A short introduction to Yacas and GUYacas

Søren Højsgaard
Statistics and Decision Theory Research Unit,
Danish Institute of Agricultural Sciences,
Research Center Foulum, DK–8830 Tjele, Denmark

July 10, 2006

Contents

1 Introduction

GUYacas is a graphical user interface (GUI) to Yacas ("Yet another computer algebra system") on Windows platforms. GUYacas is available from the GUYacas homepage http://gbi.agrsci.dk/ shd/Misc/GUYacas where also installation instructions can be found.

Yacas is developed by Ayal Pinkhuis (who is also the maintainer) and others. Yacas is available (for various platforms) at yacas.sourceforge.org.

There is a comprehensive documentation (300+ pages) of Yacas (also available at yacas.sourceforge.org) and the documentation contains many examples.

The examples given here are largely taken from the Yacas documentation (especially from the introductory chapter) but organised differently.

2 A sample session

A few sample calculations are given below. Note that lines starting with a hash (#) are regarded as comments by GUYacas and are not executed.

```
# Assign value to a variable:
In> n:=10
Out>10;
# Make numerical calculation:
In> n^2+25
Out>125;
# Differentiate:
In> D(x)Sin(x)
Out>Cos(x);
# Expand polynomium:
In> Expand((1+x)^3);
Out>x^3+3*x^2+3*x+1;
# Integrate:
In> Integrate(x,a,b)Sin(x);
Out>Cos(a)-Cos(b);
# Taylor expansion
In> texp := Taylor(x,0,3) Exp(x)
Out>x+x^2/2+x^3/6+1;
# Show the result in nice form:
In> PrettyForm(texp);
Out>
     2    3
    x    x
x + -- + -- + 1
    2    6


# Show the result in Latex format
In> TexForm(texp)
Out>$x + \frac{x ^{2}}{2}  + \frac{x ^{3}}{6}  + 1$
```

3 Simple Yacas calculations

3.1 Simple calculations

In Yacas you can make simple calculations. The assign operator `:=` provides a handle on objects (observe that Yacas is case–sensitive):

```
In> (10 + 2) * 5 + 7^7
Out>823603;
In> 1/14+5/21*(30-(1+1/2)*5^2);
Out>(-12)/7;
```

3.2 Setting and clearing a variable

The function Set() and the operator := can both be used to assign values to global variables.

```
In> n := (10 + 2) * 5
Out>60;
In> n := n+n
Out>120;
In> Set(z, Cos(a))
Out>True;
In> z+z
Out>2*Cos(a);
```

To clear a variable binding execute Clear():

```
In> n
Out>120;
In> Clear(n)
Out>True;
In> n
Out>n;
```

Currently there is no difference between assigning variables using Set() or using the operator :=. The latter can however also assign lists and define functions.

3.3 Symbolic and numerical evaluations, precision

Evaluations are generally exact:

```
In> Exp(0)
Out>1;
In> Exp(1)
Out>Exp(1);
In> Sin(Pi/4)
Out>Sqrt(1/2);
In> 355/113
Out>355/113;
```

To obtain a numerical evaluation (approximation), the N() function can be used:

```
In> N(Exp(1))
Out>2.7182818285;
In> N(Sin(Pi/4))
Out>0.7071067811;
In> N(355/113)
Out>3.1415929203;
```

The N() function has an optional second argument, the required precision:

```
In>  N(355/133,20)
Out>2.66917293233082706767;
```

The command Precision(n) can be used to specify that all floating point numbers should have a fixed precision of n digits:

```
In>  Precision(5);
Out>True;
In> N(355/113)
Out>3.14159;
```

3.4 Rational numbers

Rational numbers will stay rational as long as the numerator and denominator are integers:

```
In> 55/10
Out>11/2;
```

3.5 Complex numbers and the imaginary unit

The imaginary unit i is denoted I and complex numbers can be entered as either expressions involving I or explicitly Complex(a,b) for a+ib.

```
In> I^2
Out>-1;
In> 7+3*I
Out>Complex(7,3);
In> Conjugate(%)
Out>Complex(7,-3);
In> Exp(3*I)
Out>Complex(Cos(3),Sin(3));
```

3.6 Symbolic calculation

Some exact manipulations :

```
In>  1/14+5/21*(30-(1+1/2)*5^2);
Out>(-12)/7;
In>  0+x;
Out>x;
In>  x+1*y;
Out>x+y;
In>  Sin(ArcSin(alpha))+Tan(ArcTan(beta));
Out>alpha+beta;
```

3.7 Recall the most recent line – the % operator

The operator % automatically recalls the result from the previous line.

```
In> (1+x)^3
Out>(x+1)^3;
In> %
Out>(x+1)^3;
In> z:= %
Out>(x+1)^3;
```

3.8 Pi (π)

There is a built in variable Pi for π. An approximate numerical value is obtained with N(Pi). Yacas knows some simplification rules using Pi (especially with trigonometric functions).

```
In> Pi
Out>Pi;
In> N(Pi)
Out>3.141592653589793;
In> Sin(Pi/4)
Out>Sqrt(1/2);
```

