Kurze Einführung in GP-Pari

Matthias Orth

Uni Kassel

14. April 2022

Download und Installation

Internetadresse: `https://pari.math.u-bordeaux.fr/download.html`

- Dort ausführbare Installationsdatei gemäß Betriebssytem herunterladen
- Nach Installation hat man, z.B. unter Windows, das Programm `gp.exe` und kann direkt starten.
- Der ordner `doc` enthält nützliche Hilfe-Dateien, z.B. Benutzerhandbuch `users.pdf` oder `tutorial.pdf`
- Wenn man `gp.exe` benutzt, kann man `?` und dann Eingabetaste drücken, um im Programm Hilfe aufzurufen.
- Mit `\q` und Eingabetaste kann man das gp-Programm verlassen. (q für quit.)

Ein erstes Programm

- Selbst geschriebene Programme kann man in Dateien mit Endung `.gp` speichern.
- Der Pari-Befehl `\r` dient zum Einlesen von `.gp`-Dateien. `\r` eintippen, dann die gewünschte Datei per Drag-and-Drop ins gp-Fenster ablegen.
- Der Dateipfad erscheint. Jetzt Eingabetaste drücken und das Programm wird ausgeführt.

Inhalt der Datei `code01.gp`

```
print("Hello World!");
```

Ausgabe von GP-Pari nach Lesen von `code01.gp`

```
Hello World!
```

Schleifen: `for`

Inhalt der Datei `code02.gp`

```
for(i=1,3,print("Hello World!"))
```

Ausgabe von GP-Pari nach Lesen von `code02.gp`

```
Hello World!
Hello World!
Hello World!
```

- In GP-Pari steht "oo" für das Symbol ∞. Was passiert, wenn man eingibt: `for(i=1,oo,print("Hello World!"))`?

Schleifen: `for`

Inhalt der Datei `code02.gp`

```
for(i=1,3,print("Hello World!"))
```

Ausgabe von GP-Pari nach Lesen von `code02.gp`

```
Hello World!
Hello World!
Hello World!
```

- In GP-Pari steht "oo" für das Symbol ∞. Was passiert, wenn man eingibt: `for(i=1,oo,print("Hello World!"))`?
- Tipp: Rechnungen kann man mit `Strg+C` unterbrechen.

Schleifen: Geschachteltes for

Inhalt der Datei code03.gp

```
for(i=1,3,for(j=1,3,print("Der groesste gemeinsame Teiler
    von "i" und "j" ist "gcd(i,j)".")))
```

Ausgabe von GP-Pari nach Lesen von code03.gp

```
Der groesste gemeinsame Teiler von 1 und 1 ist 1.
Der groesste gemeinsame Teiler von 1 und 2 ist 1.
Der groesste gemeinsame Teiler von 1 und 3 ist 1.
Der groesste gemeinsame Teiler von 2 und 1 ist 1.
Der groesste gemeinsame Teiler von 2 und 2 ist 2.
Der groesste gemeinsame Teiler von 2 und 3 ist 1.
Der groesste gemeinsame Teiler von 3 und 1 ist 1.
Der groesste gemeinsame Teiler von 3 und 2 ist 1.
Der groesste gemeinsame Teiler von 3 und 3 ist 3.
```

Schleifen: Spezielle `for`-Schleifen

- `forperm` iteriert über Permutationen einer Menge $\{1, \ldots, N\}$
- `forsubset` iteriert über Teilmengen einer endlichen Menge
- `forprime` iteriert über Primzahlen in einem Intervall

Schleifen: `while`

- `isprime` testet, ob eine Zahl prim ist.
- Mit `i++` bzw. `i--` wird die in `i` gespeicherte Zahl um 1 vergrößert bzw. verringert.
- (Kann genauso gut schreiben: `i=i+1` bzw. `i=i-1`.)
- `while`-Schleifen werden durchlaufen, *so lange* eine angegebene logische Bedingung erfüllt ist.

