PARI-GP Tutorial. Math 791N, Fall 2009

1. Introduction & Installation

In this handout we give a quick summary on how to use the *free software* PARI/GP. PARI/GP is the front-end of a very powerful package of *C* library tailored for arbitrary precision integer arithmetic with loads of number-theoretic functions. It is developed by Professor Henri Cohen *et al* at Université Bordeaux I. It is widely used all over the world, not only in universities, but also in government agencies (e.g. NSA) and even in industries (rumors has it that Industrial Lights & Magics uses PARI).

Compare to commercial/industrial-strength softwares like Words and Firefox, the user interface of PARI/GP is noticeably more primitive. But based on my teaching experience the learning curve is not very steep, and PARI/GP really is lots of fun! Also, it comes with a very detailed manual, plus a tutorial which contains far more information that we will ever need in this case. The goal of this writeup is to help you get up to speed as quickly as possible. You might want to *lightly* skim through the first 20 pages or so. But just like as any program, reading the manual is not a substitute for actually using PARI. So roll up your sleeves and play! Warning: it's addictive!

Download Freewares

PARI-GP is available on several platforms:

(i) UNIX
go to `http://pari.math.u-bordeaux.fr/download.html` and build it yourself: get `pari-2.3.4.tar.gz`

(ii) Window
go to the PARI page above and get the two **self-installinging zip file**: `Pari-2-3-4.exe`

(iii) Macs:
I have no experience whatsoever with Mac, but I was told that the thing to do is to download a program called `fink` which installs UNIX program onto Macs:

`http://fink.sourceforge.net`

Follow the instruction there to download `fink` *and* the two files for the Unix platforms (see (i) above). Once `fink` is installed, type

```
su apt-get install gp-pari
```

into a terminal and 'everything should work' – so I am told!

Note:

- The UNIX tar file comes with documents; for other platforms make sure you get the manuals (see the course website). As always: read the **README** files before you do anything!
- The PARI webpage list several optional but very useful additional supplements. Three very useful ones are
  - `PariEmacs` (listed under the 'Extras' section near the bottom of the PARI webpage): this allows you to run PARI inside Emacs, making it easy to the output of long calculation
  - `Readline` (listed under the 'Extras' section): it allows you to edit commands using the usual UNIX control sequences,
  - various *Optional Packages* (listed near the top of the webpage), such as `elldata.tgz` (elliptic curves data), `galdata.tgz` (data for computing high degree Galois groups), `nftables.tgz` (number fields data).

2. Using PARI as a calculator

A few generic comments before we start:

- To load a file into PARI:
  inside the PARI window, type `\r (name of file)`
  Note: the file you want to load must be in the same directory in which you run PARI.
- To quit PARI:
  inside the PARI window, type `\q`

A few things to keep in mind:

- PARI handles *arbitrary precision integers/rational arithmetic.* In order words, a fraction (say) 2/3 is kept as a fraction, as oppose to an infinite decimal 0.66666666666666 (the default floating point accuracy is 28 decimal places)
- if you actually want the decimal, type
  `2./3` or `2/3.` or `2/3 * 1.` etc
- variable names are case sensitive, so `Big, BIG, biG` are all different variable names
- to declare a variable you simply set
  `blah = 3`
- the label `I` has been reserved to denote the usual square-root of -1 !! So
  `I*I`
  will return `-1.` Similarly, `Pi` denotes what you think it is (to 28 decimal places), but neither `i` , `E` , nor `e` has been reserved.

Some basic data-type:

- Vector:
  `v = vector(10,j,j^2)`
  This command defines a vector of size 10 **and** initializes it by setting the j-th component to be j^2. To access the k-th coordinate, type
  `v[k]`
  NOTE: the first coordinate of a vector `v` in PARI is `v[1]`
- Matrix:
  `m = matrix(5,7,j,k,j+k)`
  This command defines a 5×7-matrix, i.e. a matrix with 5 rows and 7 columns, **and** initializes it by setting the (j,k)-component to be $k+j$. To access the (j,k)-th coordinate, type
  `m[j,k]`
- Congruence Arithmetic
  To work with 3 (mod 7), type
  `m = Mod(3,7)`
- Polynomials:
  there are two ways to define a polynomials:
  ```
  f = x^3 + 2*x*y - z
  g(x, y, z) = x^3 + 2*x*y - z
  ```
  the first line simply defines a object of the type 'polynomial', but you cannot **evaluate** f. The second method actually specifies a **user-defined function** – more on that later – which just happens to be a polynomial

NOTE #1. As a general rule, PARI always display the result of any command. Sometimes that might NOT be desirable; for example,

`v= vector(10000,j,0)` ⟵ you DO NOT want to see the output!

To **suppress** the display of a calculation, put a **semicolon** at the end of the command:

`v= vector(10000,j,0);` ⟵ this is better!

