Some new GP features

A tutorial

B. Allombert

IMB
CNRS/Université de Bordeaux

10/01/2022

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` sets `a` to `V[1]`, `b` to `V[2]` and `c` to `V[3]`. It is now 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 factorizations and discrete logarithms) 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+55340232221128654887*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 dual L-function 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
```

lfuneuler

`lfuneuler(L,p)` returns the Euler factor at *p* of the *L*-function *L* assuming it is known. This is especially useful at bad places.

```
? lfuneuler(Mod(2,5),3)
%47 = 1/(I*x+1)
? lfuneuler(Mod(2,5),5)
%48 = 1
? L=lfungenus2([x^2 + x,x^3 + x^2 + 1]);
? lfuneuler(L,11)
%50 = 1/(121*x^4+11*x^2+1)
? lfuneuler(L,13)
%51 = 1/(13*x^2+5*x+1)
```

polylogmult

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

```
? polylogmult([2,2,2],[1,-1,1])
%52 = 0.071635745321742507017850817313956412547
```

zetamultdual

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

```
? v = [3,5,2,2];
? vd = zetamultdual(v)
%54 = Vecsmall([2,2,2,1,1,1,2,1])
? zetamultdual(vd)
%55 = Vecsmall([3,5,2,2])
? zetamult(v)
%56 = 5.4280607575206868132850520365445624522E-5
? zetamult(vd)
%57 = 5.4280607575206868132850520365445624522E-5
```

zeros of Bessel functions

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

```
? bz=besseljzero(Pi,1)
%58 = 6.5531024735734022070727271054828304261
? besselj(Pi,bz)
%59 = -5.9576642536872543212799829806359075922E-40
? bz=besselyzero(2,10)
%60 = 32.143002257627550524573578252532211033
? bessely(2,bz)
%61 = -1.9968602274696716479973132188802789421E-39
```

harmonic

`harmonic(n,k=1)` computes the sum $\sum_{i=1}^{n} 1/i^k$ as a rational number.

```
? harmonic(10)
%62 = 7381/2520
? sum(i=1,10,1/i)
%63 = 7381/2520
? harmonic(10,3)
%64 = 19164113947/16003008000
? sum(i=1,10,1/i^3)
%65 = 19164113947/16003008000
```

harmonic

For large *n*, an approximate value can be computed with `psi`:

```
? harm(n,k) =
{
  my(s=k-1);
  (-1)^s*derivnum(x=0,psi(n+1+x)-psi(1+x),s)/s!;
}
? harm(100000,3)
%67 = 1.2020569031095947853972381615115333241
? harmonic(100000,3)*1.
%68 = 1.2020569031095947853972381615115333241
```

serdiffdep

Let S be a power series, `seralgdep(S,p,r)` finds a polynomial equation of degree p with polynomial coefficients of degree $< r$. satisfied by S, `serdiffdep(S,p,r)` finds an (inhomogenous) linear equation of degree p with polynomial coefficients of degree $< r$ satisfied by S.

```
? S = sum(i=0,100,binomial(2*i,i)/(i+1)*T^(i+1))\
    + O(T^101);
? seralgdep(S,3,3)
%70 = x^2-x+T
? S^2-S+T
%71 = 0
? serdiffdep(S,3,3)
%72 = [(4*T-1)*x-2,-1]
? (4*T-1)*S'-2*S==-1
%73 = 1
```

quadunitnorm, quadunitindex

`quadunitnorm(D)` returns the norm of fundamental unit of the order of discriminant *D*.

```
? D=4*nextprime(2^40);
? norm(quadunit(D))
%75 = 1
? quadunitnorm(D)
%76 = 1
```

`quadunitindex(D, f)` return the index of the unit group of the order of conductor *f* in the unit group of the order of fundamental discriminant *D*.

```
? quadunitindex(5,13)
%77 = 7
? w^7
%78 = 8+13*w
```

polsubcyclofast

`polsubcyclofast(n,d)` returns polynomials defining the subfields of $\mathbb{Q}(\zeta_n)$ with cyclic Galois groups of order *d*. When *d* is small, it is faster than `polsubcyclo` but return larger polynomials. Usually the polynomials returned by `polsubcyclo` can be recovered by `polredabs` up to sign.

```
? polsubcyclo(10^7+19,7)
%79 = x^7+x^6-4285722*x^5-941022196*x^4+17812918101
? P=polsubcyclofast(10^7+19,7)[1]
%80 = x^7-210000399*x^5+321720611268*x^4+4278170618
? polredabs(P)
%81 = x^7-x^6-4285722*x^5+941022196*x^4+17812918101
```

snfrank

`snfrank` compute the *p*-rank of a Smith normal form:

```
? snfrank([4,4,2], 2)
%82 = 3
? snfrank([4,4,2], 4)
%83 = 2
? snfrank([4,4,2], 8)
%84 = 0
? snfrank([4,4,2], 0)
%85 = 3
```

poltomonic

`poltomonic(P)` return a monic, integral polynomial U such that $U(x) = CT(x/L)$ for some $C, L \in \mathbb{Q}$.

```
? poltomonic(9*x^2 - 1/2)
%86 = x^2 - 2
? U = poltomonic(9*x^2 - 1/2, &L)
%87 = x^2 - 2
? L
%88 = 6
? U / subst(9*x^2 - 1/2, x, x/L)
%89 = 4
```

Miscellaneous

```
? eulerreal(100)
%90 = 2.9035283466610974970546038347644358751E138
? eulerfrac(100)
%91 = 290352834666109749705460383476443587507755300
? Qfb(1,2,3).disc
%92 = -8
? solve(x=-oo,oo,exp(x)-3*x)
%93 = 0.61906128673594511215232699402092223330
```