Inhalt der Datei `code04.gp`

```
i=2;
while(isprime(i),print("Hello World");i--)
```

Schleifen: `while`

- `isprime` testet, ob eine Zahl prim ist.
- Mit `i++` bzw. `i--` wird die in `i` gespeicherte Zahl um 1 vergrößert bzw. verringert.
- (Kann genauso gut schreiben: `i=i+1` bzw. `i=i-1`.)
- `while`-Schleifen werden durchlaufen, *so lange* eine angegebene logische Bedingung erfüllt ist.

Inhalt der Datei `code04.gp`

```
i=2;
while(isprime(i),print("Hello World");i--)
```

Ausgabe von GP-Pari nach Lesen von `code04.gp`

```
Hello World
```

if-Abfragen

- if-Abfragen sind von der Form `if([...],[...],[...])`.
- Vor dem ersten Komma steht eine logische Bedingung.
- Zwischen den Kommas steht eine Abfolge von Befehlen, die ausgeführt wird, *wenn* die logische Bedingung erfüllt ist.
- Nach dem zweiten Komma steht, was *ansonsten* ausgeführt wird.
- Für logische Bedingungen benutze: `==`, `!=`, `<`, `&&`, `||`,...

Inhalt der Datei `code05.gp`

```
if(!isprime(37),print("Hello World!"),print("Hi!"))
```

Ausgabe von GP-Pari nach Lesen von `code05.gp`

```
Hi!
```

Kombination for und if

Inhalt der Datei code06.gp

```
for(i=231,239,\
if(isprime(i),print(i" ist eine Primzahl"),\
print(i" ist keine Primzahl")))
```

Ausgabe von GP-Pari nach Lesen von code06.gp

```
231 ist keine Primzahl
232 ist keine Primzahl
233 ist eine Primzahl
234 ist keine Primzahl
235 ist keine Primzahl
236 ist keine Primzahl
237 ist keine Primzahl
238 ist keine Primzahl
239 ist eine Primzahl
```

Listenkonstruktion mit Mengenschreibweise

Inhalt der Datei code07.gp

```
a=[1..10];a
b=[fibonacci(x)|x<-[1..10]];b
c=[x|x<-[1..100], isprime(x)];c
print("Die 17.-kleinste Primzahl ist "c[17]".");
```

Ausgabe von GP-Pari nach Lesen von code07.gp

```
%9 = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
%10 = [1, 1, 2, 3, 5, 8, 13, 21, 34, 55]
%11 = [2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43,
    47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97]
Die 17.-kleinste Primzahl ist 59.
```

Permutationen: Typ Vecsmall

Inhalt der Datei code08.gp

```
p1=Vecsmall([1,3,2]);p2=Vecsmall([2,1,3]);
p1*p2
p1^2
p1^-1
permorder(p1)
permcycles(p1)
```

Ausgabe von GP-Pari nach Lesen von code08.gp

```
%14 = Vecsmall([3, 1, 2])
%15 = Vecsmall([1, 2, 3])
%16 = Vecsmall([1, 3, 2])
%17 = 2
%18 = [Vecsmall([1]), Vecsmall([2, 3])]
```

Reelle und komplexe Zahlen

- GP-Pari kann mit reellen und komplexen Zahlen rechnen.
- Bei reellen Zahlen kann man mit `\p` bestimmen, wie viele Nachkommastellen berechnet werden sollen.
- Eingebaut sind `I`, `Pi`, `exp`, `sin`, `cos`, `log`, `sqrt`,...

Inhalt der Datei `code09.gp`

```
I^2
Pi
sqrt(2)
\p 50;
Pi
exp(1)
log(1)
```

Reelle und kompexe Zahlen (2)

Ausgabe von GP-Pari nach Lesen von `code09.gp`

```
%1 = -1
%2 = 3.1415926535897932384626433832795028842
%3 = 1.4142135623730950488016887242096980786
   realprecision = 57 significant digits (50 digits
    displayed)
%4 = 3.14159265358979323846264338327950288419716939937751
%5 = 2.71828182845904523536028747135266249775724709370000
%6 = 0.E-57
```

$0.E-57$ bedeutet: Null Komma Null. . . (bis auf 57 Nachkommastellen)
Natürlich gilt bei exakter Rechnung: $\log(1) = 0$.