3.9 PrettyForm and TeXForm

Results can be output on the screen in a more readable form with the function PrettyForm():

```
In> xxx:= (1+x)^2+k^3
Out>(x+1)^2+k^3;
In> PrettyForm(xxx);
Out>
         2    3
( x + 1 )  + k
```

The output can be exported to TeX with TeXForm(), e.g.

```
In> TeXForm(xxx)
Out>"$\left( x + 1\right)  ^{2} + k ^{3}$";
```

4 Commands

4.1 Factorial

```
In>  40!
Out>815915283247897734345611269596115894272000000000;
```

4.2 Taylor expansions

Exand Exp(x) in three terms around 0 and a:

```
In> Taylor(x,0,3) Exp(x)
Out>x+x^2/2+x^3/6+1;
In> Taylor(x,a,3) Exp(x)
Out>Exp(a)+Exp(a)*(x-a)+((x-a)^2*Exp(a))/2+((x-a)^3*Exp(a))/6;
```

The InverseTaylor() function builds the Taylor series expansion of the inverse of an expression. For example, the Taylor expansion in two terms of the inverse of Exp(x) around x=0 (which is the Taylor expansion of Ln(y) around y=1):

```
In> InverseTaylor(x,0,2)Exp(x);
Out>x-1-(x-1)^2/2;
In> Taylor(y,1,2)Ln(y);
Out>y-1-(y-1)^2/2;
```

4.3 Solving equations

Solve equations symbolically with:

```
In>  Solve(x/(1+x) == a, x);
Out>{x==a/(1-a)};
In>  Solve(x^2+x == 0, x);
Out>{x==0,x==(-1)};
```

(Note the use of the == operator, which does not evaluate to anything, to denote an "equation" object.) Currently Solve is rather limited.

To solve an equation (in one variable) like Sin(x)-Exp(x)=0 numerically taking 0.5 as initial guess and an accuracy of 0.0001 do:

```
In> Newton(Sin(x)-Exp(x),x, 0.5, 0.0001)
Out>-3.1830630118;
```

4.4 Expanding polynomials

```
In>   Expand((1+x)^3);
Out>x^3+3*x^2+3*x+1;
```

4.5 Simplifying an expression

The function Simplify() attempts to reduce an expression to a simpler form.

```
In>  (x+y)^3-(x-y)^3
Out>(x+y)^3-(x-y)^3;
In>  Simplify(%)
Out>6*x^2*y+2*y^3;
```

4.6 Analytical derivatives

Analytic derivatives of functions can be evaluated:

```
In>  D(x) Sin(x);
Out>Cos(x);
In>  D(x) D(x) Sin(x);
Out>-Sin(x);
```

The D function also accepts an argument specifying how often the derivative has to be taken, e.g:

```
In>  D(x,2)Sin(x)
Out>-Sin(x);
```

4.7 Integration

```
In>  Integrate(x,a,b)Sin(x);
Out>Cos(a)-Cos(b);
In> Integrate(x,a,b)Ln(x)+x;
Out>b*Ln(b)-b+b^2/2-(a*Ln(a)-a+a^2/2);
In> Integrate(x)1/(x^2-1);
Out>Ln(2*(x-1))/2-Ln(2*(x+1))/2;
In> Integrate(x)Sin(a*x)^2*Cos(b*x);
Out>((2*Sin(b*x))/b-(Sin((-2)*x*a-b*x)/((-2)*a-b)+Sin((-2)*x*a+b*x)/((-2)*
    a+b)))/4;
```

4.8 Limits

```
In>  Limit(x,0)Sin(x)/x;
Out>1;
In> Limit(n,Infinity)(1+(1/n))^n
Out>Exp(1);
In> Limit(h,0) (Sin(x+h)-Sin(x))/h;
Out>Cos(x);
```

4.9 Variable substitution

```
In>  Subst(x,Cos(a))x+x;
Out>2*Cos(a);
```

4.10 Solving ordinary differential equations

```
In> OdeSolve(y''==4*y);
Out>C210*Exp(2*x)+C214*Exp((-2)*x);
In> OdeSolve(y'==8*y)
Out>C252*Exp(8*x);
```

5 Strings and lists

Strings are simply sequences of characters enclosed by double quotes, for example:

```
In> "this is a string with \"quotes\" in it"
Out>"this is a string with "quotes" in it";
```

Lists are ordered groups of items. Yacas represents lists by putting the objects between braces and separating them with commas. Items in a list can be accessed through the [ ] operator, for example:

```
In> uu:={a,b,c,d,e,f};
Out>{a,b,c,d,e,f};
In> uu[2];
Out>b;
In> uu[2 .. 4];
Out>{b,c,d};
```

Note the "range" expression (the spaces around the .. operator are necessary, or else the parser will not be able to distinguish it from a part of a number):

```
In> 2 .. 6
Out>{2,3,4,5,6};
```

In Yacas, vectors are represented as lists and matrices as lists of lists. Any Yacas expression can be converted to a list.

