference tables are characterized.

Example	Font	Description
`long`	typewriter	constant literal text
name	slanted	textstring, always quoted (single ' or double ")
var	italic	algebraic expression, with or without symbols, scalar, vector, matrix.
vector	italic	vector, numeric or symbolic (except Lapack)
matrix	italic	matrix, numeric or symbolic (except Lapack)
sym	italic	symbolic variable
COUNT	small capitals	integer, often > 0 required.

4.2 Commands

Commands are used for setting modes and options. In contrast to functions, arguments my be supplied without parenthesis. Only one command per textinput is allowed.

Name	Option	Description	Mod	Ref

format	`short`
	`long`
	BASE COUNT
		displayformat for floating-point numbers: 5 digits, all digits or in BASE with COUNT digits.	M,O	

syms	sym_1 [$, sym_2, \ldots$]	declare $sym_1, \ldots$ as symbolic variable.	O	
clear	var_1 [$, var_2, \ldots$]	delete variables $var_1, \ldots$	M,O	
who		display all variables	M,O	
path		show searchpath	M,O	
addpath	*path*	add to searchpath	M,O	
hold		lock/unlock graphic window	M,O	
hold	`on`	lock graphic window	M,O	
hold	`off`	unlock graphic window	M,O	

4.3 Operators

4.3.1 Define

Type	Example O-Mode	Example M-Mode
Numer	`-3.214e-12`	`-3.214e-12`
Vector	`[1 2 3 4]`	`[1,2,3,4]`
Matrix	`[1 2 ; 3 4]`	`matrix([1,2],[3,4])`
Vectorelement	`x(2)`	`x[2]`
Matrixelement	`x(2,1)`	`x[2,1]`
Range	`2:5`	
Variable	`x = 2.321`	`x : 2.321;`

4.3.2 Calculate

This table shows operators with one example each for Octave-mode (O-mode) and Maxima-mode (M-mode). Operations are executed according to their precedence (*prec*), and, if these are equal, in *order*.

Function	O-Mode	M-Mode	Prec	Order
Addition	x + y	x + y	4	left-right
Subtraction	x - y	x - y	4	left-right
Scalar Multiplication	x .* y	x * y	3	left-right
Vector/Matrix Multiplication	x * y	x . y	3	left-right
Scalar Division	x ./ y	x / y	3	left-right
Matrix Division (right)	x / y		3	left-right
Matrix Division (left)	x \ y		3	left-right
Scalar Exponentiation	x .^ y	x ^ y	1	left-right
Vector/Matrix Exponentiation	x ^ y	x ‘y	1	left-right
Range	x:y:z		5	left-right
Assignment	x = z	x : y	10	right-left
Assignment	x += z		10	right-left
Assignment	x -= z		10	right-left
Assignment	x /= z		10	right-left
Assignment	x *= z		10	right-left
Preincrement	++x	++x	10	left-right
Predecrement	--x	--x	10	left-right
Postincrement	x++	x++	10	right-left
Postdecrement	x--	x--	10	right-left
Adjunct	x′		1	right-left
Factorial		x!	1	left-right

4.3.3 Comparison and Logical Operators

Function	O-Mode	M-Mode	Prec	Order
Less	x < y	x < y	6	left-right
Less or equal	x <= y	x <= y	6	left-right
Larger	x > y	x > y	6	left-right
Larger or equal	x >= y	x >= y	6	left-right
Equal	x == y	x == y	6	left-right
Not equal	x ~= y	x ~= y	6	left-right
And	x & y	x & y	7	left-right
Or	x | y	x | y	9	left-right
Not	~x	~x	8	left-right

4.4 Programming

Both modes provide similar constructs with different grammar.

Oktave Mode

Branch	`if x==0 xp=1; end`
	`if x==0 xp=1; else xp=2; end`
Loop	`while x<17 x=x+1; end`
	`for i=0:100 x=x+i; end`
Jump	`return, continue, break`
Function	`function y=ttwo(x) y=2*x; end`
Evaluate	`eval('x=24')`
Textoutput	`printf('Ergebnis=%f', x)`
Errormessage	`error('Fehler')`

Maxima Mode

Branch	`if x==0 then ( xp:1 );`
	`if x==0 then ( xp:1 ) else ( xp:2 );`
Loop	`while x<17 do ( x:x+1 );`
	`for i:0 step 1 thru 100 do ( x:x+i );`
Jump	`return, continue, break`
Function	`ttwo(x) := 2*x;`
Block	`block( [x,y], x:1, y:1, z:z+x+y );`
Textoutput	`printf('Ergebnis=%f', x);`
Error	`error('Fehler');`

