

# Jasymca - Symbolic Calculator for Mobile Devices

Helmut Dersch

March 7, 2006

## Abstract

Jasymca is a symbolic calculator written for mobile phones and PDAs. It solves and manipulates equations, handles basic calculus problems, and provides a few more typical functions of computer algebra systems. The syntax is loosely related to GNU-Maxima. While it is fun to play with mobile devices, it teaches the use of computer algebra systems, and may even be used to solve some real world problems.

## Contents

<b>1</b>	<b>Introduction</b>	<b>3</b>
<b>2</b>	<b>Installation</b>	<b>4</b>
2.1	Example 1 . . . . .	4
2.2	Example 2 . . . . .	5
<b>3</b>	<b>Usage</b>	<b>6</b>
3.1	User Interface . . . . .	6
3.2	Calculations . . . . .	6
3.2.1	Operators . . . . .	6
3.2.2	Variables . . . . .	7
3.2.3	Functions . . . . .	7
3.2.4	User Functions . . . . .	8
3.2.5	DIVIDE ( <i>expr1,expr2</i> )	
	DIVIDE ( <i>expr1,expr2, var</i> ) . . . . .	9

3.2.6	GCD ( <i>expr1,expr2</i> ) . . . . .	9
3.2.7	SUM ( <i>exp, ind, lo, hi</i> )	
	LSUM ( <i>exp, ind, list</i> ) . . . . .	9
3.3	Manipulating Expressions . . . . .	10
3.3.1	EXPAND( <i>expr</i> ) . . . . .	10
3.3.2	RAT ( <i>expr</i> )	
	FLOAT ( <i>expr</i> ) . . . . .	10
3.3.3	REALPART ( <i>expr</i> )	
	IMAGPART ( <i>expr</i> ) . . . . .	11
3.3.4	SQFR ( <i>expr</i> ) . . . . .	11
3.3.5	SUBST ( <i>a, b, c</i> ) . . . . .	11
3.3.6	TRIGRAT ( <i>expr</i> ) . . . . .	12
3.3.7	TRIGEXP ( <i>expr</i> ) . . . . .	13
3.4	Solving Equations . . . . .	13
3.4.1	SOLVE ( <i>expr, var</i> ) . . . . .	13
3.4.2	ALGSYS ( <i>[expr1, expr2,...],[var1,var2,...]</i> ) . . . . .	14
3.4.3	ALLROOTS ( <i>expr</i> ) . . . . .	15
3.5	Calculus . . . . .	15
3.5.1	DIFF ( <i>expr</i> )	
	DIFF ( <i>expr, var</i> ) . . . . .	15
3.5.2	INTEGRATE ( <i>expr</i> )	
	INTEGRATE ( <i>expr,var</i> ) . . . . .	16
3.5.3	ROMBERG ( <i>exp,var,ll,ul</i> ) . . . . .	17
3.5.4	TAYLOR ( <i>exp, var, pt, pow</i> ) . . . . .	18
3.6	Input and Output . . . . .	18
3.6.1	SAVE ( <i>filename, var1, var2, ...</i> ) . . . . .	18
3.6.2	LOADFILE ( <i>filename</i> ) . . . . .	19
<b>4</b>	<b>Building from sources</b>	<b>19</b>
<b>5</b>	<b>Design considerations and related work</b>	<b>20</b>
5.1	Jsc1-meditor . . . . .	20
5.2	Yacas . . . . .	21
5.3	Jasymca . . . . .	21
<b>6</b>	<b>License</b>	<b>22</b>

# 1 Introduction

Jasymca (Java Symbolic Calculator) is a program written in Java employing the Microedition [1] API (“Midlet”). The main capabilities are:

- Differential calculus, symbolic integration of rational and a few more types of functions.
- Symbolic solution of linear, quadratic and a more types of equations (sqrt, hyperbolic, trigonometric) and systems of such equations.
- Trigonometric simplifications. Conversions to and from complex exponentials.
- Polynomial handling, roots, gcd, square free decomposition. . .
- Definition of user functions, which can be used in symbolic calculations.
- Exakt arithmetic using big integer rationals.
- Numeric routines for root finding and integration.
- Basic numeric calculator capabilities.
- Syntax loosely related to GNU-Maxima [2].
- Binary size less than 200kByte.

Why symbolic calculation on mobile devices? There are many powerful pocket calculators available, but their use has disadvantages. The reason is that almost any engineering student (or other technical professional) at some point has to learn one of the serious math tools for computer workstations (Maxima [2], Matlab [3], Octave [4], Maple [5], Mathematica [6] . . .). Using one of the nifty pocket calculators shifts this point into the future, and makes it necessary to learn several user interfaces and concepts which is wasted effort. As math instructor I witness, that the more powerful the personal pocket calculator is, the larger is the resistance against using a workstation, and consequently many miss this important skill.