NOTE #2: the Vector and Matrix data type in PARI are *very flexible*: you can put just about anything in the coordinates! For example,

```
        v = [x^3 - 3*y*z, Mod(1,7), 3/7, matrix(2,2,j,k,j+2*k) ]
```

NOTE #3: To access PARI's online help, type (inside PARI)

```
        ?  (command)   for a specific command;
        ?  (command)   for the general help menu
```

Output:

- first, PARI *remembers* the answers of all your calculations:
  ```
  ?  (x+2)*(x+6)
  %6 = x^2 + 8*x + 12
  ?  4+5
  %7 = 9
  ?  %6
  %8 = x^2 + 8*x + 12
  ```
  NOTE: '%6' refers to the answer of the sixth (correct) calcuation (PARI does not count the incorrect ones). The only reason we have '6' here is because I happen to have done 5 calculations before.
- if you specifically want PARI to display a certain variable/value, you can use the `print` comment:
  ```
  ?  print(4+5)
  ?  9
  ```
  Since this is NOT a calculation, this value is not stored by PARI; in particular, there is no '%' sign.
  If you need to print out more than one thing at a time, separate the quantities with comma:
  ```
  ?  print("First and second value:  ",4," ",5)
  ?  First and second value:  4 5
  ```
  Beware of the double quotes; otherwise you could get misleading output:
  ```
  ?  print("this time, no double quotes:  ",4,5)
  ?  this time, no double quotes:  45
  ```
  There are additional printing comments, such as `print1` and `printp`; I will leave these to you to explore on your own.

Additional Info

- you can change adjust the number of significant digits of floating-point calculations:
  ```
  ?  \p 50
  realprecision = 57 significant digits (50 digits displayed)
  ```
  Note that this is **not** retroactive: if you want to increase the number of significant digits of an earlier calculation you must do it again.
- Every once in a while PARI would complain that it runs out of memory and/or the list of precomputed primes. If you run PARI under UNIX, you can increase the number of precomputed primes by
  ```
  gp -p [primelimit]
  ```
  And you can increase the *initial* stack size using
  ```
  gp -s [initial stack size]
  ```
  (the default stack size is 4000000, and the default prime limit is 500000). Note:
  - Of course you can combine these two commands.
  - If you set too high a primelimit, it would take a while before you can fire up PARI.
  - How high an initial stack you can set depends on how much memory you have in your memory; if you set it too high you would end up swapping memory in and out of your hard drive, which is *extremely* slow. For normal daily calculation, the initial stack size is more than adequate.

3. PARI Programming

for-loop:

The following example computes the first 10 Fibonacci numbers:

```
f=vector(10,j,0)
f[1]=0
f[2]=1
for(k=3,10, f[k] = f[k-1] + f[k-2])
```

Note that PARI does NOT display the output of the FOR statement. For that you can use, for example,

```
for(k=1,10, print(f[k]) )
```

or better yet,

```
for(k=1,10, print(k,“ ”,f[k]) )
```

Note the double quotes!

Before we can describe the syntax for `if-then-else`, `while` and `until` we need to make a brief digression about logical statements. The symbols

<   >   == ⟵ this is TWO equal signs put together

are **comparison operators** – given two quantity `x` and `y` the statement

`x>y`

is either true or false.[1] If it is true then PARI returns the value 1, otherwise, PARI returns the value 0. Examples:

```
? x=3
%1 = 3
? y=5;
%2 = 5
? x>y
%3 = 0
? x<y
%4 = 1
? x==y
%5 = 0
```

We can now state the syntax of the three constructions:

while

The following example computer $n!$ (that's n-factorial)

```
k=1;
while(n>0, k=k*n; n=n-1);
print(k)
```

until

n-factorial again, but using **until**:

```
k=1;
until(n=0, k=k*n; n=n-1);
print(k)
```

if-then-else:

[1]assuming `x, y` can be compared – otherwise PARI returns an error message

```
if(x==y,print("Same!"))
```

If-then-else:

```
if(x==y,print("Same!"), print("Different!"))
```

If you need to consider more than one logical statement, use the **and operator** && as well as the **or operator** ||:

```
if(x>0 && y>0, print("Both positive!"), print("Not so!"));
if(x>0 || y>0, print("At least one positive!"), print("Not so!"));
```

User-defined functions:

More factorial, this time recursive:

```
f(n) = if(n<0,print("Invalid input!"),if(n==0,return(1),return(n*f(n-1))))
```

As you can see, pretty soon these expressions, especially the parenthesis, are impossible to keep track of. It would be a lot easier to write your functions and/or programs in a separate file and load them into PARI. You can:

First, write whatever you need to write in a TEXT editor (for Windows machines: use Notepad) *without* firing up PARI. And then save the file. For example, say you have the following file, called fact.gp:

```
f(n) =
{   if(n<0,
       print("Invalid input!"),
       if(n==0,
          return(1),
          return(n*f(n-1))
       )
    )
}
```

See how much more readable! Now you can load this into PARI using the command (inside PARI)

```
\r fact.gp
```

Before I forget, let me mention one peculiar odd feature of PARI programs: when you are done writing your programs above, enclosed by a pair of braces { ··· } , **remember to hit the return (or enter) key a couple times**; without these extra blank lines at the end of the program, PARI sometimes would complain about errors in your codes even though your codes are perfectly correct.

One last thing, about **local variables**. Look at this program:

```
fun(x) =
{
   y=x*x;
   if(y>4,
      print("big"),
      print("small")
   );
}
```

If you run this program, the output would look like this:

```
\r test.gp
x=5
%1=5
y=2
%2=2
print("y= ",y);
y= 2
print(fun(x));
big
print("y= ",y);
y= 25
```

Note that the value of y has changed! By now you probably have figured out what happened: we use the variable y in the function `fun`, and *this function* changes the value of y. To avoid problems like this, insert the line

```
    local(y);
```

right after the first brace { inside `fun` and PARI will treat the variable y inside `fun` as a *local variable.*