Pequena introdução ao PARI/GP

Paula Cristina Valença

February 26, 2007

1 Instalação

O **PARI/GP** é um sistema de algebra computacional particularmente indicado para computações na área de Teoria de Números e, consequentemente, para a resolução de pequenos problemas na área de Criptografia. Além das bibliotecas, contém um interpretador, `gp`, e uma linguagem de script, `gp-script`.

A página oficial é `http://pari.math.u-bordeaux.fr/`.

Para instalar o programa da distribuição de *source*, faça:

1. *Download* da última versão `pari-X.X.X.tar.gz`
2. `tar zxvf pari-X.X.X.tar.gz; cd pari-X.X.X`
3. `./Configure --prefix=$HOME/pari` (default: `/usr/local/`)
4. `make all; make install; make test-all`
5. `export PATH=$HOME/pari/bin:$PATH`

Se usa o Emacs, a distribuição inclui modos de suporte para o PARI/GP. Consulte a documentação e o ficheiro `pariemacs.txt` para mais informação.

A distribuição inclui também documentação, em particular, um manual de utilizador [BBB+b], um tutorial [BBB+a] e uma *Reference Card* [RC]. Esta última será de grande utilidade no futuro. Robert B. Ash escreveu também um tutorial do Pari/GP [Ash07].

2 Tipos do PARI e comandos básicos

Para correr, chame num terminal o comando `gp`. Para terminar, escreva `quit` ou `\q`. Para ajuda numa função ou palavra (e.g, `foo`), `?foo` ou `??foo`. O último output pode ser chamado através de `%`, `%n` para a linha `n`. O carácter `#` acciona ou desacciona o temporizador. Finalmente, `\r fich` lê um ficheiro `fich` (mais tarde importaremos pequenos programas GP desta forma).

Há quinze[1] tipos fundamentais no PARI, enumerados a seguir. Concentraremonos nos oito primeiros[2]. Esta secção segue de perto [Ash07] e [BBB+b], secção 1.2.

1. `t_INT` - $\mathbb{Z}$ - **Inteiros**

   De precisão ilimitada, introduza o inteiro, opcionalmente precedido de um $+/-$ como sinal.

2. `t_REAL` - $\mathbb{R}$ - **Reais**

   De precisão ilimitada, introduza um . (e casas decimais) para sinalizar que é real. A precisão fixada é 28 digitos e pode ser alterada com o comando `\p` ***n***.

3. `t_FRAC` - $\mathbb{Q}$ - **Racionais**

   Representados por dois componentes na sua forma reduzida, n/m (24/18 é traduzido para 4/3).

4. `t_INTMOD` - $\mathbb{Z}_n, \mathbb{Z}/n\mathbb{Z}$ - **Inteiros Modulares**

   Representados por dois componentes, o módulo m e o número representante n, $Mod(n, m)$. A secção 4 debruçará um pouco nos inteiros modulares.

5. `t_COMPLEX` - $T[i]$ - **Complexos**

   Introduzidos da forma $a + b * I$, a e b podem ser de tipo `t_INT`, `t_REAL`, `t_FRAC`, `t_INTMOD` ou `t_PADIC`.

6. `t_POL` - $T[X]$ - **Polinómios**

   Os coeficientes podem ser de qualquer tipo, a/as variável/eis podem tomar qualquer nome e são introduzidos da forma usual ($*$ não pode ser omitido: $x^2 + 4 * x + 4$, e.g.).

7. `t_MAT` - $M_{m,n}(T)$ - **Matrizes**

   Os elementos podem ser de qualquer tipo e são introduzidas separando os elementos por virgulas e as linhas por ponto e virgula, e.g.,

$$[1, 1/2, 1/3; 1/2, 1/3, 1/4; 1/3, 1/4, 1/5] = \begin{pmatrix} 1 & \frac{1}{2} & \frac{1}{3} \\ \frac{1}{2} & \frac{1}{3} & \frac{1}{4} \\ \frac{1}{3} & \frac{1}{4} & \frac{1}{5} \end{pmatrix}$$

   Para uma matriz A, $A[i, j]$ devolve o elemento na posição ij, $A[i,]$ devolve a linha i e $A[, j]$ devolve a coluna j.

