

---

# An introduction to the PARI/GP system

Karim Belabas

<http://pari.math.u-bordeaux.fr/>

# PARI/GP ?

---

«The most important thing in a programming language is the name. A language will not succeed without a good name. I have recently invented a very good name, and now I am looking for a suitable language.»

(Often attributed to) Donald Knuth.

# PARI/GP ?

---

«The most important thing in a programming language is the name. A language will not succeed without a good name. I have recently invented a very good name, and now I am looking for a suitable language.»

(Often attributed to) Donald Knuth.

«Science is knowledge which we understand so well that we can teach it to a computer; and if we don't fully understand something, it is an art to deal with it.»

(Correctly attributed to) Donald Knuth.

# PARI/GP ?

---

**PARI/GP** is a free software system for Number Theory. The system is built around three components:

- the **PARI** library, or **libpari**, the heart of the system. A large collection of fast special purpose routines, also made available as special cases of generic high level functions. Written and programmable in C.
- the **gp** interpreter, giving access to the library using the **GP** language: simpler to use than C and essentially optimal for high level code; slower in general.
- the **gp2c** compiler from GP  $\rightarrow$  C; as easy as GP, often as fast as C.

# Where to start ?

---

- Reference cards [doc/refcard\\*.dvi](#): more precisely
  - [refcard](#) (general reference, 2 pages),
  - [refcard-nf](#) (algebraic number theory, 2 pages),
  - [refcard-ell](#) (elliptic curves, 1 page),
  - [refcard-mf](#) (modular forms, 1 page),
  - [refcard-lfun](#) ( $L$ -functions, 1 page).
- Online help: [??function](#) or [???keyword](#)

# Where to start ?

---

- Reference cards [doc/refcard\\*.dvi](#): more precisely
  - [refcard](#) (general reference, 2 pages),
  - [refcard-nf](#) (algebraic number theory, 2 pages),
  - [refcard-ell](#) (elliptic curves, 1 page),
  - [refcard-mf](#) (modular forms, 1 page),
  - [refcard-lfun](#) ( $L$ -functions, 1 page).
- Online help: [??function](#) or [???keyword](#)
- GP tutorial ([doc/tutorial.dvi](#), 58 pages)

# Where to start ?

---

- Reference cards [doc/refcard\\*.dvi](#): more precisely  
[refcard](#) (general reference, 2 pages),  
[refcard-nf](#) (algebraic number theory, 2 pages),  
[refcard-ell](#) (elliptic curves, 1 page),  
[refcard-mf](#) (modular forms, 1 page),  
[refcard-lfun](#) ( $L$ -functions, 1 page).
- Online help: [??function](#) or [???keyword](#)
- GP tutorial ([doc/tutorial.dvi](#), 58 pages)
- GP user's manual ([doc/users.dvi](#), 508 pages)
- The <http://pari.u-bordeaux.fr/> website, in particular the [FAQ](#) and the [Documentation](#) tabs.

# Basics of the gp interpreter (1/2)

- = is the assignment operator. The semicolon ; is a separator between successive expressions.
- Input is evaluated line by line, as soon as `<Return>` is pressed *unless* the line ends with a = (middle of an assignment). Multi-line programs are input by surrounding the lines by braces.
- This defines a user function:

```
f(x) =  
{  
    my (a = 2*x);    \\ local variables  
    my (b = a^2);  
    return (a + b);  
}
```
- Comments: everything following `\\` to end of line, as well as `/* this text */`.
- Write a sequence of instructions (usually functions) in a file, then `\r file` to read it in.



# Basics of the `gp` interpreter (1/2)

---

- An expression has a value (result of the operation), the final result in a line is printed *unless* a trailing semicolon follows the final expression:

```
? a = 1
```

```
%1 = 1
```

```
? a = 1;    \\ nothing printed
```

- Successive results are stored in the output history `%1`, `%2`, ...
- Successive inputs are stored in the command history, which can be edited (keyboard arrows); `<TAB>` triggers a contextual completion.

