PARI/GP (autore: Vittorio Albertoni)

Premessa

Esiste una branca della matematica, chiamata teoria dei numeri, che riguarda una serie inimmaginabile di proprietà dei numeri interi e di proprietà dei sistemi algebrici che generalizzano gli interi.

Da Fermat, che è considerato l'iniziatore di questa teoria, a Gauss che ne è considerato il consolidatore e a tanti altri praticamente contemporanei di Gauss, da Galois a Lagrange e a Riemann, si è sviluppata la produzione di tutta una serie di teoremi e di dimostrazioni in materia di numeri, di algoritmi, la cui utilità concreta sfugge ad un comune mortale ma che fanno parte di un patrimonio di conoscenze da cui, inavvertitamente per quel comune mortale, sono derivate cose di cui lui stesso si serve.

Un banalissimo esempio quello della prova del nove con cui, già alla scuola elementare, ci hanno insegnato a controllare l'esattezza di una moltiplicazione tra numeri interi si basa sull'aritmetica modulare, che è una delle tante componenti della teoria dei numeri.

Un esempio meno banale quello della crittografia con l'algoritmo RSA a doppia chiave, pure basato sull'aritmetica modulare.

Per non parlare degli algoritmi risolutivi di equazioni polinomiali, del calcolo combinatorio, ecc. che conosciamo come tali senza sapere che si basano sulla teoria dei numeri.

PARI è una enorme libreria di funzioni scritte in linguaggio C che vanno alla radice della teoria dei numeri.

GP è un interprete che fornisce una interfaccia a linea di comando alle funzioni di PARI e che permette di fruire di un ambiente di sviluppo per applicazioni numeriche utilizzando un linguaggio di programmazione originale.

Il progenitore di questo software si chiamava Isabelle e fu scritto dai matematici Henri Cohen e François Dress presso l'Università di Bordeaux.

L'attuale software, PARI/GP, è stato sviluppato nel 1985 da una équipe guidata dallo stesso Henri Cohen presso il Laboratoire A2X ed è attualmente mantenuta da Karim Belabas, coadiuvato da moltissimi volontari, presso l'Università di Bordeaux.

Software libero in piena regola.

Il nome PARI è l'acronimo di Pascal ARIthmetic in quanto originariamente si era scelto il linguaggio Pascal per programmare le funzioni. Esso è rimasto anche se in seguito si è passati al linguaggio C, molto più veloce del Pascal.

Il nome GP, originariamente GPC che stava per Grand Programmable Calculateur, deriva da una semplificazione di GPC con la caduta della C.

La libreria PARI può essere utilizzata direttamente in programmi scritti con il linguaggio C, così come gli script in linguaggio GP possono essere convertiti in linguaggio C ed essere eseguiti ancor più velocemente di quanto sarebbe se li eseguissimo dalla shell GP.

Ma ritengo che queste cose esulino dall'interesse dei dilettanti cui dedico i miei manualetti e qui mi limiterò a descrivere come si possa utilizzare la libreria PARI con il linguaggio GP o in modo interattivo nella shell GP oppure attraverso script eseguibili nella stessa shell GP.

Indice

1 Installazione

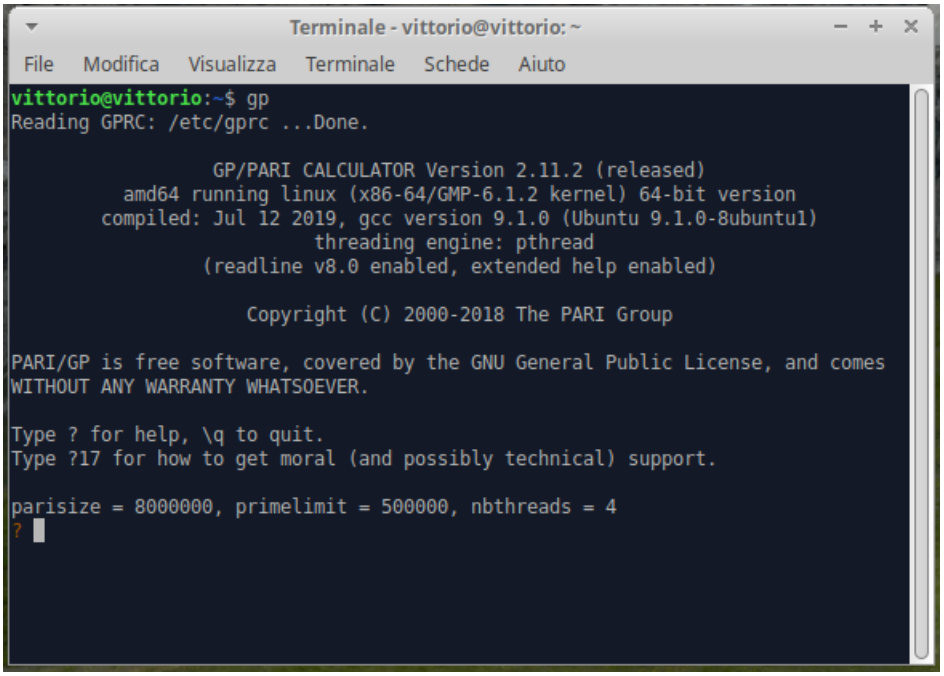
Troviamo PARI/GP all'indirizzo *https://pari.math.u-bordeaux.fr/*.

