BEGINNING PARI PROGRAMMING FOR CSE/MATH 467

W. DALE BROWNAWELL

1. Installation

You can follow the instructions in the links for various platforms (including smartphones(!)) given by RosettaCode `rosettacode.org/wiki/Category:PARI/GP`, which also gives links to instructions for handy interfaces in which to edit code, run code, and save output.

2. Beginning a Pari Session

2.1. **Using Pari's Graphical Interface.**
There is none. I find it convenient to use PariEmacs. The newer interfaces may be just as good; I just have not tried them.

2.2. **Installing the Pari-Emacs Interface.**
If you run a fedora-based version of linux, you will find it easy to install and run emacs, pari-gp, and pari-emacs using rpm's in the EPEL (Extra Packages for Enterprise Linux) repository once you have enabled the repository.

If you run a debian-based system, you can install emacs and pari-gp using your package manager. But, last time I checked, one still had to compile pariemacs oneself. The site
`http://math.univ-lille1.fr/~ramare/ServeurPerso/GP-PARI/`,
now has very complete instructions on how to do that, which I have followed for my home ubuntu machine.

In fact, the instructions on Ramaré's site also apply if you have succeeded at the more ambitious task of compiling pari-gp on a less common operating system.

2.3. **Falling into a Terminal State.**
If you start pari/gp from a command line with the command `gp`, you will have only a command-line, non-graphics, version.

3. Ending a Pari Session

To end a session, type `\q` or type `quit` at the prompt, which is %.

Date: 6 September, 2016.

4. Writing and Entering Programs

4.1. **Writing Pari Programs.** Pari programs are best stored in files with names like: `gcd.gp`. These files are plain ascii files created by your texteditor and saved by you. You may store several, indeed all, your functions in one file.

4.2. **Loading Pari Programs.** The programs we will write are actually all definitions of functions which take the input, usually numbers or vectors (lists) of numbers, and produce some output, again usually numbers or vectors of numbers, using various other functions, some of which we have to define as well.

The functions in a file are made available in a pari session via the pari prompt as in

```
% \r filename
```

where `%` is the prompt, `\r` is the read command and `filename.gp` is name of your file. The functions defined in the file will then remain available to you during the whole session. So it is important to have distinct names for distinct functions.

4.3. **Local Variables in Pari.** Because Pari remembers anything already used or defined in a session, you need to keep dummy variables involved in defining functions free from the influence of any previous use of the variables. This is done by using *local variables*.

In the following example, `x,y,t` are the local variables:

GGT.gp

```
GGT(m,n) = { local(x,y,t);   x=abs(m);y=abs(n);
                          while(y > 0,
                               t = y;
                               y = x%y;
                               x = t
                               );
                               print(x)
           }
```

Notice the following features:

- Parentheses ( ) are used for grouping mathematical expressions and for programming and user function evaluation, braces { } are used for definitions of gp-scripts, and, as we will see later, square brackets [ ] will be used to refer to entries of matrices M as $M[i,j]$ or of vectors V as $V[i]$.
- The values to be fed into the function are used in the program via local variables.
- The local variables must be declared immediately after the opening parenthesis {. They can be given a value immediately if their value is known or delayed until they are needed.

- The punctuation of a pari/gp command is usually a comma or a semicolon. Generally comma use is much more restricted.

5. Helpful Hints

5.1. **Putting Several Functions into a Single File.**

It's often a good idea so you don't have to remember later what functions are used in the definition of a function you haven't seen in a couple of weeks. You can also organize your functions this way by grouping related functions together into a single .gp file. The best reason is perhaps that then you don't have to enter all the .gp files singly by hand at the pari prompt!

There is another way to keep all your .gp files in one directory and still to allow functions to rely on other functions. If the definition being relied upon occurs in say, prelim.gp, you simply include a line `read(”prelim.gp”)` in your new function before the earlier function is evaluated. Of course, punctuation such as semicolon may be necessary at the end of this line. Notice that the quotation marks seem to be required.

5.2. **Many Happy Returns.**

When Bressoud uses `WRITE` or `PRINT`, it is better for us to `return` so the output can be used as input for another function if desired. However, for debugging purposes, you may want to use the `print` function.

5.3. **Getting Hardcopy.**

I prefer to run everything within Emacs, whence you can save copies of files and of sessions, as well as printing them without leaving the buffer. But that will involve getting pariemacs installed.

