Some new GP features

A tutorial

B. Allombert

IMB
CNRS/Université de Bordeaux

9/01/2017

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement N° 676541

Javascript port via emscripten

Thanks to Bernard Parisse it is now possible to build PARI/GP to JavaScript for use in web browser `http://pari.math.u-bordeaux.fr/gp.html`.

characters of Abelian group

New functions: general characters: `charconj` `chardiv` `chareval` `charmul` `charorder`
Dirichlet characters: `znchareval` `zncharinduce` `zncharisodd` `znchartokronecker` `znconreychar` `znconreyconductor` `znconreyexp` `znconreylog`
See Karim tutorial

logarithmic class group

New functions `bnflog` `bnflogef` `bnflogdegree` `nfislocalpower` `rnfislocalcyclo` for logarithmic class group, see Jose talk.

modular symbols

New functions for modular symbols

`msfromcusp` `msfromhecke` `msgetlevel` `msgetsign` `msgetweight`

For overconvergent modular symbols

`mstooms` `msomseval`

For p-adic L-functions

`mspadicinit` `mspadicL` `mspadicmoments` `mspadicseries`

See Bernadette talk.

parallelism

The following GP functions will use parallelism automatically if available: `polclass`, `polmodular`, `znlog`, `fflog`, `poldisc` and `polresultant` (the last two only over $\mathbb{Z}[X]$).

```
? a = ffgen(2^127,'a); g = ffprimroot(a);
? b = random(a);
? my(t=getwalltime());fflog(b, g);getwalltime()-t
%3 = 1141
? my(t=getwalltime());polmodular(101);getwalltime()
%4 = 782
? default(nbthreads,1)
? my(t=getwalltime());fflog(b, g);getwalltime()-t
%6 = 5034
? my(t=getwalltime());polmodular(101);getwalltime()
%7 = 5131
```

derivum

It is now possible to compute numerical k-th derivatives.

```
? derivnum(x=0,zeta(x),5)
%1 = -120.00023290755845472453598583779581975
? lfun(1,0,5)
%2 = -120.00023290755845472453598583779581975
```

It is also possible to ask for several derivatives at once:

```
? derivnum(x=0,exp(2*x),[1,3,5])
%2 = [2.00000000,8.00000000,32.0000000]
```

nfmodpr / nfmodprlift

GP has a new interface to work modulo a prime ideal, which converts number field elements to finite field elements.

```
? nf=nfinit(a^2-67); bnf=bnfinit(nf);
? pr=idealprimedec(nf,11)[1];
? modpr=nfmodprinit(nf, pr);
? u=bnf.fu[1]
%4 = Mod(5967*a-48842,a^2-67)
? u11=nfmodpr(nf,u,modpr)
%5 = 3
? o=fforder(u11)
%6 = 5
? nfmodprlift(nf,u11^2,modpr)
%7 = 9
? u^2
%8 = Mod(-582880428*a+4771081927,a^2-67)
```

nfeltsign

`nfeltsign` return the sign of the real embeddings of an algebraic bumber.

```
? nf=nfinit(a^2-2);
? nf.roots
%2 = [-1.41421356,1.41421356]
? u=Mod(1-a,a^2-2)^1000;
? subst(lift(u),a,-sqrt(2))
%4 = 5.96602869E382
? subst(lift(u),a,sqrt(2))
%5 = 0.E364
? nfeltsign(nf,u)
%6 = [1,-1]
```

nfislocalpower

```
? K = nfinit(y^2+1);
? P = idealprimedec(K,2)[1]; \\ the ramified prime
? nfislocalpower(K,P,-1, 2) \\ -1 is a square
%3 = 1
? nfislocalpower(K,P,-1, 4) \\ ... but not a 4-th p
%4 = 0
? nfislocalpower(K,P,2, 2)  \\ 2 is not a square
%5 = 0
```

bnrinit

The functions `bnfinit` and `bnrinit` do not require the flag 1 for most operations.

```
? bnf=bnfinit(a^2-101329);
? bnr=bnrinit(bnf,1);
? setrand(1); lift(rnfkummer(bnr))
%3 = x^3+(264934206*a-84334433799)*x
     +(113518196297 50*a-3613535788129902)
```

rnfidealprimedec

```
? N = nfinit(a^2+23);
? R = rnfinit(N, x^3-x-1);
? pr= idealprimedec(N,11);
? [[id.e,id.f]| id<-pr]
%4 = [[1,2]]
? Pr=rnfidealprimedec(R, pr[1]);
? [[id.e,id.f]| id<-Pr]
%6 = [[1,2],[1,2],[1,2]]
```