By providing similar programs for doing math on mobile devices, we have to learn only one concept, which we can equally apply to the desktop computer. Specifically, if you manage to run Jasymca on a mobile device, you can seamlessly proceed with GNU-Maxima on the workstation. This lowers the barrier to use the

workstation for math. So, Jasymca is meant to serve a pedagogical purpose, but it may also be fun to use a popular toy like cellphones for math, and it may even be able to solve a few real problems.

## 2 Installation

There are two versions of Jasymca: One compiled for J2ME to be installed on mobile devices (*Jasymca.jar*), and one compiled for J2SE which can be installed on any desktop computer with Java-runtime installed (*JasymcaSE.jar*). Both are included in the downloadable [7] archive.

To install Jasymca on your desktop computer, copy the file *JasymcaSE.jar* to some convenient folder. Run the program by executing the command  
`java -cp JasymcaSE.jar Jasymca.`

To install Jasymca on a mobile device, first check that a suitable Java runtime is installed and the following requirements are satisfied:

- Java API specs CLDC 1.1, MIDP 2.0, JSR-75.
- Maximum JAR-size > 200kByte.
- Heapsize >= 2MByte

. For Nokia phones, this information can be obtained from Nokia's website [8]. I am running Jasymca on a Nokia 6230i. Most Palm handhelds are suitable, e.g. the Tungsten E running the IBM-Java runtime. Jasymca also runs in the WTK2.2 J2ME emulator from Sun [1], and should therefore be useable on many more devices.

Installation depends on the particular device. I suggest to check the manual of your device for Midlet-installation procedures. The following two examples are for Linux-users, who can not run the Windows programs supplied with the mobile devices.

### 2.1 Example 1

Installation onto Nokia 6230i from Linux desktop PC:

- The Nokia phone has all Java software preinstalled. Enable bluetooth connections for software installation.

- On the Linux PC, install `gammu` [9] and bluetooth support.
- Enter the directory, which contains the files `Jasymca.jar` and `Jasymca.jad`, and execute the command  

```
gammu --nokiaaddfile Application Jasymca
```

You have to acknowledge the installation request on the phone.

## 2.2 Example 2

Installation onto Palm Tungsten E from Linux desktop PC:

- Download and install the free Java J2ME runtime for Palm PDAs [10].
- Java-Midlets are installed either by packaging them into PALM executables (`.prc`) and using the standard sync-operation. Alternatively, you can install Midlets using the onboard Midlet installer `MidletHQ`, which comes with the Java Runtime. I will explain this route in the following.
- On your Linux-desktop, start any HTTP-server and put `Jasymca.jar` into the server directory. I use `MonkeyWeb` on a DSL-installation.
- Establish a TCP-connection between PC and Palm. I use a `pppd-over-USB` connection. Irda, Bluetooth or WLAN should equally work.
- Start `MidletHQ` on the Palm, choose “Install MidLet” and type the URL of the file to install, e.g.  

```
http://192.168.1.2/Jasymca.jar.
```
- The file will be installed after you agree to install from an insecure source (which I certainly am), and you start the program by tapping the desktop icon.

Upon startup `Jasymca` tries to open and read an initialisation file named `Jasymca.rc` if present. This should be a plain textfile containing valid `Jasymca` expressions. Useful are assignments and custom function definitions. On J2SE devices, the name of the startup file can be set by the environmental variable `JASYMCA_RC`. On J2ME devices the file is supposed to be in the folder `vfs`. The file can be created and edited using the build-in text-editor and filebrowser (Use the menu-option *File*). The `vfs`-directory is located in the persistent storage area of the device independent of memory cards. It can only be accessed by the `Jasymca`-application, but you can copy it to any other location using `Jasymca`’s filebrowser.

## 3 Usage

This section describes all Jasymca commands and features. By following the examples it may be used as tutorial.

### 3.1 User Interface

User interaction proceeds via the worksheet concept. Commands are edited and typed using a text input frame. On a cellphone this is similar to typing SMS-messages. The command string is supplied to Jasymca by hitting the Enter button or Enter menu option. Commands and variable names in Jasymca are case-insensitive. The command line must be terminated by a semicolon. Jasymca's answer is supplied in a separate window together with a copy of the input. Each answer is labeled with a unique index starting with the letter d. It may later be referenced using this index. The worksheet is scrollable (options  $\wedge$  and  $\vee$ ), and previous inputs can be recalled in the text input window using the history buttons or menu options labeled  $<$  and  $>$ . No history function is available in the J2SE version. The J2SE version can be exited by typing `exit()` ;.

```
(c1) 3+2-12+7/2.3;  
(d1)      -3.9565217391304346  
(c2) 5-1+2*6;  
(d2)      16.0  
(c3) d1+d2;  
(d3)      12.043478260869566  
(c4) d3/5;  
(d4)      2.408695652173913
```