[1] na realidade há ainda um 16º tipo, `t_QFI/t_QFR`, que se refere a formas quadráticas e que está sujeito a desaparecer em versões futuras

[2] no entanto, encontrará por certo referências aos outros tipos na literatura na área de Criptografia

8. `t_VEC/t_COL` - T^n - **Vectores**

   Os elementos podem ser de qualquer tipo. Um vector de linha toma a forma $[1, w, w^2]$, por exemplo. Um vector de coluna distingue-se por um `~` no fim, $[1, w, w^2]$`~`, por exemplo. Para um vector v, $v[i]$ dá o elemento na posição i

9. `t_LIST` - T^n - **Listas**

   Criadas ou através da função `listcreate(`n`)` (cria uma lista vazia de tamanho máximo n) ou da função `List(`v`)` (converte o vector ou lista v numa lista ou cria uma lista vazia se v é omitido).

10. `t_STR` - **Strings**

11. `t_QUAD` - $\mathbb{Q}[w]$ - **Corpos Quadráticos**

12. `t_PADIC` - $\mathbb{Q}_p$ - **P-ádicos**

13. `t_POLMOD` - $T[X]/P(X)T[X]$ - **Polinómios mod P(X)**

14. `t_SER` - $T((X))$ - **Séries de potências**

15. `t_RFRAC` - $T(X)$ - **Funções Racionais**

3 Exemplo: Primalidade e Factorização

Os problemas de primalidade e factorização ocupam uma posição primordial na área de Criptografia, nomeadamente, criptografia de chave pública. O seguinte exemplo é um "aperitivo" para apresentar algumas funções e propriedades úteis.

Recorde que `?`*`comando`* e `??`*`comando`* apresentam uma descrição de *`comando`*.

Considere o número $n = p * q$, produto de dois primos p e q,

```
gp > p = nextprime( 2^100 ); q = nextprime( p + 1 ); n = p*q
```

O Teorema Fundamental da Aritmética garante que n tem uma factorização única (a menos da ordem). Para simplificar, sabemos à partida qual é ($n = pq$) mas vamos supor por um momento que não sabemos. Mais, nem sequer sabemos se n é ou não um número primo.

Comecemos por um método rude: testar se é divisível por algum número até 20000. Primeiro, ligamos o *timer*.

```
gp > #
gp > factor(n, 20000)
```

Como explica o resultado? E por que razão continuar a tentar dividir por todos os números até $\sqrt{n}$ não é uma boa estratégia?

O Pequeno Teorema de Fermat diz que se n é primo e n não divide a ($n \nmid a$), $a^{n-1} = 1 \mod n$. A implicação inversa nem sempre se verifica mas o teorema dá-nos uma técnica simples para testar se n não é primo. O comando `isprime` confirma que, realmente, n não é primo. Note o tempo que o GP demorou a calcular o `isprime`.

```
gp > Mod(2, n)^(n-1)
gp > isprime(n)
```

Note agora o tempo necessário para calcular os factores de n (os algoritmos usados são optimizados).

```
gp > factor(n)
```

Como curiosidade, corra agora a seguinte sequência de instruções,

```
gp > { u = nextprime(sqrtint(n));
i = 1;
while(i < 100 && n%u, u = nextprime(u); i = i + 1);
u
}
gp > u == q
gp > q - p
gp > #
```

Esta permitiu-nos descobrir os factores de n quase imediatamente. Isso deve-se à forma particular de p e q: p e q são primos próximos, $q - p = 54$, o que leva a que $n - p^2 = 54p$, isto é, p e q estão próximos de $\sqrt{n}$ ($p < \sqrt{n} < q = p + 54$).

4 Do Algoritmo de Euclides à Aritmética Modular

O PARI/GP permite alguma programação básica[3], com base nas estruturas condicionais e de ciclo tradicionais. No que se segue, crie e edite um ficheiro com extensão `.gp` e importe no GP através do comando `\r` *`fich`*.

Por exemplo, a função factorial pode ser definida da seguinte forma,

[3] aliás, é possivel extender a biblioteca do PARI e importar funções exteriores. Ver [BBB+c] para mais informação

```
\\ exemplo simples da definicao de uma funcao em GP
myFact(n) =
{ local(fact);

  if( type(n) != "t_INT" || n < 0,
      return(-1));
  fact = 1;
  while( n > 0,
         fact = fact*n;
         n = n - 1;
  );
  fact
}
```

Com base neste exemplo, escreva uma função que calcule o maior divisor comum (gcd) de dois inteiros não negativos através do algoritmo de Euclides,

Algoritmo 1 Algoritmo de Euclides

Input: dois inteiros n e m não-negativos
Output: o maior divisor comum de n e m, $\gcd(n, m)$
  while $m \neq 0$ **do** [Terminou?]
    $r \leftarrow n \mod m$ [Passo Euclideano]
    $n \leftarrow m$
    $m \leftarrow r$
  end while
  return n

Estime a complexidade do algoritmo de Euclides[4].