Nel momento in cui scrivo (Gennaio 2023) la versione stabile è la 2.15.1, rilasciata il 2 novembre 2022.
Dalla pagina DOWNLOAD possiamo scaricare gli installer per Windows e Mac.
Abbiamo anche un file `.apk` per Android.
A chi usa Linux, ancora da alcuni erroneamente considerato uno smanettone, viene offerto il source code in un file `.tar.gz`. Troviamo istruzioni per la sua compilazione nell'INSTALLATION MANUAL, nella pagina TUTORIALS. E' un po' un'operazione da mal di testa ma, fortunatamente, quasi sempre si trova PARI/GP nel repository della distro in uso e lo si può installare con il Gestore dei programmi.

2 Come funziona

Se lavoriamo su Linux o Mac apriamo la shell di PARI/GP con il comando `gp` a terminale. Se lavoriamo su Windows abbiamo a disposizione un lanciatore con icona su cui cliccare.

Apriamo così la shell di PARI/GP, anzi, di GP/PARI, in quanto si tratta di una finestra di GP sulla libreria PARI. Su un mio computer equipaggiato Linux appare così



e si autodefinisce GP/PARI CALCULATOR.

In effetti si tratta di una calcolatrice con la quale, oltre alle solite banali operazioni aritmetiche, accedendo alle funzioni della libreria PARI richiamandone il nome con adeguata sintassi, possiamo fare una enorme varietà di calcoli ricollegabili alla teoria dei numeri.

Se scriviamo in corrispondenza al prompt, contrassegnato dal simbolo `?`, `3*4` otteniamo il risultato `12` nella riga successiva.

Se scriviamo `fibonacci(25)` otteniamo il risultato `75025`, che è il numero che occupa il venticinquesimo posto della successione di Fibonacci.

Se scriviamo `factor(725)` otteniamo il risultato `2 volte 5 e 29`, che sono i fattori che compongono il numero `725`.

Se scriviamo `gcd(15,20)` otteniamo il risultato `5`, che è il massimo comune divisore di `15` e `20`.

Possiamo scrivere i comandi che vogliamo eseguire, uno via l'altro, nelle righe di un file di testo che salviamo con l'estensione `.gp`.

La loro esecuzione avviene leggendo il file di testo con il comando
`\r "<percorso_al_file.gp>"`
Con questo stesso comando vedremo che possiamo richiamare funzioni create da noi e memorizzate in file di testo od anche eseguire veri e propri script redatti con il linguaggio GP.

Per esempio, se scrivo i comandi visti prima

```
3*4
fibonacci(25)
factor(725)
gcd(15,20)
```

così come elencati, in un file che salvo come `comandi.gp`, in una directory Programmi della mia home, con
`\r "/home/vittorio/Programmi/comandi.gp"`
ottengo, uno via l'altro, i risultati.

I risultati che otteniamo sono preceduti dal simbolo `%` seguito da un numero progressivo: è questo un contrassegno che serve per richiamare il risultato stesso e renderlo protagonista di un'altra elaborazione.
Se, per esempio, volessimo triplicare il risultato contrassegnato da `%5` basterebbe scrivere `%5*3`.

Tutto questo per quanto riguarda la così detta calcolatrice.