# The PARI philosophy (1/4)

---

Algebraic expressions are input naively and represented exactly:

? 1 + 1

%1 = 2

? 2 / 6

%2 = 1/3

? (x+1)^(-2)

%3 = 1/(x^2 + 2\*x + 1)

? Mod(2,5)^3 \\ in  $\mathbb{Z}/5\mathbb{Z}$

%4 = Mod(3, 5)

? Mod(x, x^2+x\*y+y^2)^3 \\ in  $\mathbb{Q}[x, y]/(x^2 + xy + y^2)$

%5 = Mod(y^3, x^2 + y\*x + y^2)

# The PARI philosophy (2/4)

---

Other expressions are approximated numerically or as power series.

? Pi

%1 = 3.1415926535897932384626433832795028842

? log(2)

%2 = 0.69314718055994530941723212145817656807

? log(1+x)

%3 =  $x - \frac{1}{2}x^2 + \frac{1}{3}x^3 - \frac{1}{4}x^4 + \frac{1}{5}x^5 - \frac{1}{6}x^6 (\dots)$

# The PARI philosophy (2/4)

---

Other expressions are approximated numerically or as power series.

? Pi

%1 = 3.1415926535897932384626433832795028842

? log(2)

%2 = 0.69314718055994530941723212145817656807

? log(1+x)

%3 =  $x - 1/2*x^2 + 1/3*x^3 - 1/4*x^4 + 1/5*x^5 - 1/6*x^6 (\dots)$

In both cases a default accuracy is used (`realprecision` / `realbitprecision` or `seriesprecision`), which can be changed using `\pn`, `\pbn` or `\psn`, e.g.

? \pb 20

`realbitprecision = 20 significant bits (6 decimal digits displayed)`

? Pi

%4 = 3.14159

? \ps 3

`seriesprecision = 3 significant terms`

# The PARI philosophy (3/4)

---

“Everything that should make sense actually does.” A domain is determined where inputs make sense and computations are performed there:

```
? T = x^2 + 1;
```

```
? factor(T) \\ in  $\mathbb{Q}[x]$ 
```

```
%7 =
```

```
[x^2 + 1 1]
```

```
? factor(T * Mod(1,5)) \\ in  $\mathbb{F}_5[x]$ 
```

```
%8 =
```

```
[Mod(1, 5)*x + Mod(2, 5) 1]
```

```
[Mod(1, 5)*x + Mod(3, 5) 1]
```

```
? factor(T*(1 + O(5^3))); \\ in  $\mathbb{Q}_5[x]$ 
```

# The PARI philosophy (3/4)

“Everything that should make sense actually does.” A domain is determined where inputs make sense and computations are performed there:

```
? T = x^2 + 1;
```

```
? factor(T) \\ in  $\mathbb{Q}[x]$ 
```

```
%7 =
```

```
[x^2 + 1 1]
```

```
? factor(T * Mod(1,5)) \\ in  $\mathbb{F}_5[x]$ 
```

```
%8 =
```

```
[Mod(1, 5)*x + Mod(2, 5) 1]
```

```
[Mod(1, 5)*x + Mod(3, 5) 1]
```

```
? factor(T*(1 + 0(5^3))); \\ in  $\mathbb{Q}_5[x]$ 
```

```
? Mod(3,6) + Mod(2,4)
```

```
%10 = Mod(1, 2)
```

```
? Mod(1, x) + Mod(1, x+1) \\ in the null ring !
```

```
%11 = Mod(0, 1)
```

# The PARI philosophy (4/4)

---

Precomputations are useful! Various `init` functions are attached to certain mathematical contexts and their result are fed to other routines to specify a given context:

```
? E = ellinit([1,2]);      \\  $E/\mathbb{Q} : y^2 = x^3 + x + 2$ 
? elltors(E)
%2 = [4, [4], [[1, 2]]]
? ellmul(E, [1,2], 2)     \\  $[2]P$  on  $E$ ,  $P = [1, 2]$ 
%3 = [-1, 0]
```

# The PARI philosophy (4/4)

---

Precomputations are useful! Various `init` functions are attached to certain mathematical contexts and their result are fed to other routines to specify a given context:

```
? E = ellinit([1,2]);    \\ E/Q : y^2 = x^3 + x + 2
? elltors(E)
%2 = [4, [4], [[1, 2]]]
? ellmul(E, [1,2], 2)    \\ [2]P on E, P = [1,2]
%3 = [-1, 0]

? K = nfinit(y^2 + 23);    \\ Q(sqrt(-23)), basic invariants
? idealfactor(K,2)    \\ factor 2Z_K
? K = bnfinit(K);    \\ same field, deeper invariants
? K.clgp    \\ the class group of K is cyclic of order 3
%7 = [3, [3], [[2, 0; 0, 1]]]
```