### 3.2 Calculations

#### 3.2.1 Operators

The standard math operators are  $+$ ,  $-$ ,  $*$ ,  $/$ . For exponentiation one may use either of  $\wedge$  or  $**$ .  $i$  is the imaginary unit, and  $\text{pi}$  the constant  $\pi$ . Use  $!$  for factorials, and round parenthesis  $()$  for grouping expressions. Equations may be entered using  $=$ , they are internally converted to an expression  $lhs - rhs$ . Vectors, matrices and their elements are marked with  $[]$ . Commas are used for lists of elements or expression-lists.

```

(c1) 7^3;
(d1)      343.0
(c2) (3+2*i)**4;
(d2)      (-119.0+120.0*i)
(c3) 7!;
(d3)      5040.0
(c4) 2^(2^(2^3));
(d4)      1.157920892373162E77
(c5) ((2^2)^2)^3;
(d5)      4096.0
(c6) [2,3,4]+[4,5,6];
(d6)      [6.0, 8.0, 10.0]
(c7) d6[2]*d6[3];
(d7)      80.0
(c8) x^2+3 = y*x-2;
(d8)      ((x^2.0)+3.0)-((y*x)-2.0)

```

### 3.2.2 Variables

Assignments of expressions to names make use of the colon `:` operator. They are used to reference expressions and should not be used to assign values to mathematical variables (use the *SUBST* command for this). To clear a previously defined name, assign the string `null` to it.

```

(c1) x:3;
(d1)      3.0
(c2) 12+x;
(d2)      15.0
(c3) x:null;
(d3)      null
(c4) 12+x;
(d4)      (12.0+x)

```

### 3.2.3 Functions

The following numeric functions are predefined: *sqrt*, *exp*, *log*, *sin*, *cos*, *tan*, *atan*, *sign*. More functions can be defined if needed. Functions applied to inexact numbers are always evaluated. Functions of exact numbers and constants (*pi*) are eval-

uated only after calling *FLOAT* except for special values like  $e^{i\pi}$ . Special values are defined for *log* and *exp*. They may be applied to trigonometric functions by *TRIGRAT*. Squareroots of exact numbers are evaluated up to squarefree rational roots.

```
(c1) sin(3.12);
(d1)      0.02159097572609596
(c2) sin(rat(3.12));
(d2)      sin(78/25)
(c3) exp(i*pi);
(d3)      -1.0
(c4) sqrt(1553.375);
(d4)      39.41287860585674
(c5) sqrt(rat(1553.375));
(d5)      17/4*sqrt(86)
(c6) sqrt(rat(-12));
(d6)      2*i*sqrt(3)
(c7) sin(pi);
(d7)      sin(pi)
(c8) trigrat(sin(pi));
(d8)      0
(c9) float(sin(pi));
(d9)      1.2246467991473532E-16
```

### 3.2.4 User Functions

User functions are defined by the `:=` operator. They may be multivariate, and they can be used like any other function, e.g. differentiated, integrated, used in solving equations etc. Only very few functions are predefined. A set of often used functions may be loaded at startup. User functions are as fast as compiled functions, so there is no point in programming them in Java and blowing up the binary.

```
(c1) sinh(x) := 1/2*(exp(x) - exp(-x));
(d1)      sinh
(c2) sinh(12);
(d2)      81377.39570642984
(c3) diff(sinh(x), x);
```



```

(d3)      (0.5*exp(x)+0.5*exp(-x))
(c4) choose(n,k):=n!/((n-k)!*k!);
(d4)      choose
(c5) atan2(y,x):=atan(y/x)+sign(y)*(1-sign(x))/2 * pi;
(d5)      atan2
(c6) diff(atan2(x+1,x-1),x);
(d6)      -1.0/(x^2+1.0)

```

### 3.2.5 **DIVIDE** (*expr1,expr2*) **DIVIDE** (*expr1,expr2, var*)

Divide with remainder. Expressions are numbers, polynomials or rational functions. *var* the main variable. Returns a vector of the quotient and remainder. Complex numbers return the ratio of the imaginary parts.

```

(c1) divide(122344,7623);
(d1)      [16, 376]
(c2) divide(2+i,3+2*i);
(d2)      [1/2, 1/2]
(c3) divide(x^3*z-1,x*z-x,x);
(d3)      [x^2*z/(z-1.0), -1.0]
(c4) divide(x^3*z-1,x*z-x,z);
(d4)      [x^2, (x^3-1.0)]

```

### 3.2.6 **GCD** (*expr1,expr2*)

Calculates the greatest common denominator of *expr1* and *expr2*. The expressions may be numbers or single and multivariate polynomials.

```

(c1) gcd(32897397,24552502);
(d1)      377
(c2) gcd(z*x^5-z,x^2-2*x+1);
(d2)      (x-1)

