Tutorial: using libpari in GP scripts

Karim Belabas

http://pari.math.u-bordeaux.fr/
Looking for the right GP function

E.g. Hensel lifts:

users.dvi

???keyword

pari-users@pari.math.u-bordeaux.fr

pari-dev@pari.math.u-bordeaux.fr

???Hensel
Looking for the right GP function

E.g. Hensel lifts:

users.dvi

???keyword

pari-users@pari.math.u-bordeaux.fr

pari-dev@pari.math.u-bordeaux.fr

???Hensel

But we can as well inspect what **libpari** provides

???keyword@

libpari.dvi

??"Hensel lifts"@
An example: \( p \)-adic square root

\[ \text{sqrt}(2+0(7^{30})) \]

\text{install}(\text{Zp_sqrtlift}, \text{GGGL})

\text{Zp_sqrnlift}(2,3,7,30)
How does install(name, code) work?

- it opens the running gp program, as loaded in memory: `dlopen(NULL,)` (exposes all of libpari);
- it looks for a symbol matching the name: `dlsym(, name)`, returns the address of some machine code in memory;
- it associates a “prototype” to the symbol (expected arguments and return type), and records this data in the parser table.
- from that point on, a new GP function is available, to call a libpari function as if it had been built-in into the interpreter.
How does install(name, code) work?

- it opens the running gp program, as loaded in memory: `dlopen(NULL, )` (exposes all of libpari);
- it looks for a symbol matching the name: `dlsym(, name)`, returns the address of some machine code in memory;
- it associates a “prototype” to the symbol (expected arguments and return type), and records this data in the parser table.
- from that point on, a new GP function is available, to call a libpari function as if it had been built-in into the interpreter.

N.B. We can load symbols from other libraries, and give them arbitrary names in GP

```c
install(big_factors_C,"GGG","issmooth","./libbig_factors.so");
```
How does install(name, code) work?

- It opens the running gp program, as loaded in memory: `dlopen(NULL,)` (exposes all of libpari);
- It looks for a symbol matching the name: `dlsym(, name)`, returns the address of some machine code in memory;
- It associates a “prototype” to the symbol (expected arguments and return type), and records this data in the parser table.
- From that point on, a new GP function is available, to call a libpari function as if it had been built-in into the interpreter.

N.B. We can load symbols from other libraries, and give them arbitrary names in GP:

```
install(big_factors_C,"GGG","issmooth","./libbig_factors.so");
```

From a user’s point of view, this can remain black magic. The only difficulty is to provide the correct prototype: it can (mostly) be inferred from the C prototype, as documented in `libpari.dvi`. 
Prototypes (simplified)

- First character \( i, l, v \) : return type int / long / void. (Default: GEN)

- One letter for each mandatory argument: G (GEN), & (GEN*), L (long), n (variable)

- \( p \) to supply realprecision, \( P \) to supply seriesprecision.

- Special constructs for optional arguments and default values:
  - DG (optional GEN, NULL if omitted),
  - D& (optional GEN*, NULL if omitted),
  - Dn (optional variable, \(-1\) if omitted),

**GEN Zp_sqrtlfft(GEN b, GEN a, GEN p, long e) \( \rightarrow \) GGGL**