# The PARI philosophy (4/4)

---

```
? L = lfuninit(1, [100]);    \\ Riemann  $\zeta$  at  $1/2 + it$ ,  $|t| < 100$ 
```

```
? lfunzeros(L,30)
```

```
%8 = [14.134..., 21.022..., 25.010...]
```

# The PARI philosophy (4/4)

---

```
? L = lfuninit(1, [100]);    \\ Riemann  $\zeta$  at  $1/2 + it$ ,  $|t| < 100$ 
```

```
? lfunzeros(L,30)
```

```
%8 = [14.134..., 21.022..., 25.010...]
```

```
? A = alginit(nfinit(y), [-1,-1]);    \\ quaternion alg.  $(-1,-1)_{\mathbb{Q}}$ 
```

```
? algiscommutative(A)
```

```
%10 = 0
```

## The PARI philosophy (4/4)

---

```
? L = lfuninit(1, [100]);    \\ Riemann  $\zeta$  at  $1/2 + it$ ,  $|t| < 100$ 
```

```
? lfunzeros(L,30)
```

```
%8 = [14.134..., 21.022..., 25.010...]
```

```
? A = alginit(nfinit(y), [-1,-1]);    \\ quaternion alg.  $(-1,-1)_{\mathbb{Q}}$ 
```

```
? algiscommutative(A)
```

```
%10 = 0
```

See also `galoisinit` (Galois groups), `nfmodprinit` ( $\mathbb{Z}_K \rightarrow \mathbb{Z}_K/\mathfrak{p}$ ), `rnfinit` ( $K \subset L$ ), `thueinit` ( $P(x,y) = a$ ), `rnfisnorminit` ( $N_{L/K}(x) = a$ ), `qfisominit` ( $\Lambda \simeq \Lambda'$ ?), `intnuminit`, `sumnuminit`...

# GP gems (1/3) : Euclid

---

```
GCD(a,b) = {  
    while(b, [a,b] = [b, a%b]);  
    return (a);  
}
```

# GP gems (1/3) : Euclid

---

```
GCD(a,b) = {  
    while(b, [a,b] = [b, a%b]);  
    return (a);  
}
```

```
/* [d,u] = GCDEXT(a,b):  au + bv = d; */
```

```
GCDEXT(a,b) = {  
    my(u = 1, v = 0);  
    while(b,  
        my([q,r] = divrem(a,b));  
        [a, b] = [b, r];  
        [u, v] = [v, u-q*v];  
    );  
    return ([a,u]);  
}
```

# GP gems (2/3) : determinant in $M_n(\mathbb{Z})$ via CRT

---

We must compute  $\det M \pmod{p}$  for primes  $p \leq x$ , such that

$$\prod_{p \leq x} p > 2B, \quad \text{where } B = \text{Hadamard}(M) := \prod_i \|M_i\|_2.$$

**Theorem** (Rosser-Schoenfeld, weak version).

$$\sum_{p \leq x} \ln p > 0.84 \cdot x, \quad \text{for } x > 100.$$

# GP gems (2/3) : determinant in $M_n(\mathbb{Z})$ via CRT

We must compute  $\det M \pmod{p}$  for primes  $p \leq x$ , such that

$$\prod_{p \leq x} p > 2B, \quad \text{where } B = \text{Hadamard}(M) := \prod_i \|M_i\|_2.$$

**Theorem** (Rosser-Schoenfeld, weak version).

$$\sum_{p \leq x} \ln p > 0.84 \cdot x, \quad \text{for } x > 100.$$

```
Hadamard(M) = sqrt( prod(i=1, #M, norml2(M[,i])) );
```

```
detZ(M) = {
```

```
  my (v, B = 2*Hadamard(M), x = max(100, log(B) / 0.84));
```

```
  v = [ matdet(M * Mod(1,p)) | p <- primes([2, x]) ];
```

```
  centerlift( chinese(v) );
```

```
}
```

