Clean GP programming

with a view to GP2C

B. Allombert

IMB
CNRS/Université de Bordeaux

14/01/2015

New GP functions

```
? expm1(1.E-10)
%1 = 1.0000000000500000000016666666667083333E-10
? powers(x,3)
%2 = [1, x, x^2, x^3]
? fromdigits([1,2,3])
%3 = 123
? qfbredsl2(Qfb(1,7,19))
%4 = [Qfb(1,1,7),[1,-3;0,1]]
? nfcompositum(nfinit(a^2+1),x^2+a,x^2+a+1)
%5 = [x^4+Mod(4*a+2,a^2+1)*x^2+1]
```

New GP functions for elliptic curves

```
? ellissupersingular(ellinit([1,0],19))
%6 = 1
? ellisdivisible(ellinit([1,1]),[72,611],3)
%7 = 1
? ellxn(ellinit([1,1]),3)
%8 = [x^4-2*x^2-8*x+1,4*x^3+4*x+4]
```

Function definition

Define your function as follow:

```
fun(x,y,z=0,t=1)=
{
  my(a,b,c);
  ...;
  a;
}
addhelp(fun,"computes the ... of x and y...");
```

- Put the braces on the line after the = sign.
- Denote all optional arguments with `z=`.
- Declare all local variables with `my()`.
- Avoid useless `return` at the end.
- Add a trailing semicolon.

strictargs

Normally, the arguments of user-defined GP function are all optionnals. Using `default(strictargs,1)`, the arguments are mandatory unless an explicit default value is provided.

strictargs

```
default(strictargs,0);
fun(a,b=1)=[a,b];
fun(2)
fun()
default(strictargs,1);
fun(a,b=1)=[a,b];
fun(2)
fun()
  ***   missing mandatory argument 'a' in user
  ***   function.
```

Loops

```
my(s);
forprime(p=3,,
  if(p%4==1,s++,
             s--);
  if(s==1,
     return(p))
);
```

- Indent your code.
- Do not declare automatic loop variable (`p` here).

Polynomials

```
trans(P)=subst(P,x,x+1)
trans1(P)=subst(P,'x,'x+1)
trans2(P)=my(v=variable(P));subst(P,v,v+1)
trans3(P,x=variable(P))=subst(P,x,x+1)
```

- Write indeterminate using `'x` instead of `x`.
- Use `variable()` to query the input variable
- Better: let caller specify the variable.

precision

```
nome(x)=exp(2*I*Pi*x)
nome1(x)=localprec(precision(x));exp(2*I*Pi*x)
nome2(x,prec=precision(x))=localprec(prec);\
                exp(2*I*Pi*x)
```

- Return results with the same accuracy as the input, or as specified by the caller.
- Use `localprec()` when using constants and functions that depend on `realprecision`.

Trapping error

You can use iferr() to trap errors. Be careful to trap only the errors you expect:

```
iferr(tan(Pi/2),E,Vec(E))
mytan(x)=iferr(tan(x),E,oo)
mytan(x)=iferr(tan(x),E,oo, #Vec(E)==5 &&
  Vec(E)[1..4]==
   ["e_DOMAIN","tan","argument","=","Pi/2 + kPi"])
```

For GP2C

- Avoid global variables.
- Otherwise declare them using global().
- Run `gp2c-run -WLS` on your program to check for problems.
- Use `\g1` under GP when reading your script to get warnings about copy problems.

For GP2C

Create the file `example.gp`

```
rho(n)=
{
  my(x,z);
  x=2; y=5;
  while(gcd(y-x,n)==1,
       x=(x^2+1)%n;
       y=(y^2+1)%n; y=(y^2+1)%n
       );
  gcd(n,y-x)
}
```

For GP2C

```
gp2c-run -WL example.gp
Warning:example.gp:4: variable undeclared
y
example.gp: In function ‘rho’:
example.gp:3:16: warning: unused variable ‘z’
```

This reports two warnings.