Se $\gcd(n, m) = d$ então existe u e v tal que $un + vm = d$. Extenda a sua função de forma a calcular u e v segundo o algoritmo 2.

Compare o resultado da sua função com o da função `bezout`.

```
gp > a = 2^101+1; b = 2^237+2^5+1;
gp > myXGCD(a, b)
gp > bezout(a, b)
```

Para além das propriedades subjacentes ao algoritmo 1, a versão extendida (2) é uma das formas mais eficientes de calcular o inverso, quando existe, de um inteiro modular.

[4]Knuth [Knu98] prova que para n e m uniformemente distribuídos entre 1 e N, o número de passos é, no máximo, $\lceil (\ln(\sqrt{5}N))/(\ln((1+\sqrt{5})/2)) \rceil - 2 \approx 2.078 \ln N + 1.672$ e é, em média, aproximadamente $12 \ln 2/\pi^2 \ln N + 0.14 \approx 0.843 \ln N + 0.l4$

Algoritmo 2 Algoritmo de Euclides extendido

Input: dois inteiros n e m não-negativos
Output: uma lista $[u, v, d]$ tal que $d = \gcd(n, m)$ e $un + vm = d$

$u \leftarrow 1, d \leftarrow n$ [Inicialização]
if $m = 0$ **then**
  $v \leftarrow 0$
  return $[u, v, d]$
else
  $v_1 \leftarrow 0, v_3 \leftarrow m$
end if
while $v_3 \neq 0$ **do** [Terminou?]
  $t_3 \leftarrow d \mod v_3$, $q \leftarrow \lfloor d/v_3 \rfloor$ [Passo Euclideano]
  $t_1 \leftarrow u - qv_1, u \leftarrow v_1, d \leftarrow v_3, v_1 \leftarrow t_1, v_3 \leftarrow t_3$
end while
$v \leftarrow (d - nu)/m$
return $[u, v, d]$

Para um dado inteiro positivo n, e um inteiro a, $(a \mod n)$ pode ser interpretado como representando o resto positivo da divisão inteira de a por n. Com base nisso, $a = b \mod n$ se e só se $n|(a - b)$.

Para qualquer a, o resto r da divisão inteira de a por n obedece $0 \leq r \leq n-1$ e $(\mod n)$ define assim um conjunto com n elementos. Adicionalmente operações de adição e multiplicação derivam imediatamente da definição,

$$(a \mod n) \oplus (b \mod n) := (a + b) \mod n$$
$$(a \mod n) \otimes (b \mod n) := (a \times b) \mod n$$

Este conjunto equipado das operações de adição e multiplicação acima definidas denotamos por $\mathbb{Z}_n$ ou $\mathbb{Z}/n\mathbb{Z}$. No PARI/GP, elementos deste tipo são representados da forma `Mod(a, n)`.

Note que se $\gcd n, m = 1$, existem u e v tal que $un + vm = 1$ e, dada a discussão acima, $vm = 1 \mod n$ (dado que $vm - 1 = un$). Isto é, $(v \mod n)$ é o inverso de $(m \mod n)$.

```
gp > c = Mod(myXGCD(a, b)[1], b)
gp > d = Mod(a, b)^-1
gp > print("Da o mesmo resultado? ", if(c == d, "sim", "nao"))
```

Teste para $a = 5$. O que acontece? Porquê?

References

[Ash07] Robert B. Ash. *A Pari/GP Tutorial*. `http://www.math.uiuc.edu/~r-ash/GPTutorial.pdf`, Jan. 2007.

[BBB+a] C. Batut, K. Belabas, D. Bernardi, H. Cohen, and M. Olivier. *A Tutorial for PARI/GP*. version 2.3.1.

[BBB+b] C. Batut, K. Belabas, D. Bernardi, H. Cohen, and M. Olivier. *User's Guide to PARI/GP*. version 2.3.1.

[BBB+c] C. Batut, K. Belabas, D. Bernardi, H. Cohen, and M. Olivier. *User's Guide to the PARI library*. version 2.3.1.

[Coh96] Henri Cohen. *A Course in Computational Algebraic Number Theory*. Number 138 in Graduate Texts in Mathematics. Springer, 1996.

[Knu98] Donald E. Knuth. *Seminumerical Algorithms*, volume 2 of *The Art of Computer Programming*. Addison-Wesley, third edition, 1998.

[RC] Pari-gp reference card. version 2.3.0.