Some new GP features

A tutorial

B. Allombert

IMB
CNRS/Université de Bordeaux

11/01/2016

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

parisizemax (recall from last year)

GP can now increase the stack dynamically to avoid stack overflow. Set in your `.gprc` the default `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
```

`threadsizemax` is also available for thread stacks.

64bit Windows support with mingw64

Thanks to Eric Driver it is now possible to build PARI/GP for 64bit Windows using mingw64.
As a result, we now provide 64bit Windows binaries.
It is also possible to build PARI with cygwin64.

Hexadecimal and binary input

Integers can be input in hexadecimal and binary format with the prefix 0*x* and 0*b*:

```
? 0xBABEC007
%14 = 3133063175
? 0b11111
%15 = 31
```

Bit precision

GP has support for setting the default precision in bits:

```
? default(realbitprecision,100)
```

or

```
? \pb100
```

The internal precision of `t_REAL` is still a multiple of 32/64 bits:

```
? bitprecision(1.)
%2 = 128
```

Bit precision

It is possible to set the precision in bit locally:

```
? localbitprec(32); bitprecision(1.)
%2 = 64
```

A small number of functions will actually compute the result to the required precision to be faster:

```
? localbitprec(100);intnumromb(x=2,3,zeta(x))
%1 = 1.3675256886839791457066699269000336640
? ##
  ***   last result computed in 180 ms.
? localbitprec(128);intnumromb(x=2,3,zeta(x))
%2 = 1.3675256886839791457066699268939213567
? ##
  ***   last result computed in 2,869 ms.
```

call

The function `call` allows to call a fonction on a vector of arguments:

```
? call(sin,[Pi/6])
%1 = 0.50000000000000000000000000000000000000
? call("_!",[5])
%2 = 120
? printc(x[..])=call(printsep,[":",x]);
? printc(1,2,3)
1:2:3
```

Associative arrays

It is now possible to define associative arrays with arbitrary index:

```
? M = Map(); \\ create an empty map
? mapput(M,"foo",17);
? mapput(M,888,289);
? mapput(M,x^2+1,4913);
? Vec(M)
%5 = [888,x^2+1,"foo"]
? mapget(M,"foo")
%6 = 17
? mapisdefined(M,888,&n)
%7 = 1
? n
%8 = 289
```

Associative arrays

```
? mapdelete(M,888);
? mapget(M,888)
  ***   at top-level: mapget(M,888)
  ***                 ^-------------
  *** mapget: non-existent component in mapget:
      index not in map
```

Maps can be converted to/from two-columns matrices:

```
? M=Map(["c",3;"b",2;"a",1]);
? mapget(M,"b")
%2 = 2
? Mat(M)
%3 = ["a",1;"b",2;"c",3]
```

L functions

An interface to *L*-functions has been added, see Karim tutorial. The old (broken) functions `zetak` and `zetakinit` have been removed. They can easily be replaced by `lfun`. For example, to compute the value of the Dedekind ζ function of $\mathbb{Q}(\sqrt{-1})$ at 2, one can simply do

```
? lfun(x^2+1,2)
%1 = 1.5067030099229850308865650481820713960
```

Dirichlet, Hecke and Artin and elliptic L-functions are also available. Theses functions honor the bit precision.

Summation methods

The function `sumnum` now use Euler-MacLaurin instead of Abel-Plana. Two summation methods using Gauss quadrature have been added: `intnumgauss` for quadrature, and `sumnummonien` for infinite sum.

Modular polynomial for SEA

PARI will now automatically compute missing modular polynomials needed for the SEA algorithm, by using Hamish parallel implementation of Sutherland algorithm.
So it is possible to use SEA even when `seadata` is not installed, thought it is slower.

ellfromeqn

The function `ellfromeqn` computes a Weierstrass model for a genus 1 curve given by a plane model $P(X, Y) = 0$, using formulae from Artin, Tate and Villegas.
Counting points on an elliptic curve given by an Edward model

```
? F=ellfromeqn(X^2+Y^2-(1+3*X^2*Y^2));
%1 = [0,-4,0,-12,48]
? E=ellinit(F);
? ellcard(E,nextprime(2^100))
%3 = 1267650600228229911275035985812
```

elltwist

The function `elltwist` returns a twist of an elliptic curve by a quadratic extension. The extension can be omited for finite fields. Example: Twist by an extension of discriminant -3:

```
? E=ellinit([1,7]*Mod(1,19));lift(elltwist(E))
%1 = [0,0,0,11,12]
? elltwist(ellinit([0,a2,0,a4,a6]),-3)
%2 = [0,-3*a2,0,9*a4,-27*a6]
? E=ellinit([a1,a2,a3,a4,a6]*Mod(1,2));
? lift(elltwist(E,x^2+x+T))
%1 = [a1,a2+a1^2*T,a3,a4,a6+a3^2*T]
```

ellminimaltwist

If E is a rational elliptic curve, `ellminimaltwist(E)` returns a discriminant which gives the twist of E whose minimal model has the minimal discriminant (or minimal conductor, if the flag is set)

To find the curve with j-invariant 3 and minimal discriminant and conductor

```
? E=ellminimalmodel(ellinit(ellfromj(3)));
? ellglobalred(E)[1]
%2 = 357075
? D = ellminimaltwist(E,1)
%3 = -15
? E2=ellminimalmodel(ellinit(elltwist(E,D)));
? ellglobalred(E2)[1]
%5 = 14283
```

ellisomat

If *E* is a rational elliptic curve, `ellisomat(E)` computes representatives of the isomorphism classes of elliptic curves *Q*-isogenous to *E*.

```
? E=ellinit([0,-1,1,-10,-20]); \\ X_0(11)
? [L,M]=ellisomat(E,1);
? L \\ List of representatives curves
%3 = [[-31/3,-2501/108],[-23461/1875,-28748141/1687
      [-1/3,19/108]]
? M \\ isogeny matrix
%4 = [1,5,5;5,1,25;5,25,1]
```

Without the flag, it also returns the isogenies from/to the original curve.

ellmoddegree

If E is a rational elliptic curve, `ellmoddegree(E)` computes the modular degree of E divided by the square of the Manin constant.

```
? E=ellinit("5077a1");
? ellmoddegree(E)
%2 = [1984,-128]
```

qforbits

`qforbits` returns the orbits of V under the action of the group of linear transformation generated by the set G.

```
?  Q=matid(6); G=qfauto(Q); V=qfminim(Q,3);
?  apply(x->[x[1],#x],qforbits(G,V))
%2 = [[[0,0,0,0,0,1]~,6],[[0,0,0,0,1,-1]~,30],
      [[0,0,0,1,-1,-1]~,80]]
```

We see there is only one orbit for each norm ≤ 3.

parforvec

parforvec is the parallel version of forvec, with an interface similar to parfor:

```
? parforvec(v=[[1,3],[1,3]],factorback(v) \
             ,f,print(v,":",f))
```

Miscellaneous

```
? cotanh(1)
%1 = 1.3130352854993313036361612469308478329
? sinc(Pi/2)
%2 = 0.63661977236758134307553505349005744814
? ramanujantau(101)
%3 = 81742959102
? zetamult([2,1])
%4 = 1.2020569031595942853997381615114499908
```