Introduction to PARI/GP (1/3)

Karim Belabas

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

UNIVERSITÉ PARIS-SUD 11

Glossary

As the name suggests, PARI/GP is two-sided :

- PARI is a library of C routines, oriented towards number-theoretic applications. (Fast)
- gp is an interpreter, giving access to PARI through a command-line shell. It is programmable, in the scripting language GP. (Easy)

The `gp2c` compiler is a standalone tool, translating GP scripts to PARI C code. Transparent interface `gp2c-run`, loading optimized scripts into a new session. (Fast + Easy, but supports a subset of the language.)

What PARI/GP does well

- standard computations on integers and floats in arbitrary precision ; (e.g. factorization and primality testing for integers).
- transcendental functions ;
- univariate polynomials (e.g. factorization over $\mathbb{C}$, $\mathbb{Q}_p$, number fields) and formal power series ;
- number fields and class field theory (strongest point) ;
- elliptic curves ;
- linear algebra over $\mathbb{Z}$, $k[X]$, or a field ;
- lattice reduction and standard applications (shortest vectors, recognizing algebraic numbers, etc.).

What PARI/GP supports

- numerical analysis (numerical integration, summing series, linear algebra) ;
- graphism.

What PARI/GP does badly

- non-prime finite fields ;
- multivariate polynomials or power series ;
- sparse computations, asymptotically efficient algorithms.

Conclusion

PARI/GP is not a computer algebra system, although it includes many facilities for symbolic computations.

Availability

- Free software (GPL), public development version and bug-tracking database.
- Requires between 6 to 15MB disk space, and 4MB RAM (i.e. no practical restrictions).
- Some architectures better supported (e.g Linux + gcc + ix86), but highly portable (from PDA to mainframes)
- The development version supports both the native multiprecision kernel and GMP.

Variables — Programming (1/4)

Assignment : `x = 1`

Instruction separator : `;`

(at end of line `;` prevents printing of results)

Variables are neither declared nor typed, although their *value* has a type. One may use x prior to any assignment : it is then a degree 1 polynomial. On the other hands, x[5] cannont be used before `x` is initialized to a vector type.

GP types — Programming (2/4)

- `t_INT`, `t_FRAC`
- `t_COMPLEX`, `t_QUAD`
- `t_REAL`, `t_PADIC`
- `t_INTMOD`, `t_POLMOD`
- `t_POL`, `t_RFRAC`,
- `t_SER`
- `t_QFI`, `t_QFR`
- `t_VEC`, `t_COL`, `t_MAT`
- `t_LIST`, `t_STR`, `t_VECSMALL`

GP types — Programming (2/4)

- t_INT, t_FRAC (exact)
- t_COMPLEX, t_QUAD (? ?)
- t_REAL, t_PADIC (inexact)
- t_INTMOD, t_POLMOD (exact, modular)
- t_POL, t_RFRAC,
- t_SER (inexact)
- t_QFI, t_QFR
- t_VEC, t_COL, t_MAT
- t_LIST, t_STR, t_VECSMALL

Whitespace ignored, but an executable statement = a single line. Executed as soon as `Enter` is pressed unless

- line is terminated by `=`, or
- line in a group enclosed between braces `{ }`; group is executed as the closing brace is found. (Braces are then removed from the input.)

For instance `fun(x) =`

```
  {
    x * x
  }
```

is equivalent to `fun(x) = x*x`

Programming (4/4)

User function : returned value is the result of the last evaluated expression in the function body. `fun(x, y) = x * y \\standard`

```
fun(x, y) = \\local variables
  local(z = x*y); z^2
```

```
fun(x, y = 2) = x * y \\default argument
```

Arguments are passed as *parameters* (copy made if mutable object).

Member function : different syntax, unique argument, arguments passes as *variables*. `x.a = x[1]`