```

### 3.2.7 **SUM** (*exp, ind, lo, hi*) **LSUM** (*exp, ind, list*)

Sum of *exp*. *exp* starts at *lo* and is incremented until *hi*, or *exp* uses all elements of *list*, which must evaluate to a vector.

```
(c1) sum(1/k^2, k, 1, 1000);
(d1)      1.6439345666815615
(c2) lsum(x, x, allroots(x^4+x^3-2*x));
(d2)      -1.0
```

### 3.3 Manipulating Expressions

#### 3.3.1 EXPAND(*expr*)

Evaluates expression converting it to canonical format. Polynomial expressions are multiplied through, and variables are ordered alphabetically

```
(c1) (x-1)*(x^2+3-2*x)+(3-x^4)*2*x;
(d1)      ((x-1.0)*(x^2.0)+(3.0-(2.0*x)))+
          ((3.0-(x^4.0))*2.0*x)
(c2) expand(d1);
(d2)      (-2.0*x^5+x^3-3.0*x^2+11.0*x-3.0)
(c3) expand((x+z)*(y+z)*(x+y));
(d3)      ((y+x)*z^2+(y^2+2.0*x*y+x^2)*z+(x*y^2+x^2*y))
```

#### 3.3.2 RAT(*expr*) FLOAT(*expr*)

Converts numbers and numbers in *expr* between rational and floating point format. Arithmetic operations with rational numbers are exact while floating point arithmetic may introduce rounding errors. Many Jasympca operations (e.g. *SOLVE*) require rational numbers and call *RAT* in their initialisation step. Most builtin functions like *exp* or *sin* operate on floating point numbers. The conversion to rational numbers uses continued fraction expansion with relative accuracy determined by the constant `RATEPSILON`. This constant defaults to  $2 \cdot 10^{-8}$ .

```
(c1) sqrt(2);
(d1)      1.4142135623730951
(c2) rat(d1);
(d2)      8119/5741
(c3) ratepsilon:1e-14;
(d3)      1.0E-14
(c4) rat(d1);
(d4)      9369319/6625109
```

```
(c5) float (d2);
(d5)      1.4142135516460548
```

### 3.3.3 REALPART (*expr*) IMAGPART (*expr*)

Real and imaginary parts of expression. If expression contains variables, these are assumed to be real-valued.

```
(c1) (3+i*x)/(2-i*x);
(d1) ((3.0+(1.0*i*x))/(2.0-(1.0*i*x)))
(c2) realpart (d1);
(d2) (-x^2+6.0)/(x^2+4.0)
(c3) imagpart (d1);
(d3) 5.0*x/(x^2+4.0)
```

### 3.3.4 SQFR (*expr*)

Square-free decomposition of polynomial *expr*. Factors of equal multiplicity are grouped together. This decomposition is exact.

```
(c1) expand((x-1)^3*(x-2)^2*(x-3)*(x-4));
(d1) (x^7-14.0*x^6+80.0*x^5-242.0*x^4+
      419.0*x^3-416.0*x^2+220.0*x-48.0)
(c2) sqfr (d1);
(d2) (((x^2-7.0*x+12.0)*((x-2.0)^2.0))*((x-1.0)^3.0))
```

### 3.3.5 SUBST (*a, b, c*)

Substitute *a* for *b* in *c*. Versatile command for manipulating expressions with many uses. *b* may be a single name of a variable or a complete subexpression in *c*. In the former case, this command may be used to evaluate expressions at certain points. In the latter case, substitutions can be performed. Matching of *b* in *c* is performed on a lexical base.

```
(c1) 2*sqrt(x)*exp(-x^2);
(d1) ((2.0*sqrt(x))*exp((-x^2.0)))
(c2) subst(3,x,d1);
(d2) 4.275041016605005E-4
```

```

(c3)  x^3*sin(2.5*x+pi/7)/sqrt(z^2+1);
(d3)  ((x^3.0)*(sin(((2.5*x)+0.14285714285714285*pi)))/
      sqrt(((z^2.0)+1.0))))
(c4)  subst(pi,x,d3);
(d4)  pi^3*sin(2.642857142857143*pi)/sqrt(z^2+1.0)
(c5)  subst(y,z^2+1,d4);
(d5)  pi^3*sin(2.642857142857143*pi)/sqrt(y)
(c6)  float(d5);
(d6)  27.935689998518907/sqrt(y)
(c7)  x^3+2*x^2+x+7;
(d7)  (((x^3.0)+(2.0*(x^2.0)))+x)+7.0)
(c8)  subst(z^3+2,x,d7);
(d8)  (z^9+8.0*z^6+21.0*z^3+25.0)

```

### 3.3.6 TRIGRAT (*expr*)

A normalized representation of trigonometric expressions is calculated. The following filters are applied to *expr*:

- Numbers are rationalized using *RAT*.
- User defined functions are expanded.
- Trigonometric functions are converted to exponentials.
- Exponential functions are normalized according to the equation  $(e^x)^n = e^{nx}$ .
- All products of exponentials with equal variable are collected according to  $e^{ax} \cdot e^{bx} = e^{(a+b)x}$
- Complex exponentials are back converted to trigonometric functions.
- Squareroots are executed if possible.