Another use of lists is the associative list, sometimes called a hash table, which is implemented in Yacas simply as a list of key-value pairs. Keys must be strings and values may be any objects. Associative lists can also work as mini-databases. As an example:

```
In> u:={};
Out>{};
In> u["name"]:="Isaia";
Out>True;
In> u["occupation"]:="prophet";
Out>True;
In> u["is alive"]:=False;
Out>True;
In> u["name"];
Out>"Isaia";
```

The list u now contains three sublists:

```
In>  u;
Out>{{"is alive",False},{"occupation","prophet"},{"name","Isaia"}};
```

Lists evaluate their arguments;

```
In> {1+2,3}
Out>{3,3};
```

Assignment of multiple variables is also possible using lists:

```
In> {x,y}:={2!,3!}
Out>{2,6};
```

More examples:

```
In>  m:={a,b,c};
Out>{a,b,c};
In>  Length(m);
Out>3;
In>  Reverse(m);
Out>{c,b,a};
In>  Concat(m,m);
Out>{a,b,c,a,b,c};
In>  m[1]:="blah blah";
Out>True;
In>  m;
Out>{"blah blah",b,c};
In>  Nth(m,2);
Out>b;
```

Lists can also be used as function arguments when a variable number of arguments are expected.

6 Matrices

```
In> E4:={ {u1,u1,0},{u1,0,u2},{0,u2,0} };
Out>{{u1,u1,0},{u1,0,u2},{0,u2,0}};
In> PrettyForm(E4);
Out>
/                        \
| ( u1 ) ( u1 ) ( 0 )    |
|                        |
| ( u1 ) ( 0 )  ( u2 )   |
|                        |
| ( 0 )  ( u2 ) ( 0 )    |
\                        /
```

Inverse:

```
In> E4i:=Inverse(E4)
Out>{{u2^2/(u1*u2^2),0,(-u1*u2)/(u1*u2^2)},{0,0,(u1*u2)/(u1*u2^2)},{(-u1*u
    2)/(u1*u2^2),(u1*u2)/(u1*u2^2),u1^2/(u1*u2^2)}};
In> Simplify(E4i)
Out>{{1/u1,0,(-1)/u2},{0,0,1/u2},{(-1)/u2,1/u2,u1/u2^2}};
In> PrettyForm(Simplify(E4i))
Out>
/                         \
| / 1  \ ( 0 )  / -1 \    |
| | -- |        | -- |    |
| \ u1 /        \ u2 /    |
```

```
|                         |
| ( 0 )  ( 0 )  / 1  \    |
|               | -- |    |
|               \ u2 /    |
|                         |
| / -1 \ / 1  \ / u1  \   |
| | -- | | -- | | --- |   |
| \ u2 / \ u2 / |   2 |   |
|               \ u2  /   |
\                         /
```

Determinant:

```
In> Determinant(E4)
Out>(-(u1*u2)^2)/u1;
In> Determinant(E4i)
Out>Undefined;
In> E4i:=Inverse(E4)
Out>{{(u2*(u2-(u1*u2)/u1)+(u1*u2^2)/u1)/(u1*u2*(u2-(u1*u2)/u1)+(u1*u2)^2/u
    1),(u1*u2^2)/((u1*u2*(u2-(u1*u2)/u1)+(u1*u2)^2/u1)*u1),(-u1*u2)/(u1*u2
    *(u2-(u1*u2)/u1)+(u1*u2)^2/u1)},{0,(-u1*u2^2)/((u1*u2*(u2-(u1*u2)/u1)+
    (u1*u2)^2/u1)*u1),(u1*u2)/(u1*u2*(u2-(u1*u2)/u1)+(u1*u2)^2/u1)},{0,(u1
    *(u2-(u1*u2)/u1))/(u1*u2*(u2-(u1*u2)/u1)+(u1*u2)^2/u1),u1^2/(u1*u2*(u2
    -(u1*u2)/u1)+(u1*u2)^2/u1)}};
In> Simplify(E4i)
Out>{{1/u1,1/u1,(-1)/u2},{0,(-1)/u1,1/u2},{0,0,u1/u2^2}};
In> Simplify(Determinant(E4i))
Out>(-1)/(u1*u2^2);
```

Note that there are two issues here: The two calculations of the inverse E4i look different – but they reduce to the same after using the Simplify() function. Secondly, there is a problem in calculating the determinant for the first version of E4i while there is not for the second.

7 Functions

The := operator can be used to define functions. One and the same function name such as f may be used by different functions if they take different numbers of arguments:

```
In>  f(x):=x^2;
Out>True;
In>  f(x,y):=x*y;
Out>True;
In>  f(3)+f(3,2);
Out>15;
```

Functions may return values of any type, or may even return values of different types at different times.