Some new GP features

A tutorial

B. Allombert

IMB
CNRS/Université de Bordeaux

13/01/2015

parisizemax

It is now possible to configure GP to increase the stack dynamically to avoid stack overflow. Set in your `.gprc`

```
parisizemax = "4G"
```

(set `parisizemax` to the maximum stack size you can afford.) GP will dynamically increase the stack until it reaches the limit, and reset the stack size to the default when the computation ends.

```
? default(parisizemax,"32M")
? bnfinit(x^8+10001).no
  *** bnfinit: Warning: increasing stack size
      to 16000000.
%19 = 81920
```

parisizemax

In this mode, `allocatemem()` can be used to increase the stacksize manually, even in a break loop.

```
? #
? setrand(1);quadclassunit(2^128+1);
? allocatemem(32*10^6);setrand(1);\
  quadclassunit(2^128+1);
```

This can be useful in conjunction with `debugmem`.
`threadsizemax` is also available for thread stacks.

localprec

It is possible to change the precision locally by using `localprec`:

```
printpi(n)=localprec(n);printf("%.*g",n,Pi)
? printpi(30)
3.14159265358979323846264338328
? printpi(40)
3.141592653589793238462643383279502884197
? q(x)=localprec(precision(x));exp(2*I*Pi*x)
```

localprec

```
? q1(x)=exp(2*I*Pi*x)
? q2(x)=localprec(precision(x));exp(2*I*Pi*x)
? q1(precision(1.,1000)/3)^3
%7 = 1.00000000+1.89735380E-19*I
? q2(precision(1.,1000)/3)^3
%8 = 1.00000000+1.48653257E-1001*I
```

varhigher

The functions `varhigher()` and `varlower()` allow to create private polynomial indeterminates with prescribed priority.
Please ask Karim.

fold

The function `fold` reduce a list using a binary operator `fold(f,[a,b,c,d])` returns $f(f(f(a,b),c),d)$.

```
? fold((x,y)->x*y, [1,2,3,4])
%1 = 24
? fold((x,y)->[x,y], [1,2,3,4])
%2 = [[[1, 2], 3], 4]
? fold((x,y)->(x+y)/(1-x*y),[1..5])
%3 = -9/19
? bestappr(tan(sum(i=1,5,atan(i))))
%4 = -9/19
```

self

The function `self()` returns the calling function or closure. This is useful for creating recursive anonymous functions:

```
? apply(n->if(n==0,1,n*self()(n-1)),[1..5])
%1 = [1,2,6,24,120]
```

This can be useful for parallel GP since it does not use global variables:

```
? parapply(n->if(n==0,1,n*self()(n-1)),[1..5])
%1 = [1,2,6,24,120]
```

logint

The function `logint(x,b)` computes the integral part of the logarithm of *x* in base *b*.

```
? logint(1000, 2)
%1 = 9
? logint(1000, 2, &z)
%2 = 9
? z
%3 = 512
```

Variadic GP function

The syntax `f(a,v[..])=expr` allows to define a GP function that takes a variable number of arguments. They are passed through the vector `v`.

```
myprintsep(s,v[..])=
{
  if(#v>0, print1(v[1]);
  for(i=2,#v, print1(s,v[i])));
  print();
}
? myprintsep(":",1,2,3,4)
1:2:3:4
```

Infinite elements

There is now an object `+oo` for positive infinite and `-oo` for negative infinite, such that `-oo<x<+oo` holds for all finite scalar objects *x*.
Usage:

```
? intnum(x = 1,+oo, 1/x^2)
%1 = 1.000000000000000000000000000000
? valuation(0,x)
%2 = +oo
? poldegree(0)
%3 = -oo
```

Isogenies between elliptic curves

The function `ellisogeny` allow to compute the image of an isogeny from its kernel.

```
? E=ellinit([0,1]);
? elltors(e)
%2=[6,[6],[[2,3]]]
? ellisogeny(E,[2,3],1)\\Weierstrassmodel for E/<P>
%3=[0,0,0,-135,-594]
? ellisogeny(E,[-1,0])
%4=[[0,0,0,-15,22],[x^3+2*x^2+4*x+3,y*x^3+3*y*x^2-2
```

`ellisogenyapply` allows to apply the isogeny on a point.