This operation may be useful also for non-trigonometric functions, e.g. for exponentials or collection of squareroots.

```

(c1)  sin(x)^2+sin(x+2*pi/3)^2+sin(x+4*pi/3)^2;
(d1)  (((sin(x)^2.0)+
      (sin((x+0.6666666666666666*pi))^2.0))+
      (sin((x+1.3333333333333333*pi))^2.0))

```

```

(c2) trigrat(d1);
(d2)      3/2
(c3) trigrat(i/2*log(x+i*pi));
(d3)      (1/4*i*log(x^2+pi^2)+(1/2*atan(x/pi)-1/4*pi)
(c4) trigrat(sin((x+y)/2)*cos((x-y)/2));
(d4)      (1/2*sin(y)+1/2*sin(x))
(c5) trigrat(sqrt(4*y^2+4*x*y-4*y+x^2-2*x+1));
(d5)      (y+(1/2*x-1/2))

```

### 3.3.7 TRIGEXP (*expr*)

Expand trigonometric functions to complex exponentials. This function is always the first step of *TRIGRAT*.

```

(c1) trigexp(i*tan(i*x));
(d1)      (-exp(x)+exp(-x))/(exp(x)+exp(-x))
(c2) trigexp(atan(1-x^2));
(d2)      -1/2*i*log(-x^2+(1-1*i))/(x^2+(-1-1*i))

```

## 3.4 Solving Equations

### 3.4.1 SOLVE (*expr*, *var*)

Solve  $expr = 0$  for variable *var*. Strategy:

- Count number of occurrences of *var* in *expr*. Occurrence may be a polynomial variable or arguments to any function. Powers of occurrences are not added, i.e.  $\sin(2x)$  and  $\sin^2(2x)$  count as one occurrence. (NB:  $\sin(2x)$  like any other trigonometric expression is converted to exponentials before *SOLVE* proceeds. This is part of the standard normalization.)
- If count is one, solve the polynomial equation. First and second order polynomials, and powers of these (biquadratic etc) are solved symbolically, others numerically (in which case Jasympca gives up, if it encounters nonconstant coefficients).
- If occurrence is a function, invert it (if possible). This works for the predefined functions, and also for some userdefined functions.

- If count is two, only one case is further considered: The occurrence of  $var$  with  $\sqrt{Polynomial(var)}$ . In this case  $\sqrt{Polynomial(var)}$  is isolated, squared, and the resulting equation is solved. In all other cases *SOLVE* gives up.
- The final result is fed through *TRIGRAT* in order to recombine complex exponentials and logs to trigonometric functions. This step may not always succeed.

The solutions  $x_i$  are returned as a vector of linear factors  $x - x_i$  which may look weird. All complex solutions are usually provided. The *SUBST* command can be used to prove the result (see c6). This is sometimes necessary since the squaring step (see above) introduces wrong artificial solutions, which are not detected. The accuracy is mainly determined by the value of *RATEPSILON*, which governs the conversion to rational numbers. (NB: The Maxima *solve* command does not solve the example (c4) below).

```
(c1) solve(x^2-1, x);
(d1) [(x-1), (x+1)]
(c2) solve(x^2-2*x*b+b^2, x);
(d2) (x-b)
(c3) expand( sin(x)^2+2*cos(x)=0.5 );
(d3) (sin(x)^2+(2.0*cos(x)-0.5))
(c4) solve(d3, x);
(d4) [(x+12399/6898), (x-12399/6898),
      (x-36862/25635*i), (x+36862/25635*i)]
(c5) float(d4);
(d5) [(x+1.797477529718759), (x-1.797477529718759),
      (x-1.4379559196411158*i), (x+1.4379559196411158*i)]
(c6) subst(x-d4[1], x, d3);
(d6) (sin(12399/6898)^2+(2.0*cos(12399/6898)-0.5))
(c7) float(d6);
(d7) -3.361808387225551E-9
```

### 3.4.2 ALGSYS ([*expr1*, *expr2*,...],[*var1*,*var2*,...])

Solve system of equations. The number of variables must match the number of equations. Strategy:

