Some new GP features

A tutorial

B. Allombert

IMB
CNRS/Université de Bordeaux

02/06/2021

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

foreach, parforeach

`foreach` iterate over the elements of a vector. `parforeach` is the parallel version.

```
? V = binomial(10)
%1 = [1,10,45,120,210,252,210,120,45,10,1]
? foreach(V,n,print1(sigma(n)," "))
1 18 78 360 576 728 576 360 78 18 1
? my(s=0);parforeach(V,n,sigma(n),S,s+=S);s
%3 = 2794
```

holes in simultaneous assignment

The syntax `[a, b, c] = V` set `a` to `V[1]`, `b` to `V[2]` and `c` to `V[3]`. Now it is possible to omit some variables:

```
? gcdext(135,95)
%4 = [-7,10,5]
? [,v,d]=gcdext(135,95); [v,d]
%5 = [10,5]
? my([u,,d]=gcdext(135,95)); [u,d]
%6 = [-7,5]
```

setdebug

The function `setdebug` allows to set DEBUGLEVEL for each debug domains separately. Without parameters, `setdebug` returns the debug domains and their values. The current domains are `alg, arith, bern, bnf, bnr, bnrclassfield, bb_group, compiler, ell, ellanal, ellcard, ellisogeny, ellrank, ellsea, factor, factorff, factorint, factormod, fflog, galois, gammamellininv, genus2red, hensel, hyperell, intnum, io, isprime, lfun, mat, mathnf, mf, mod, mpqs, ms, mt, nf, nffactor, nflist, nfsubfields, padicfields, pol, polclass, polgalois, polmodular, polroots, qf, qflll, qfsolve, qfisom, quadclassunit, rnf, stark, subcyclo, subgrouplist, thue, trans, zetamult`

setdebug

```
? setdebug()
%7 = ["alg",0;"arith",0;"bern",0;"bnf",0;"bnr",0;..
? setdebug("qflll",5);
? nfinit(x^8+1);
% Entering L^2 (double): LLL-parameters (0.990,0.51
% K2 K3 K4 K5 K6 K7 K8 Time LLL: 0
% Entering L^2 (dpe): LLL-parameters (0.990,0.510)
% K2 K3 K4 K5 K6 K7 K8 Time LLL: 0
? setdebug("qflll")
%10 = 5
? default(debug,0)
```

`default(debug,0)` (or `\g0`) set all the debug domains to 0.

permcycles

`permcycles` returns the cycle decomposition of a permutation. Fixed points are returned as length-1 cycles.

```
? permcycles(Vecsmall([2,7,1,8,4,5,9,10,3,6]))
%12 = [Vecsmall([1,2,7,9,3])
%      ,Vecsmall([4,8,10,6,5])]
? permcycles(Vecsmall([3,1,4,5,9,2,6,8,7]))
%13 = [Vecsmall([1,3,4,5,9,7,6,2]),Vecsmall([8])]
```

halfgcd for integers

Let a, b be two integers. `halfgcd(a,b)` returns $[M, [x, y]]$ where M is a 2×2 matrix of determinant ± 1 such that `M*[x,y] = [a,b]` and $a \geq \sqrt{\max(|x|, |y|)} > b$

```
? [M,C] = halfgcd(23,59)
%14 = [[3,-1;-5,2],[10,3]~]
? M*[23,59]~
%15 = [10,3]~
```

halfgcd for polynomials

Let a, b be two polynomials. `halfgcd(a,b)` returns $[M, [x, y]]$ where M is a 2×2 matrix of determinant of degree 0 such that `M*[x,y] = [a,b]` and $\deg a \geq \max(\deg x, \deg y)/2 > \deg b$

```
? P = truncate(sqrt(1+x+O(x^4))); Q = x^4;
? [M,C] = halfgcd(P, Q)
%17 = [[-16*x-32,1;-1/4*x^2+2*x+6,1/64*x-5/32],
%      [-4*x^2-32*x-32,5*x+6]~]
? M*[P,Q]~
%18 = [-4*x^2-32*x-32,5*x+6]~
```

bnrinit

It is now possible to work with the n-torsion quotient of a ray class group without computing the full `bnrinit` (which could require costly factorization and discrete logarithm) by using `bnrinit(bnf,,n)`.

```
? bnf = bnfinit(a^2+47);
? bnr = bnrinit(bnf,77); bnr.cyc
%20 = [120,30,3]
? bnr3 = bnrinit(bnf,77,,3); bnr3.cyc
%21 = [3,3,3]
? bnrclassfield(bnr3)
%22 = [x^3+(-132*a-6171)*x+(5863/2*a+765259/2),
%      x^3+(-231/2*a+735/2)*x+(-203/2*a-14749/2),
%      x^3+(-6*a+3)*x+(-67/2*a+695/2)]
```

bnrinit

This is advantageous when the full `bnr` would be too costly to compute

```
? p = nextprime(2^64);
? \\ bnr = bnrinit(bnf,p); very very slow
? bnr = bnrinit(bnf,p,, (2*3*5*7)^10); \\ fast
? bnr.cyc
%26 = [2100]
? F=bnrclassfield(bnr,3)
%27 = [x^3+5534023221128654887*x
%       +3336489818229325039369372040 63*a]
```