`rnfidealfactor` is also available.

polclass, polmodular

`polclass` and `polmodular` now handles more invariants, the most important being j (code 0) Weber f (code 1) and γ_2 (code 5).

```
? polclass(-23)
%1 = x^3+3491750*x^2-5151296875*x+12771880859375
? polclass(-23,5)
%2 = x^3+155*x^2+650*x+23375
? polclass(-23,1)
%3 = x^3-x^2+1
? polmodular(5)
%4 = x^6+(-y^5+3720*y^4-4550940*y^3+2028551200*y^2-
? polmodular(5,5)
%5 = x^6+(-y^5+1240*y^2)*x^5+(20620*y^4+66211200*y)
? polmodular(5,1)
%6 = x^6-y^5*x^5+4*y*x+y^6
```

alggroup

`alggroup` return the group algebra associated to a group.

```
? G=galoisinit(x^6+108);
? A=alggroup(G);
? algissemisimple(A)
%3 = 1
? apply(algdim,algsimpledec(A))
%4 = [1,1,4]
```

zetamultall

`zetamultall` allows to compute all multi-ζ value at once.

```
? \p100
   realprecision = 115 significant digits (100 digi
? Z=zetamultall(10);
? ##
  ***   last result computed in 28 ms.
? V=vector(#Z,i,zetamult(zetamultconvert(i,0)));
? ##
  ***   last result computed in 921 ms.
? zmult(evec) = Z[zetamultconvert(evec,2)];
? zmult([2,3,5])
%4 = 0.06720349152930022968164943566400749276999679
? zetamult([2,3,5])
%5 = 0.06720349152930022968164943566400749276999679
```

lfungenus2

the `lfun` interface now support L function of genus-2 curves over $\mathbb{Q}$. For example for the curve

$$y^2 + (x^3 + x^2 + 1)y = x^2 + x$$

```
? L=lfungenus2([x^2+x, x^3+x^2+1]);
? lfunan(L,10)
%2 = [1,-3,-2,4,0,6,0,-3,3,0]
? localprec(19); lfun(L,1)
%3 = 0.090490390832429629 11
```

See Karim tutorial on L-functions.

ellpointtoz over the *p*-adic number

```
? E=ellinit([0,0,1,-1,0],O(37^8));
? P=[0,0]; ellisoncurve(E,P)
%12 = 1
? z=ellpointtoz(E,P)
%13 = Mod((21+2*37+8*37^2+36*37^3+29*37^4+27*37^5+2
? Q=liftall(bestappr(ellztopoint(E,z^5)))
%14 = [1/4,-5/8]
? ellisoncurve(E,Q)
%15 = 1
```

Elliptic curves with prime order

The function `ellsea(...,1)` allows to find random curves with prime order faster.

```
? getcurve(p)=
{
  parfor(a=1,oo,
    my(E=ellinit([1,a],p));
    isprime(ellsea(E,1))
    ,r ,if(r,return(a)));
}
? p=randomprime(2^128);
? a=getcurve(p);
? ##
  ***   last result computed in 3,885 ms.
? E=ellinit([1,a],p);
? isprime(ellcard(E))
```

Miscellaneous

`serprec`: return the precision of power series

```
? serprec(x^2+O(x^9),x)
%1 = 9
```

`qfeval`: evaluate a binary quadratic form

```
? qfeval(matid(24),[1..24])
%2 = 4900
? qfeval(matid(24),[1..24],vector(24,i,1))
%3 = 300
? sumnumap(i=1,i^-(2+1/i))
%4 = 1.4759745320394854707170086661697192944
? matpermanent([a,b;c,d])
%5 = d*a+c*b
```