- All linear equations (i.e. having constant coefficient with regards to  $var_i$ ) are identified. A pivoting algorithm is used to solve the corresponding variables and eliminate them from the remaining equations.
- The remaining equations are fed one at a time through *SOLVE*. If this does not work, it may help to change the order of the equations.
- Multiple solutions are properly collected, and a vector of solution vectors is returned. There may be artificial wrong solutions, see *SOLVE* above. The other remarks of *SOLVE* equally apply.

```
(c1) algsys([2=a0, a1=0, a2*xs^2+a1*xs+a0=3-xs, 2*a2*xs+a1=-1],
           [a2, a1, a0, xs]);
(d1)      [[(xs-2), (a2+1/4), (a0-2), a1]]
(c2) algsys([a*xs+3*a=3-xs^2, a=-2*xs], [a, xs]);
(d2)      [[(-sqrt(6)+(xs+3)), (2*sqrt(6)+(a-6))],
           [(sqrt(6)+(xs+3)), (-2*sqrt(6)+(a-6))]]
(c3) float(d2);
(d3)      [[(xs+0.5505102572168221), (a-1.1010205144336442)],
           [(xs+5.449489742783178), (a-10.898979485566356)]]
```

### 3.4.3 ALLROOTS (*expr*)

All numeric roots of polynomial *expr* are returned as vector. This routine internally calls *SQFR*, and then applies Bairstow's [11] algorithm to each factor.

```
(c1) expand((x-1)^3*(x-2)^2*(x-3)*(x-4));
(d1)      (x^7-14.0*x^6+80.0*x^5-242.0*x^4+
           419.0*x^3-416.0*x^2+220.0*x-48.0)
(c2) allroots(d1);
(d2)      [4.0, 3.0, 2.0, 2.0, 1.0, 1.0, 1.0]
(c3) allroots(x^2+1);
(d3)      [1.0*i, -1.0*i]
```

## 3.5 Calculus

### 3.5.1 DIFF (*expr*) DIFF (*expr*, *var*)

Differentiate *expr* with respect to variable *var*. If *var* is not specified, the main variable of *expr* is used. If *expr* contains functions this function is treated as

variable (see c4,c5).

```
(c1) diff(a*x^3);
(d1)      3.0*a*x^2
(c2) diff(a*x^3, a);
(d2)      x^3
(c3) diff(3*sqrt(exp(x)+2), x);
(d3)      1.5*exp(x)/sqrt(exp(x)+2.0)
(c4) diff(sin(x));
(d4)      1.0
(c5) diff(sin(x), x);
(d5)      cos(x)
```

### 3.5.2 INTEGRATE (*expr*) INTEGRATE (*expr*, *var*)

Integrate *expr* using variable *var*. Strategy:

- If *expr* is rational (i.e. quotient of two polynomials, whose coefficients do not depend on *var*) we use the standard approach: Separate a polynomial part, then separate a square free part using Horowitz' [12] method, and finally integrate the rest using partial fractions. The final terms are collected to avoid complex expressions, but this step may not always work. Try *TRIGRAT* on the result in this case. (*Example c5*).
- Expressions of type  $g(f(x)) \cdot f'(x)$  and  $\frac{f'(x)}{f(x)}$  are sometimes detected and properly treated. (*Examples c6, c7*)
- The standard functions and substitutions of type  $(a \cdot x + b)$  are always identified.
- 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* and a few more. (*Examples c8, c9*)
- 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. (*Example c11*)
- $\sqrt{ax^2 + bx + c}$  is handled. (*Example c10*)



- If all fails, you should use *ROMBERG*.

The variable *var* should be specified whenever a function  $f(x)$  is used, otherwise this function will be the variable (see example (c2)). The results can be quickly proven using *DIFF*, see (c12).

```
(c1) integrate(x^2+x-3, x);
(d1)      (0.3333333333333333*x^3+0.5*x^2-3.0*x)
(c2) integrate(sin(x));
(d2)      0.5*sin(x)^2
(c3) integrate(sin(x), x);
(d3)      -cos(x)
(c4) expand((x^3+2*x^2-x+1)/((x+i)*(x-i)*(x+3)));
(d4)      (x^3+2.0*x^2-x+1.0)/(x^3+3.0*x^2+x+3.0)
(c5) integrate(d4);
(d5)      (-1/4*log(x^2+1.0)+(-1/2*log(x+3.0)+
      (-1/2*atan(x)+(x-1/4*i*pi))))
(c6) integrate(x*exp(-2*x^2), x);
(d6)      -0.25*exp(-2.0*x^2)
(c7) integrate(exp(x)/(1+exp(x)), x);
(d7)      log(exp(x)+1.0)
(c8) integrate(x^3*exp(-2*x), x);
(d8)      (-0.5*x^3+1.5*x^2+3.0*x+3.0)*exp(-2.0*x)
(c9) integrate(x^2*log(x), x);
(d9)      (0.3333333333333333*x^3*log(x)-
      0.1111111111111111*x^3)
(c10) integrate(sqrt(x^2-1), x);
(d10)      (0.5*x*sqrt(x^2-1.0)-
      0.5*log(2.0*sqrt(x^2-1.0)+2.0*x))
(c11) integrate(sin(x)*cos(3*x)^2, x);
(d11)      (-0.03571428571428571*cos(7.0*x)+
      (0.05*cos(5.0*x)-0.5*cos(x)))
(c12) diff(d5, x);
(d12)      (x^3+2*x^2-x+1)/(x^3+3*x^2+x+3)
```