Prima di arrivare al prompt per utilizzarla, nella finestra della shell abbiamo alcune utilissime indicazioni.
La prima riguarda l'help, che si apre semplicemente scrivendo un punto interrogativo (`?`) in corrispondenza del prompt. Su questo ritornerò nel seguito.
La seconda riguarda il modo di uscire dalla shell attraverso il comando `\q`. Lo si può fare anche con `Ctrl+D`.
La terza ci invita, se cerchiamo supporto morale o tecnico, a digitare `?17`. In questo modo otteniamo indicazione su come trovare la documentazione (soprattutto il manuale e il tutorial) di PARI/GP e altro.

Abbiamo, infine, le indicazioni `parisize`, `primelimit` e `nbthreads` con i loro valori di default (nel caso dell'illustrazione, rispettivamente, di `8000000`, `500000` e `4`).
La prima indicazione riguarda i byte di memoria allocata per il funzionamento del programma (nel caso 8 milioni di byte, cioè poco meno di 8 MB).
La seconda indicazione riguarda il valore fino al quale sono stati precalcolati i numeri primi, precalcolo finalizzato a velocizzare l'esecuzione di certe funzioni.
La terza riguarda il numero di threads attivabili nel corso di elaborazioni parallele.

Le prime due assegnazioni di default consentono l'efficiente comportamento del programma in situazioni di normalità.
In caso di necessità è possibile modificare queste assegnazioni: la prima con il comando nella shell `gp -s`, la seconda con il comando `gp -p`, seguiti dal numero che si intende sostituire a quello di default.

La terza assegnazione è fatta in relazione alle potenzialità dell'hardware su cui stiamo operando.

3 Tipi di dato

I tipi di dato che possiamo elaborare con PARI sono elencati nel paragrafo 1.3 del manuale.

Qui rammento quelli più alla portata di un dilettante.

3.1 Tipi base

Numerici

I tipi numerici sono l'intero (`t_INT`), il reale (`t_REAL`), il razionale (`t_FRAC`) e il complesso (`t_COMPLEX`).

L'intero si scrive in quanto tale: `6` è un numero intero.

Il reale si scrive con la parte decimale separata da un punto: `1.75` è un numero reale.

Il razionale è un rapporto tra numeri interi che non ha per risultato un intero: `5/2` è un numero razionale.

Il complesso è formato da una parte reale e da una parte immaginaria e si scrive così `x+I*y`, dove `x` è la parte reale e `y` è la parte immaginaria.

Tutti i tipi numerici sono trattati in precisione arbitraria e non c'è praticamente limite al numero delle cifre significative gestite: il limite è posto solo dall'hardware.

Nella shell i numeri interi si usano scrivere fino ad oltre 3000 cifre e i numeri reali si usano scrivere in notazione scientifica evidenziando 39 cifre, virgola compresa. Queste limitazioni agiscono unicamente in sede di stampa del numero e nulla hanno a che vedere con la precisione interna, che va oltre queste limitazioni.

Stringhe

In un software che tratta praticamente solo operazioni numeriche la stringa è presente solo giusto per avere a disposizione uno strumento che serva per scrivere qualche parola qua e là.

La stringa (`t_STR`) è una successione di caratteri racchiusa tra doppi apici.

"Ciao" è una stringa.

3.2 Tipi contenitore

Sono formati da raggruppamenti organizzati di tipi base.

Essi sono il vettore riga (`t_VEC`), il vettore colonna (`t_COL`), la lista (`t_LIST`) e la matrice (`t_MAT`).

Il vettore riga si crea elencando tra parentesi quadre i componenti separati da virgola. `[1,2,3,4]` crea un vettore riga formato dai primi quattro numeri interi.

Il vettore colonna si crea nello stesso modo, aggiungendo una tilde alla fine. `[1,2,3,4]~` crea un vettore colonna formato dai primi quattro numero primi.

La lista si crea con la parola chiave `List` seguita da un vettore indicato tra parentesi tonde. `List([1,2,3,4])` crea una lista formata dai primi quattro numeri interi.

La matrice si crea indicando, tra parentesi quadre, gli elementi di ciascuna riga, separati da virgola, separando le righe con un punto e virgola. `[1,2;3,4]` crea la matrice quadrata formata dai primi quattro numeri interi.

Gli elementi contenuti sono indicizzati a partire da 1 e sono raggiungibili inserendo l'indice tra parentesi quadre. `[1,6,4,7][2]` ritorna `6`.