4.5 Functions

4.5.1 Scalar

Name(Arguments)	Function	Mod	Ref
float(*var*)	*var* as floating point number	M,O	
rat(*var*)	*var* as exact number	M,O	
realpart(*var*)	realpart of *var*	M,O	
imagpart(*var*)	imaginary part of *var*	M,O	
abs(*var*)	absolute value of *var*	M,O	
sign(*var*)	sign of *var*	M,O	
conj(*var*)	*var* conjugate complex	M,O	
angle(*var*)	angle of *var*	M,O	
cfs($var\ [, var_T]$)	continued fraction expansion of *var* with accuracy var_T	M,O	
primes(VAR)	VAR decomposed into primes	M,O	

4.5.2 Scalar Functions

Name(Arguments)	Function	Mod	Ref
sqrt(*var*)	squareroot	M,O	
exp(*var*)	exponential	M,O	
log(*var*)	natural logarithm	M,O	
sinh(*var*)	hyperbolic sine	O	
cosh(*var*)	hyperbolic cosine	O	
asinh(*var*)	hyperbolic areasine	O	
acosh(*var*)	hyperbolic areacosine	O	
sech(*var*)	hyperbolic secans	O	
csch(*var*)	hyperbolic cosecans	O	
asech(*var*)	hyperbolic areasecans	O	
acsch(*var*)	hyperbolic areacosecans	O	
sin(*var*)	sine (radian)	M,O	
cos(*var*)	cosine (radian)	M,O	
tan(*var*)	tangens (radian)	M,O	
asin(*var*)	arcsine (radian)	M,O	
acos(*var*)	arccosine (radian)	M,O	
atan(*var*)	arctangens (radian)	M,O	
atan2(var_1, var_2)	arctangens (radian)	M,O	
sec(*var*)	secans (radian)	O	
csc(*var*)	cosecans (radian)	O	
asec(*var*)	arcsecans (radian)	O	
acsc(*var*)	arccosecans (radian)	O	
factorial(N)	factorial $N!$	M,O	
nchoosek(N,K)	binomial coefficient $\binom{N}{K}$	O	
gamma(*var*)	gammafunction	M,O	
gammaln(*var*)	logarithm of gammafunction	M,O	

4.5.3 Vectors and Matrices

Name(Arguments)	Function	Mod	Ref
linspace(*var*$_1$,*var*$_2$,COUNT)	vector with COUNT numbers ranging from *var*$_1$ to *var*$_2$	O	
length(*vector*)	number of elements in *vector*	M,O	
zeros(ROWS[,COLUMNS])	matrix of zeros	M,O	
ones(ROWS[,COLUMNS])	matrix of ones	M,O	
eye(ROWS[,COLUMNS])	matrix with diagonal one	M,O	
rand(ROWS[,COLUMNS])	matrix of random numbers	M,O	
hilb(RANK)	Hilbertmatrix	M,O	
invhilb(RANK)	Inverse Hilbertmatrix	O	
size(*matrix*)	number of rows and columns	M,O	
sum(*var*)	if *var* is a vector: sum of elements, if *var* is a matrix: sum of columns.	M,O	
find(*var*)	indices of nonvanishing elements	M,O	
max(*var*)	largest element in *var*	M,O	
min(*var*)	smallest element in *var*	M,O	
diag(*var*,[OFFSET])	if *var* is a vector: matrix with *var* as diagonale, if *var* is matrix: diagonale as vector.	M,O	
det(*matrix*)	determinante	M,O	
eig(*matrix*)	eigenvalues	M,O	
inv(*matrix*)	inverse	M,O	
pinv(*matrix*)	pseudoinverse	M,O	
lu(*matrix*)	LU-decomposition	M,O	
svd(*matrix*)	singular value decomposition (Lapack)	M,O	
qr(*matrix*)	QR-decomposition (Lapack)	M,O	
eigen(*matrix*)	eigenvalues (Lapack)	M,O	

4.5.4 Polynomials

Name(Arguments)	Function	Mod	Ref
poly(*vector*)	coefficients of polynomial having roots *vector*	O	
polyval(*vector*, *var*)	functionvalue of polynomial with coefficients *vector* in the point *var*	O	
polyfit($vector_x$,$vector_y$,N)	fits N-th degree polynomial to data points having coordinates $vector_x$ and $vector_y$	O	
roots(*vector*)	roots of polynomial having coefficients *vector*	M,O	
coeff(*var*,*sym*,N)	coefficient of sym^n in *var*	M,O	
divide(var_1, var_2)	division $\frac{var_1}{var_2}$ with remainder	M,O	
gcd(var_1, var_2)	greatest common denominator	M,O	
sqfr(*var*)	squarefree decomposition	M,O	
allroots(*var*)	roots	M,O	