6. A Short Pascal/REXX to Pari Dictionary

Bressoud	**Pari**
`READ a,b`	`... x=a; y=b; ...`
`WHILE b` $\neq$ `0 DO`	`while(y != 0, ... ,`
`temp` $\leftarrow$ `b`	`temp = b`
`WRITE a`	`return[a]`
$u_1 \leftarrow 1$, $u_2 \leftarrow 0$, $u_3 \leftarrow a$	`u = [1,0,a]`
$q \leftarrow \lfloor u_3/v_3 \rfloor$	`q = floor(u[3]/v[3])`
`IF x = 0 THEN DO`	`if(x == 0, ...`
`IF x` $\neq$ `0 THEN DO`	`if(x != 0, ...`
`IF x is even DO`	`if(x%2 == 0, ...`
`IF b is odd THEN`	`if(b%2 == 1, ...`
`c` $\leftarrow$ `|a-b|`	`c = abs[a-b]`
`FOR i = 2 to n DO`	`for(i = 2, n, 1; ...`
`FOR i = 1 to n DO`	`a = vector(n,i,i)`

`a`$_i \leftarrow$ `i`	
`FOR i = r-1 to 1 by -1 DO`	`for(i = 1, r-1, 1;`
`a` $\leftarrow$ `a` $\times$ `m`$_i$ `+ w`$_i$	`a = a*m[r-i] - w[r-i]`
`f MOD d`	`f%d`
r^i `MOD n`	`r = lift(Mod(`r^i`,n))`
`i` $\leftarrow$ `i+1`	`i = i+1` or `i++`
`AND`	`&&`
`OR`	`||`

Beware: The meaning of `Mod` for gp is **completely** different from the command for Mathematica.

7. Arrays to the Finish

Multiple outputs are often best collected and output as a single list, as for **Algorithm 2.3** and **Algorithm 2.4** (a list of pairs in the latter case), whose entries can then be utilized in further programs.
Pari/gp distinguishes between lists and vectors, as well as matrices. Confusingly (for me), row vectors are not simply $1 \times n$ matrices.

7.1. **Makin' a list ...**

`L = listcreate(n)` creates an empty list named L into which we can add up to n entries. One can

- append the entry `x` to the list `L` using the command `listput(L,x)`,
- change the kth entry of the list `L` to the value x by setting $L[k] = x$ or
- (dangerous) insert `x` into the position i and move the following entries down on the list with the command `listput(L,x,i)`.
- `listkill(L)` deletes all entries, so that `L` is empty again, but it does not remove the empty list.

After a list `L` has been created, its entries can be obtained as `L[3]`, say, for the third entry. (Note the square brackets here.)

The (i, j)th entry of the matrix M can be accessed simply as $M[i, j]$. The ith row of the matrix `M` can be accessed as `M[i,]` and the jth column as $M[, j]$. An $n \times n$ identity matrix is created by the command `matid(n)`.

7.1.1. *Vectors and Matrices.*

If $f(x)$ is a polynomial, the command

$$\mathtt{vector}(\mathtt{n}, \mathtt{i}, \mathtt{f}(\mathtt{i}))$$

creates a (row) vector of length n with ith entry equal to $f(i)$. The function `vectorv` creates a column vector. One can switch between row and column vectors by transposing with a trailing superscript ˜. That is,

$$\begin{pmatrix} a & b & c \end{pmatrix}^{\sim} = \begin{pmatrix} a \\ b \\ c \end{pmatrix}.$$

If $f(i,j)$ is a function of the two variables i, j, the command

$$\mathtt{M} = \mathtt{matrix}(\mathtt{m}, \mathtt{n}, \mathtt{i}, \mathtt{j}, \mathtt{f}(\mathtt{i}, \mathtt{j}))$$

creates an m by n matrix M whose (i,j)th entry is $M[i,j] = f(i,j)$. Once M is created, its values can be changed by setting $M[i,j]$ equal to whatever you want, and similiarly for vectors.

If M is a matrix or a vector, `mattranspose(M)` will create its transpose.

7.1.2. *Append.* Appending in pari appears a bit arcane because of the distinction between row vectors and $1 \times n$ matrices. See section 3.8.2 of the User's Guide for an explanation of **concat**(x,{y}). Pari seems not to consider matrices to simply be vectors of vectors. Henri Cohen is a **very** bright mathematician and programmer. So he undoubtedly had good reasons for this approach.

concat takes matrices or vectors with the same number of rows and places them side-by-side to produce new matrices with the same number of rows. Using transposes, one can stack matrices with the same number of columns on top of each other. But one cannot simply do this with vectors as $1 \times n$ matrices. Taking transposes of row vectors and concatenating will simply produce the transpose of the concatenation of the original row vectors. Instead one must first convert at least one of the vectors to be a matrix.