Per aggiungere un elemento a un vettore dobbiamo trasformarlo in lista con `List()`, aggiungere l'elemento alla lista con `listput(<lista>, <elemento>)` e ritrasformare la lista in vettore con `Vec(<lista>)`.

Esempio:

Per aggiungere il numero `18` al vettore `v = [7,8,12,15]` agiamo così:

```
l = List(v)
listput(l,18)
v = Vec(l)
```

e il nuovo `v` sarà `[7, 8, 12, 15, 18]`.

E' così chiarita l'utilità della lista, che, a prima vista, sembrerebbe un inutile doppione del vettore.

4 Operatori

Gli operatori sono simboli che, posti tra due o più operandi, formano una espressione che fornisce un risultato.

4.1 Operatori aritmetici

Con i tipi numerici possiamo utilizzare i seguenti operatori, elencati in ordine di priorità di esecuzione, per compiere le più comuni operazioni aritmetiche:
^ per l'elevamento a potenza,
* per la moltiplicazione,
/ per la divisione,
\ per la divisione euclidea (divisione con resto),
% per il resto della divisione euclidea,
+ per la somma,
- per la sottrazione.

L'operatore / tra due interi, se il risultato non è un intero, produce una frazione.
`4/2` fornisce il risultato `2`; `5/2` fornisce il risultato `5/2`.

Per ottenere un risultato espresso come numero reale, almeno uno degli operandi deve essere un numero reale.
`5./2` fornisce il risultato `2.500000000000000000000000000000000000000`.

L'operatore * tra matrici o vettori produce il prodotto matriciale.

4.2 Operatori di confronto

Sono i seguenti
== uguale,
!= diverso,
< minore,
<= minore o uguale,
> maggiore,
>= maggiore o uguale.

Fare molta attenzione ad utilizzare il giusto simbolo per verificare l'uguaglianza, costituito da doppio uguale (==). Se utilizziamo un semplice uguale (operatore di assegnazione) combiniamo disastri in quanto, anche in un chiaro contesto di confronto, GP attuerebbe uno scambio di valori non voluto.

4.3 Operatori logici

Sono
! NOT
&& AND
|| OR

5 Funzioni

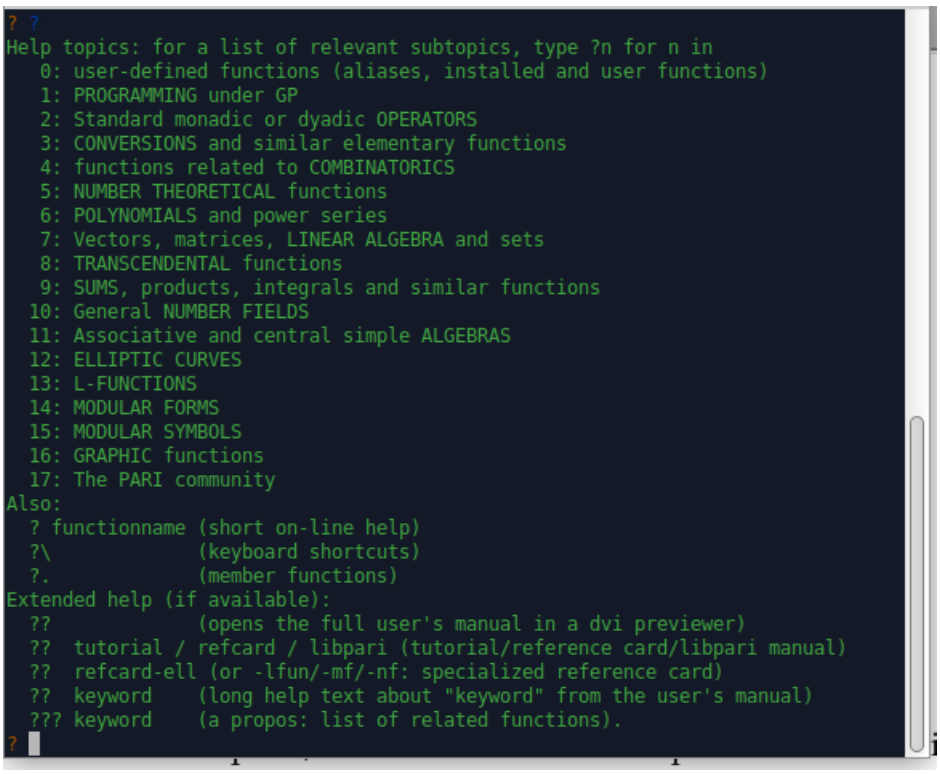
La ricchezza di PARI è costituita dalla vastissima quantità di funzioni preconfezionate che consentono di eseguire anche calcoli complicatissimi con il minimo sforzo, richiamandole con il linguaggio GP.