# GP gems (2/3) : modular determinant, continued

---

Without Rosser-Schoenfeld estimate, using an infinite loop over primes:

```
detZ2(M) = {  
  my (p, q = 1.0, B = 2*Hadamard(M), v = List());  
  forprime(p = 2, +oo,  
    listput(v, matdet(M * Mod(1,p)));  
    q *= p; if (q > B, break);  
  );  
  centerlift( chinese(v) );  
}
```



# GP gems(3/3), partial squarefree factorization over $\mathbb{F}_q[X]$

If  $T \in \mathbb{F}_q[X]$  factors into irreducibles as  $T = \prod_i T_i^{e_i}$  and

$$u = \gcd(T, T'), \quad v = T/u, \quad w = u / \gcd(u, v^{\deg T})$$

then

$$v = \prod_{i: p|e_i} T_i \quad \text{and} \quad w = \prod_{i: p|e_i} T_i^{e_i} = W(X^p).$$

# GP gems(3/3), partial squarefree factorization over $\mathbb{F}_q[X]$

If  $T \in \mathbb{F}_q[X]$  factors into irreducibles as  $T = \prod_i T_i^{e_i}$  and

$$u = \gcd(T, T'), \quad v = T/u, \quad w = u / \gcd(u, v^{\deg T})$$

then

$$v = \prod_{i: p \nmid e_i} T_i \quad \text{and} \quad w = \prod_{i: p \mid e_i} T_i^{e_i} = W(X^p).$$

```
vW(F, T) = {  
  my(p = F.p, q = p^(F.f), n = poldegree(T));  
  my(u,v,w,W, X = variable(T));  
  u = gcd(T,T'); v = T/u; w = u / gcd(u, lift(Mod(v,u)^n));  
  W = apply(a->a^(q/p), substpol(w, X^p, X));  
  return ([v, W]);  
}
```

```
F = ffgen(5^7, 't);    \\ a generator for  $\mathbb{F}_{5^7}$ 
```

```
T = random(F*x^10) * random(F*x^10)^5;
```

```
[v,W] = vW(F, T)
```

# Control structures (1/3)

This program computes

$$R(x) = \left( \zeta(2) \sum_{a \leq \sqrt{x}} \mu(a) \left\lfloor \frac{x}{a^2} \right\rfloor - x \right) x^{(-2/5)} = O(1) \text{ under GRH.}$$

```
R(x) = {  
    my(s);  
    s = zeta(2) * sum(a=1, sqrt(x), moebius(a)*(x\a^2));  
    (s - x) / x^0.4;  
}  
? R(10^7)  
time = 3 ms.  
%1 = 0.052092560787004188970344062406837190410  
? R(10^12)  
time = 832 ms.  
%2 = 0.010948893958117048274619354741759352927  
? R(10^15);  
time = 51,805 ms.
```

## Control structures (2/3)

Another version, using the fact that multiplicative functions such as `moebius` also accepts inputs of the form `[a, factor(a)]` instead of a plain integer `a`:

```
S(x) = {  
  my(s = 0);  
  forfactored(N = 1, floor(sqrt(x)),  
    my(a = N[1]);  
    s += moebius(N)*(x\a^2));  
  (zeta(2)*s - x) / x^0.4  
}  
?  
S(10^7);  
time = 7 ms.  
?  
S(10^12);  
time = 984 ms.  
?  
S(10^15);  
time = 35,903 ms.
```

# Control structures (3/3)

---

`if`(*bool*, *seq*<sub>1</sub>, *seq*<sub>2</sub>)

`while`(*bool*, *seq*)

`for`(*i* = *a*, *b*, *f*)  $f(a), f(a + 1), \dots$

`forprime`(*p* = *a*, *b*, *f*) same over primes in  $[a, b]$

`forstep`(*i* = *a*, *b*, *step*, *f*)  $f(a), f(a + \textit{step}), \dots$

`fordiv`(*N*, *d*, *f*)  $\sum_{d|N} f(d)$

`forvec`(*X* =  $[[a, b], [c, d]]$ , *f*)  $f(a, c), f(a, c + 1), \dots, f(a, d)$   
 $f(a + 1, c), f(a + 1, c + 1), \dots, f(a + 1, d)$   
...  
 $f(b, c), f(b, c + 1), \dots, f(b, d)$

`break` / `next` / `return`

Check also `forsubset`, `forperm`, `forpart`, `forsubgroup`, `forell`, `forfactored`,  
`fordivfactored`...