4.5.5 Equations and Expressions

Name(Arguments)	Function	Mod	Ref
subst(var_x,var_y,var_z)	substitute var_x for var_y in var_z	M,O	
trigrat(*var*)	trigonometric and other simplifications	M,O	
trigexp(*var*)	trigonometric expansion	M,O	
solve(*var*, *Sym*)	solves $var = 0$ for *Sym*.	M,O	
algsys([var_1, $var_2, \ldots$],[sym_1, $sym_2, \ldots$])	solves the system of equations $var_1 = 0, var_2 = 0, \ldots$ for $sym_1, sym_2, \ldots$.	M,O	
linsolve(*matrix*, *vector*)	solves $matrix \cdot x = vector$ for x	M,O	
linsolve2(*matrix*, *vector*)	solves $matrix \cdot x = vector$ for x (Lapack)	M,O	
linlstsq(*matrix*, *vector*)	solves $matrix \cdot x = vector$ for x, overdetermined (Lapack)	M,O	

4.5.6 Calculus

Name(Arguments)	Function	Mod	Ref
sum(*var*,*sym*, INITIAL, FINAL)	$\sum_{sym=initial}^{sym=final} var$	M,O	
lsum(*var*,*sym*,*vector*)	$\sum_{sym\in vector} var$	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(*expr*, var_a, var_b)	$\int_{var_a}^{var_b} exprdx$	O	
taylor(var_f, *sym*, var_p, N)	N-th Taylorpolynomial of the function var_f with variable *sym* in point var_p.	M,O	
ode(*var*, sym_y, sym_x)	solves the linear differential equation $y' = f(x) \cdot y + g(x)$. *var* is the right-hand-side of this equation.	M,O	
fzero(*expr*,var_a, var_b)	find function zero between var_a and var_b	O	

4.5.7 Plots

Name(Arguments)	Function	Mod	Ref
plot(*x*,*y* [,*option*])	Plot *x* versus *y* using *option* *x* and *y* are equalsized vectors. *option* is a string, specifying color (one of `r,g,b,y,m,c,w,k`) and symbol (one of `+,*,o,x`). Default: blue lines.	M,O	
loglog(*x*,*y*[,*option*])	logarithmic plot	M,O	
linlog(*x*,*y*[,*option*])	semilogarithmic plot	M,O	
loglin(*x*,*y*[,*option*])	semilogarithmic plot	M,O	
print(*name*)	write graphic in eps-file.	M,O	

5 Installation

Applet

The applet is automatically started when visiting the Jasymca homepage [8] with a java-enabled (version ≥ 1.5) webbrowser, no installation required. To create a local copy of the Jasymca homepage move the directory `Applet` of the distribution onto your computer, and click the file `indexEN.html` (or open it in your browser). Included is the online reference, the other links in `indexEN.html` are empty.

Application

The application Jasymca includes the LAPACK routines and allows you to work with files. Move the directory `Applikation` onto your computer. It contains all data and programs required to run Jasymca. A current version (≥ 1.5) of the Java-Runtime is required and can be downloaded from Sun [12]. Jasymca may also be installed on removeable media (usb-sticks, memory cards) and read-only media (cd-rom).

To start the program try to double-click the program icon `jasymca.jar` in `Applikation`. If this does not work, open a command window on your computer, navigate to the directory `Applikation` and issue the command
`java -jar jasymca.jar`. Jasymca starts up in Maxima-mode by the command
`java -jar jasymca.jar ui=Maxima`.

Upon startup Jasymca searches a file named `Jasymca.Octave.rc` or `Jasymca.Maxima.rc` depending on startup mode. If the environmental variable `JASYMCA_RC` is set on your system, Jasymca searches for a startup file under this name extended by `.Octave.rc` or `.Maxima.rc`. This file may be used to load often used constants or functions.

Midlet

The midlet version is mostly compatible with the applet including the plotting function. The user interface is taken from my program FnattLavME [2], and details may be looked up there.

The installation on mobile devices like cellphones or pdas depends on the system. Required is a java installation providing the APIs CLDC 1.1, MIDP 2.0, JSR-75. Most current systems are either equipped with Java or it can be installed from the manufacturer. The Jasymca midlet consists of the files `Jasymca.jar` and `Jasymca.jad`, both in the directory `Midlet`. It can be installed over the internet, or through a local connection (USB/Bluetooth/...) from your PC. Please consult the manual of your mobile device. Detailled instructions for installations on Palm Tungsten E and Nokia 6230i are available here [3].

6 License

Jasymca is free and open software. you can redistribute it and/or modify it under the terms of the GNU General Public License. The license text is available in the distribution and and can be downloaded from the homepage [8].

This document may only be copied for private purposes.

Jasymca uses modules and parts from other programs which are listed together with their licenses in this chapter.

- The files BigInteger.java, Random.java, and MPN.java are slightly modified versions from the GNU-Classpath [14] project.

  ```
  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.
  ```

- The files JMath.java and SFun.java are derived from public sourdes by 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
  ```

- BLAS, LAPACK und JLAPACK:

References

[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/indexEN.html

[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/