rnfconductor

The reverse function `rnfconductor` now has a flag 1 to compute only the n-torsion quotient of the `bnr` where n is the degree of the abelian extension.

```
? \\ R = rnfconductor(bnf,F[1]); \\very slow
? R = rnfconductor(bnf,F[1],1); \\fast
? [cnd,bnr2,subg] = R; cnd
%30 = [[18446744073709551629,0
%       ;0,18446744073709551629],[]]
? bnr2.cyc
%31 = [3]
```

rnfconductor

The flag 2 allows to compute only the conductor and its factorization.

```
? [cnd,cndf] = rnfconductor(bnf,F[1],2);
? cnd
%33 = [[18446744073709551629,0
%       ;0,18446744073709551629],[]]
? cndf
%34 = Mat([[18446744073709551629,
%           [18446744073709551629,0]~,1,2,1],1])
```

galoissplittinginit

`galoissplittinginit(P)` is a faster alternative to `galoisinit(nfsplitting(P))` that assumes P irreducible. but does not require the group to be weakly supersolvable.

```
? P = x^5+20*x+16;
? polgalois(P)
%36 = [60,1,1,"A5"]
? G = galoissplittinginit(P);
? G.pol == nfsplitting(P)
%38 = 1
? galoisidentify(G)
%39 = [60,5]
? galoisfixedfield(G,[G.group[2],G.group[6]],1)
%40 = x^6-1600*x^4+1536000*x^2+32768000*x+163840000
```

lfunparams

`lfunparams` returns the triplet $[N, k, Vga]$ that describes the functional equation.

```
? E = ellinit([2,3]);
? [N,k,vga] = lfunparams(E)
%42 = [880,2,[0,1]]
```

lfundual

`Ld=lfundual(L)` returns the L-function dual of L, such that the usual functional equation holds:

$$\Lambda_L(s) = \epsilon \Lambda_{Ld}(k - s)$$

where ϵ is the root number.

```
? L = lfunqf(matdiagonal([1,2,3,4]));
? Ld = lfundual(L);
? eps = lfunrootres(L)[3]
%45 = 2.4494897427831780981972840747058913920
? lfunlambda(L,Pi)/lfunlambda(Ld,2-Pi)
%46 = 2.4494897427831780981972840747058913920
```

polylogmult

The function `polylogmult` is an extension of `zetamult` to compute multiple polylogarithm values.

```
? polylogmult([2,2,2],[1,-1,1])
%47 = 0.071635745321742507017850817313956412547
```

zetamultdual

The function `zetamultdual(s)` returns the dual sequence of *s*.

```
? v = [3,5,2,2];
? vd = zetamultdual(v)
%49 = Vecsmall([2,2,2,1,1,1,2,1])
? zetamultdual(vd)
%50 = Vecsmall([3,5,2,2])
? zetamult(v)
%51 = 5.428060757520686813285052036544562452 2E-5
? zetamult(vd)
%52 = 5.428060757520686813285052036544562452 2E-5
```

zeros of Bessel functions

The functions `besseljzero` and `besselyzero` return the *n*-th zero of the corresponding Bessel function:

```
? bz=besseljzero(Pi,1)
%53 = 6.5531024735734022070727271054828304261
? besselj(Pi,bz)
%54 = -5.95766425368725432127998298063590759 22E-40
? bz=besselyzero(2,10)
%55 = 32.143002257627550524573578252532211033
? bessely(2,bz)
%56 = -1.996860227469671647997313218802789421E-39
```

hypergeom

`hypergeom` now supports power series:

```
? hypergeom([1,1,1],[5/3,5/3],z+O(z^5))
%57 = 1+9/25*z+81/400*z^2+6561/48400*z^3+59049/5929
? hypergeom([1,1,1],[5/3,5/3],1/2)
%58 = 1.2579176749481097328724460713125550477
? hypergeom([1,1,1],[5/3,5/3],1/2+z+O(z^5))
%59 = 1.2579176749481097 33+0.761146792710307 2052*z
% +0.788737051026845 7097*z^2+1.007980535530049 765*z
% +1.437897250391074 849*z^4+O(z^5)
```

ramanujantau

The function `ramanujantau` can now compute the coefficients of other newforms of level 1, for weight 12, 16, 18, 20, 22, or 26.

```
? mf=mfinit([1,16,1],1);
? S=mfeigenbasis(mf)[1];
? mfcoef(S,101)
%62 = -817641571654098
? ramanujantau(101,16)
%63 = -817641571654098
? mfcoef(S,10001)
%64 = -870376615320443566571062660188
? ramanujantau(10001,16)
%65 = -870376615320443566571062660188
```

Here, `ramanujantau` is much faster than `mfcoef`.

lambertw

`lambertw` now supports complex arguments and all branches.

```
? l=lambertw(-1/4)
%66 = -0.35740295618138890 31
? l*exp(l)
%67 = -0.25000000000000000
? l=lambertw(-1)
%68 = -0.31813150520476413 53+1.33723570143068940 9*I
? l*exp(l)
%69 = -1.00000000000000000+5.42101086242752217 0E-2
? l=lambertw(-1,1)
%70 = -2.06227772959828388 5+7.58863117847251262 3*I
? l*exp(l)
%71 = -1.00000000000000000+1.35525271560688054 3E-1
```

Miscellaneous

```
? eulerreal(100)
%72 = 2.9035283466610974970546038347644358751E138
? eulerfrac(100)
%73 = 29035283466610974970546038347644358750775530
? Qfb(1,2,3).disc
%74 = -8
```