Vektoren/Listen

- Vektoren kann man auch wie durch eine Art `for`-Schleife definieren. (Siehe Beispiel.)
- Durch `#v` kann man auf die Länge des Vektors `v` zugreifen.
- Mit `v[i]` kann man den `i`-ten Eintrag von `v` abrufen.

Inhalt der Datei `code13.gp`

```
v=vector(13,i,nextprime(i))
#v
v[8]
```

Ausgabe von GP-Pari nach Lesen von `code13.gp`

```
%44 = [2, 2, 3, 5, 5, 7, 7, 11, 11, 11, 11, 13, 13]
%45 = 13
%46 = 11
```

Matrizen

- Man kann Matrizen eingeben, indem man die Einträge zwischen eckige Klammern `[...]` schreibt.
- Dabei beginnt `,` einen neuen Eintrag und `;` eine neue Zeile.
- z.B. ist `[1,2,3;1,2,3]` eine (2×3)-Matrix.
- Es gelten die gewohnten Rechenregeln.
- Mit nachgestelltem `~` wird transponiert.

Inhalt der Datei `code10.gp`

```
m=[1,2,3;1,2,3];v=[1;1;1];
m*v
v*m
m+[0,0,1;0,0,0]
```

Matrizen (2)

Ausgabe von GP-Pari nach Lesen von `code10.gp`

```
%7 =
[6]

[6]

  ***   at top-level: v*m
  ***                 ^--
  *** _*_: inconsistent operation 'RgM_mul' t_MAT (3x1) ,
    t_MAT (2x3).
%8 =
[1 2 4]

[1 2 3]
```

Abbildungen

- Abbildungen zwischen endlichen Mengen kann man mit `Map` definieren.
- Eingabe ist eine Matrix mit zwei Spalten.
- Links in jeder Zeile soll ein Element des Definitionsbereichs stehen, rechts das Bild des Elements.
- Das Bild eines Elements (falls definiert) bekommt man mit `mapget(NameDerAbb,ElementImDefinitionsbereich)`.

Abbildungen (2)

Inhalt der Datei code11.gp

```
chartonum=["A","B","C","D","E","F","G","H","I","J","K",\
"L","M","N","O","P","Q","R","S","T","U","V","W","X","Y",\
"Z";1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,\
22,23,24,25,26];
numbof=Map(chartonum~);
mapget(numbof,"N")
```

Ausgabe von GP-Pari nach Lesen von code11.gp

```
%3 = 14
```

Funktionen

- Kann eigene Funktionen definieren. Es gibt mehrere Möglichkeiten, eine Funktion zu schreiben. (siehe Benutzerhandbuch, Abschnitt 2.7)
- Angenommen, wir wollen die Prozedur `neuefunktion` schreiben, die drei Eingaben benötigt. Wir können schreiben: `neuefunktion(a,b,c)={...}`.
- In den geschwungenen Klammern steht eine Reihe von Befehlen, durch `;` getrennt, die die Eingaben `a,b,c` verarbeiten.
- Es ist oft der Fall, dass man mit Hilfe von `return(...)` ein bestimmtes Objekt als Ergebnis der Funktion ausgibt.
- Ein `return(...)` ist nicht nötig wenn man z.B. nur `print(...)`-Befehle in Abhängigkeit von `a,b,c` ausführen will.

Verschlüsselungsfunktion encryptcaesar

Angenommen, wir haben die Umkehrabbildung charof von numbof aus code11.gp auch definiert. Betrachte jetzt:

Inhalt der Datei code12.gp

```
shiftperm(n)={Vecsmall(concat(vector(n-1,i,i+1),[1]))};
caesarshift(key)=(shiftperm(26))^(key);

encryptcaesar(string,key)={
my(a,b,c,d);
a=strsplit(string);
b=[mapget(numbof,a[i])|i<-[1..#a]];
c=[(caesarshift(key))[b[i]]|i<-[1..#b]];
d=[mapget(charof,c[i])|i<-[1..#c]];
return(strjoin(d));
}
encryptcaesar("HELLOWORLD",3)
```

Verschlüsselungsfunktion `encryptcaesar` (2)

- `shiftperm(n)` gibt eine Permutation aus, die jede Zahl $i \in \{1, \ldots, n-1\}$ auf $i+1$ abbildet und n auf 1.
- `caesarshift(key)` ist eine durch `key` bestimmte Potenz dieser Permutation.
- `encryptcaesar(string,key)` spaltet den eingegebenen Text in Einzelbuchstaben, übersetzt die Buchstaben in Zahlen und permutiert sie. Danach wird in Buchstaben zurück übersetzt und alles wieder zum Ausgabetext zusammengefügt.