Here is a simple example of how one can go about doing it.

```
?  a = [1, 2]
% 1 = [1, 2]
?  b = Mat(a)
% 2 = [1, 2]
?  c = [3, 4]
% 3 = [3, 4]
?  d = concat(b~,c~)~
% 4 = [1, 2; 3, 4]
```

7.2. **Common pitfalls.** Once the vector `v` has been defined, its jth entry can be $\underline{\text{re}}$-defined by, e.g., `v[j] = 2 + i`. But the initial definition of a list apparently CANNOT be made this way! Similarly for matrices.

"[E]mbedded braces (in parser) is not yet implemented." That means that if you want to use a component of a vector value in a function and that vector has been defined by a subroutine, you have to give the vector a name (via a local variable) and then input the component. For example, if `subr(x,y)` returns a vector and you want to add the second component to the list `l`, you need to use

$$\mathtt{v} = \mathtt{subr}(\mathtt{x}, \mathtt{y});$$
$$\mathtt{listput}(\mathtt{l}, \mathtt{v}[2])$$

and not "simply" ~~`listput(l,subr(x,y)[2])`~~.

7.3. **Useful commands.**

- If `L` is a list, then `length(L)` returns the length of the list.
- To create a vector `V` from a list `L` of length `n`, we set

$$\mathtt{V} = \mathtt{vector}(\mathtt{n}, \mathtt{i}, \mathtt{L}[\mathtt{i}])$$

- `random(n)` selects a "random" number less than n.
- `bin(n)` gives a vector whose entries are the binary representation of n.

8. Logging Output from Sessions

`\l filename` logs output to the named file. This may be useful, for example if you can't run everything in a single session. However accessing the entries later can be challenging, Cf. Input/Output from the PARI/GP faq.

9. Timing

You can find out how much time various parts of your program are taking.
# starts and stops timing for processes in a program.
## displays the time logged.

10. Reduced Precision for Faster Sieving

You can suppress the infinite (well–very great) precision by setting, say, `\p 5` This truncates the precision to 5 decimal places. This is useful to speed sieving computations up for example, since we use only an estimate of sums of $\log(p)$ for primes p to the locations r for which $r \equiv \pm t_p \mod p$, where $t_p^2 \equiv n \mod p$.

11. Matrix Reduction

For the quadratic sieve, you need to row reduce a huge matrix, say `M`, modulo 2.

If `M` is a matrix, the command `matimage(M)` will calculate a basis for the column space. Of course we can calculate a basis for the row space via

```
mattranspose(matimage(mattranspose(M))).
```

Since we are interested in it only modulo 2, we should first reduce M mod 2 via, say, `N = Mod(M,2)` and then choose representatives with the command `lift`. Although

```
lift(mattranspose(matimage(mattranspose(Mod(M,2)))))
```

should work for row reduction, I might break things down into smaller steps.

12. Documentation

You are required to document, i.e. explain, your programs. You can do this by putting comments into your program by beginning with /* and ending with */. Pari will ignore what lies in between. Or you can simply write (in English sentences, please) explanations neatly in the margin by hand.

13. Calling PARI's APIs from Other Languages

Perl:	`http://search.cpan.org/ ilyaz/Math-Pari-2.01080605/Pari.pm`
Python:	`https://code.google.com/archive/p/pari-python/`

14. Corrections

I have incorporated several helpful suggestions from Danesh Forouhari. I would be thankful for any additonal corrections or comments. Please send them to me at `wdb@math.psu.edu`.

15. Source

The home for PARI/GP is
`http://pari.math.u-bordeaux.fr/`
The most comprehensive source of information is
`http://pari.math.u-bordeaux.fr/doc.html`, with links to the

- User's Guide to PARI/GP (It's quite large, so I wouldn't just print it.)
- Tutorial (52 pages)
- Reference Card (4 pages).

It also has information on installation and on gp2c, which seems to be a program for converting gp code to c.

Other nice introductions can be found at

- R.B. Ash, A PARI/GP Tutorial,
  `http://www.uiuc.edu/r.ash/GPTutorial.pdf`
- S. Wong, PARI/GP Tutorial,
  `http://www.umass.edu/siman/09.791N/tutorial.pdf`

E-mail address: `wdb@math.psu.edu`