### 3.5.3 ROMBERG (*exp,var,ll,ul*)

Numerically integrate *exp* with variable *var* from *ll* to *ul* using the Romberg [13] algorithm. The maximum number of iterations is set by the variable ROMBERGIT

(default: 11) and the relative tolerance by ROMBERGTOL (default:  $10^{-4}$ ). Example c6 is demanding!

```
(c1)  romberg(exp(-x^2), x, 0, 5);
(d1)      0.8862259970904623
(c2)  rombergtol;
(d2)      1.0E-4
(c3)  float(romberg(sqrt(1-x^2), x, 0, 1)-pi/4);
(d3)      -2.3674555102326522E-5
(c4)  rombergtol:1e-10;
(d4)      1.0E-10
(c5)  rombergit:100;
(d5)      100.0
(c6)  float(romberg(sqrt(1-x^2), x, 0, 1)-pi/4);
(d6)      -3.188449504420987E-11
```

### 3.5.4 TAYLOR (*exp, var, pt, pow*)

Taylor series of *expr* with variable *var* at point *pt* up to power *pow*.

```
(c1)  taylor(log(x), x, 1, 1);
(d1)      (x-1.0)
(c2)  rat(taylor(exp(x), x, 0, 4));
(d2)      (1/24*x^4+1/6*x^3+1/2*x^2+x+1)
(c3)  float(taylor(x^3*sin(2*x+pi/4), x, pi/8, 2));
(d3)      (1.0569789768137512*x^2-
           0.3675116408381707*x+
           0.04188117486476518)
```

## 3.6 Input and Output

### 3.6.1 SAVE (*filename, var1, var2, ...*)

Save content of variables *var1, ...* to file. Specifying `all` saves all variables. No function definitions are saved. If *filename* contains spaces or Jasymca operators, it should be quoted.

### 3.6.2 LOADFILE (*filename*)

Reads previously saved variable definitions. The file contains standard Jasymca commands and may be edited using any text editor. The J2ME variant of Jasymca has its own builtin text editor for this purpose. Check the builtin filebrowser about available directories on your device. If *filename* contains spaces or Jasymca operators, it should be quoted.

```
(c4) c:300000;
(d4)      300000.0
(c5) save("vfs/varc",c);
(d5)      Wrote variables to vfs/varc
```

In a new session:

```
(c1) loadfile("vfs/varc");
(d1)      Loaded Variables from vfs/varc
(c2) c;
(d2)      300000.0
(c3) save("vfs/varc",all);
(d3)      Wrote variables to vfs/varc
```

## 4 Building from sources

The files in the symbjava-directory of the distribution [7] contain the basic sources of jasymca. The J2SE and J2ME versions of these sources differ by only one file: `Jasymca.java`. Uncomment either the section labeled J2SE or the one labeled J2ME.

To build the J2SE version, you need a Javacompiler (e.g. `javac`) and, after editing the file `Jasymca.java` as described, execute `javac *.java` in the directory `symbjava`.

The J2ME-version requires additional helper classes located in the `kjava`, `jsystem` and `jranner` directories as well as some icon-resources. Get and install the free J2ME development toolkit from Sun [1] and move the whole `Jasymca` directory to the applications directory of the installation. Then open and build the application using the appropriate menu-options.

## 5 Design considerations and related work

The goal of this project is a useful tool on the one hand, and a concept which enables the user to easily switch to more powerful tools on workstations.

Sun's Java Microedition [1] (J2ME) is widely available in mobile devices. Although it is not the best platform for doing math, both feature and performance-wise, the availability makes it the prime candidate for this project. For the specific implementation, I was looking for related projects which could be ported to this platform. This approach worked for my Matlab/Octave clone *FnattLabME* [7], which is a port of August Berings Java program *FnattLab* [14] to mobile devices. I considered and tested the following candidates:

### 5.1 Jscl-meditor

Meditor [15] is a symbolic math package written in Java by Raphael Jollyin and Elefterios Stamatogiannakis. It consists of an executable and a library. The distribution includes a Midlet version, which crashes the current Sun Midlet emulator (it tries to create classes in the `java.lang` package).

Unfortunately, the program is not very powerful in handling hyperbolic and trigonometric functions, both in solving equations as well as term manipulations and calculus. E.g. the trigonometric identity (see section *TRIGRAT*)

$$\sin^2(x) + \sin^2\left(x + \frac{2\pi}{3}\right) + \sin^2\left(x + \frac{4\pi}{3}\right) = \frac{3}{2}$$

is not found. Rational functions are integrated, but the results are unusably complicated (btw: a problem shared by many pocket calculators), e.g. the rational example from the *INTEGRATE* section:

$$\int \frac{x^3 + 2x^2 - x + 1}{x^3 + 3x^2 + x + 3} dx$$

leads to 4 pages of unreadable formulas, which the simplifier is not able to handle. Integration of other functions almost always fail, e.g. all non-trivial examples from the same section. The same applies to equation solving: The solutions to polynomial equations are often uncomfortable expressions, and the program fails for most other functions.