Verschlüsselungsfunktion `encryptcaesar` (2)

- `shiftperm(n)` gibt eine Permutation aus, die jede Zahl $i \in \{1, \ldots, n-1\}$ auf $i+1$ abbildet und n auf 1.
- `caesarshift(key)` ist eine durch `key` bestimmte Potenz dieser Permutation.
- `encryptcaesar(string,key)` spaltet den eingegebenen Text in Einzelbuchstaben, übersetzt die Buchstaben in Zahlen und permutiert sie. Danach wird in Buchstaben zurück übersetzt und alles wieder zum Ausgabetext zusammengefügt.

Ausgabe von GP-Pari nach Lesen von `code12.gp`

```
%39 = "KHOORZRUOG"
```

Häufigkeitsanalyse von Buchstaben

Die folgende Funktion `countcharacters` kann man bei der Kryptoanalyse von verschlüsselten Texten gut gebrauchen:

Inhalt der Datei `code15.gp`

```
countcharacters(string)={
my(a,b);
a=vector(26);
b=strsplit(string);
for(i=1,#b,a[mapget(numbof,b[i])]+=1);
return(a);
}
countcharacters("HELLOWORLD")
```

Ausgabe von GP-Pari nach Lesen von `code15.gp`

```
%59 = [0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 3, 0, 0, 2, 0, 0,
    1, 0, 0, 0, 0, 1, 0, 0, 0]
```

Rechnen mit Restklassen

- Kann `Mod(a,N)` schreiben, um Restklasse der ganzen Zahl a modulo der natürlichen Zahl $N > 1$ zu definieren.
- Mit `lift(...)` erhält man eine ganze Zahl zurück, die der kleinste nicht negative Vertreter der Restklasse ist.
- `chinese(...,...)` ist eingebaute Funktion zum simultanen Lösen von zwei Kongruenzen. (Nur wenn mathematisch möglich!)

Inhalt der Datei `code14.gp`

```
a=Mod(123456789,37)
lift(a)
chinese(Mod(3,5),Mod(1,4))
chinese(Mod(2,6),Mod(2,8))
chinese(Mod(3,6),Mod(2,8))
```

Rechnen mit Restklassen (2)

Ausgabe von GP-Pari nach Lesen von `code14.gp`

```
%47 = Mod(36, 37)
%48 = 36
%49 = Mod(13, 20)
%50 = Mod(2, 24)
  ***   at top-level: chinese(Mod(3,6),Mod(2,8))
  ***                 ^-------------------------
  *** chinese: inconsistent chinese t_INTMOD , t_INTMOD.
```

Auch bei Polynomen in einer Variable kann man mit Restklassen rechnen: z.B. `Mod(X^10,X^2+1)`.

Eingebaute Hilfe: Listen von Befehlen

- Mit `?n` für $n \in \{1, \ldots, 17\}$ kann man eine Liste von eingebauten Funktionen zu einem bestimmten Thema anzeigen lassen.
- Welches `n` zu welchem Thema gehört, kann man sehen, wenn man `?` eingibt.
- z.B. listet `?7` Funktionen zum Thema Matrizen und Lineare Algebra auf (man kann charakteristische Polynome, Kerne, Bilder,... berechnen lassen).
- Bei `?12` erhält man Infos zu Funktionen zum Thema Elliptische Kurven. Das könnte für uns im Verlauf des Semesters noch sehr nützlich werden.
- Für weitergehende Infos zu einer `funktion`: `?funktion` eingeben und/oder im Benutzerhandbuch suchen.