Modular polynomials

Two new functions

- `polmodular(p)`: returns the canonical modular polyomial of level *p*.
- `polclass(D)`: return the Hilbert class polynomial for the discriminant $D < 0$.

See Hamish talk tomorrow.

Elliptic curves over finite fields

PARI is able to count points on elliptic curves over finite fields in polynomial time for all characteristics. The algorithms used are (for $K = \mathbb{F}_{p^n}$).

- For $p = 2$: Satoh canonical lift, Mestre-Harley AGM algorithm
- For $p = 3, 5, 7, 13$: Satoh canonical lift, Harley-Kohel algorithm
- For $p = 11$ or $n > 7p/10$: Satoh canonical lift, generalised Harley algorithm
- If $p/2 < n < 7p/10$: Kedlaya algorithm
- If $n < p/2$: Schoof-Elkies-Atkin, Lercier-Joux variant
- For $n = 1$: Schoof-Elkies-Atkin algorithm

Elliptic curves over number fields

It is now possible to define elliptic curves over finite fields:

```
? K=bnfinit(a^2+1);
? E=ellinit([1,a],K);
? elltors(E)
%3 = [1,[],[]]
? pr=idealprimedec(K,17)[1];
? ellcard(ellinit(E,pr))
%5 = 14
```

hyperellcharpoly

This function allows to compute the ζ function associated to an hyperelliptic curve over a finite field of small characteristic.

```
? hyperellcharpoly((x^5+13*x+7)*Mod(1,17))
%1 = x^4-3*x^3+16*x^2-51*x+289
```

qfsolve

GP includes a port of the program qfsolve by Denis Simon to solve binary quadratic forms.
Find a solution of $x^2 + 3y^2 - 21z^2 = 0$:

```
? qfsolve([1,0,0;0,3,0;0,0,-21])
%1 = [3,2,1]~
```

For dimension 3: Find parametric solution of $x^2 + 3y^2 - 21z^2 = 0$:

```
? M = qfparam([1,0,0;0,3,0;0,0,-21],[3,2,1]~)
%2 = [-3,-12,9;2,-6,-6;1,0,3]
? v = y^2 * M*[1,x/y,(x/y)^2]~
%3 = [9*x^2-12*y*x-3*y^2,-6*x^2-6*y*x+2*y^2,3*x^2+y
? v[1]^2+3*v[2]^2-21*v[3]^2
%4 = 0
```

ellformal

New functions for elliptic curves:
ellformalw, ellformalpoint, ellformaldifferential, ellformallog, ellformalexp, ellnonsingularmultiple, ellpadicheight, ellpadicheightmatrix, ellpadics2, ellpadiclog, ellpadicL
Please ask Karim !

msinit

New functions for modular symbols:
mscuspidal, mseisenstein, msnew, mssplit, msqexpansion, mshecke, ellmsinit, msatkinlehner, msstar, mseval, mspathgens, mspathlog, msissymbol
Please ask Karim !

nfsplitting

This function computes the spliting field of the field defined by a polynomial.

```
? nfsplitting(x^5-2)
%1 = x^20+2500*x^10+50000
? poldegree(nfsplitting(x^5+x+3))
%2 = 120
? poldegree(nfsplitting(x^6+24*x-20))
%3 = 360
```

Miscellaneous

```
? 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]]
ellissupersingular(ellinit([1,0],19))
%5 = 1
? ellisdivisible(ellinit([1,1]),[72,611],3)
%6 = 1
? ellxn(ellinit([1,1]),3)
%7 = [x^4-2*x^2-8*x+1,4*x^3+4*x+4]
```