## 5.2 Yacas

Yacas [17] is a free symbolic calculator written in C++. The distribution includes a port to Java (J2SE): JYacas. It is too large to be directly usable on mobile devices and I tried to create a stripped version. I managed to reduce binary size to less than 0,3MByte, and reduced memory usage by profiling to less than 1/10 its original value. It finally ran in Sun's J2ME emulator, but was painfully slow, and still crashed any real J2ME device. This slowness is due to the basic design of JYacas as interpreter of a sophisticated set of scripts. I gave up on this approach and started to build Jasymca from scratch.

## 5.3 Jasymca

The basic concepts of computer algebra systems are described in "Computer Algebra", by J.H. Davenport [16] et al., and some programming resources used in Jasymca are described in "Paradigms of Artificial Intelligence Programming: Case Studies in Common Lisp", by Peter Norvig [18]. The basic structure and the syntax of most commands follow *GNU-Maxima* [2]. We use a canonical representation of algebraic objects, which may be one of Rational, Polynomial, Numbers, Vectors or Matrices. Vector and Matrix operations are only rudimentary at this time. A Polynomial may have simple variables like  $x$ ,  $y$ ,  $z$ , or function variables like  $\sin(x)$  or  $\log(x)$ . There is a special flavor of exponential polynomials, which represent canonical combinations of exponential functions. Numbers are complex and also come in two variants: floating point (inexact) and rational (exact) which are pairs of big integers. Big integers are implemented as `java.math.BigInteger` i.e. almost arbitrary in size. Many (not all) commands of Jasymca can be equally and with similar results applied to *GNU-Maxima*, but not vice versa.

Size of Jasymca is now 180kByte, 300kByte is an arbitrary upper limit for future versions, which will enable its use for most current cellphones. Startup is quite fast which is achieved by avoiding costly initialisations. Most operators and functions are instantiated just in time. With these features in mind, I am planning a few extensions like support for program flow (*if*, *do*) and some more commands. However, I will not add graphing capabilities or extensive matrix computation, which is both available in my package *FnattLabME* [7].

## 6 License

Jasymca is distributed under GNU-license, see the included license text. Some source files are taken from other open projects which have their own license:

- The files BigInteger.java, Random.java, and MPN.java are slightly modified copies from the GNU-Classpath [20] project. This project is also distributed under GNU-license.
- The file JMath.java is a pure Java implementation of the java.lang.Math class. It is derived from open sources and copyright Visual Numerics Inc [19]. See the source file for details of its license.

## References

- [1] <http://java.sun.com/j2me/>
- [2] <http://maxima.sourceforge.net/>
- [3] [www.mathworks.com](http://www.mathworks.com)
- [4] [www.octave.org/](http://www.octave.org/)
- [5] [www.maplesoft.com/](http://www.maplesoft.com/)
- [6] [www.wolfram.com/](http://www.wolfram.com/)
- [7] [www.hs-furtwangen.de/ dersch](http://www.hs-furtwangen.de/~dersch)
- [8] [www.forum.nokia.com/main/0,,150,00.html?matrixType=midp2](http://www.forum.nokia.com/main/0,,150,00.html?matrixType=midp2)
- [9] [www.mwiacek.com/gsm/soft/gammu.html](http://www.mwiacek.com/gsm/soft/gammu.html)
- [10] [www.palm.com/us/support/jvm/](http://www.palm.com/us/support/jvm/)
- [11] [de.wikipedia.org/wiki/Bairstow-Verfahren](http://de.wikipedia.org/wiki/Bairstow-Verfahren)
- [12] [mathworld.wolfram.com/HorowitzReduction.html](http://mathworld.wolfram.com/HorowitzReduction.html)
- [13] [de.wikipedia.org/wiki/Romberg-Integration](http://de.wikipedia.org/wiki/Romberg-Integration)
- [14] <http://reactos.ccp14.ac.uk/projects/fnattlab/>

- [15] <http://jscl-meditor.sourceforge.net>
- [16] Computer Algebra, by J.H. Davenport (Bath), Y.Siret (Grenoble) and E.Tournier (Grenoble) Academic Press (1991).
- [17] <http://yacas.sourceforge.net>
- [18] Paradigms of Artificial Intelligence Programming: Case Studies in Common Lisp, Peter Norvig, Morgan Kaufmann, 1992.
- [19] [www.vni.com/](http://www.vni.com/)
- [20] [www.gnu.org/software/classpath/](http://www.gnu.org/software/classpath/)