A queste funzioni è dedicato il Capitolo 3 del manuale, che le suddivide in gruppi, descrivendone le finalità ed indicando la sintassi con cui utilizzarle, sintassi che si riduce praticamente a quali siano e come si indichino i parametri richiesti affinché siano prodotti i risultati voluti.

Possiamo avere le stesse indicazioni ricorrendo all'help dalla shell di PARI/GP, help che si apre digitando un punto di domanda (?) in corrispondenza del prompt.

La figura all'inizio della pagina seguente mostra l'indice che compare all'apertura dell'help.

Le voci dell'indice sono contrassegnate da un numero e i numeri da 2 a 16 contrassegnano i gruppi di funzioni.



Digitando al prompt della shell uno di questi numeri preceduto dal punto di domanda otteniamo l'elenco delle funzioni appartenenti al gruppo contrassegnato da quel numero.

Finalmente, digitando il nome di una funzione preceduto dal punto di domanda otteniamo la descrizione della funzione e la sua sintassi.

Se sappiamo già il nome della funzione di cui vogliamo vedere la sintassi, basta che scriviamo questo nome preceduto dal punto di domanda.

Se in corrispondenza del prompt scriviamo

```
?factorial
```

otteniamo

```
factorial(x):  factorial of x, the result being given as a real number.
```

e veniamo così a sapere che passando alla funzione `factorial()` un argomento x otteniamo il fattoriale di x scritto come numero reale.

```
factorial(45)
```

ritorna

```
1.196222208654801945619631614956577 E56
```

Se preferiamo vedere il risultato come numero intero possiamo scrivere al prompt

```
45!
```

e avremo di ritorno

```
119622220865480194561963161495657715064383733760000000000
```

Come ho già avuto modo di ricordare nel Capitolo 3, parlando dei tipi numerici, in entrambi i casi abbiamo numeri a precisione arbitraria: le 38 cifre evidenziate nel numero reale in notazione scientifica sono solo una parte delle cifre significative, non come avviene in altri linguaggi, dove quelle evidenziate sono solo le cifre significative.

Sia il manuale sia il sistema di help sono in lingua inglese, ma facile facile; importante è conoscere la matematica.

6 Funzioni definite dall'utente

Le funzioni preconfezionate sono moltissime, ma non è detto che ci siano tutte quelle che ci servono.

Se, per esempio, scorriamo le funzioni contenute nel raggruppamento

`4: functions related to COMBINATORICS`
non troviamo nulla di direttamente dedicato al calcolo di combinazioni, disposizioni e permutazioni.

Per la verità abbiamo la funzione `binomial()` alla quale possiamo passare i parametri `x` e `k` per trovare il valore del coefficiente $\binom{x}{k}$ cui corrisponde il numero delle combinazioni senza ripetizione di x oggetti presi k per volta.
Con `binomial(20,4)` otteniamo `4845`, che è il numero di combinazioni senza ripetizione possibili raggruppando 20 oggetti a 4 per volta.

Sappiamo anche che il numero delle permutazioni di n oggetti corrisponde al fattoriale del numero n.

Se vogliamo avere a disposizione una funzione per calcolare le disposizioni possiamo costruircela noi partendo dalla formula

$$D(n,k) = \frac{n!}{(n-k)!}$$

che calcola il numero delle disposizioni senza ripetizione possibili raggruppando n oggetti k per volta.

La sintassi per costruire la funzione è la seguente

```
disposizioni(n, k) = {
   print(n!/(n-k)!);
}
```

Salviamo la funzione in un file chiamato `disposizioni.gp`.

Per utilizzare la funzione dobbiamo caricarla con il comando

```
\r "/.../.../disposizioni.gp"
```

dove al posto dei puntini mettiamo il percorso a dove è salvato il file.

Dopo avere caricato la funzione, per esempio con il comando

```
disposizioni(20, 4)
```

otteniamo il risultato `116280` che è il numero delle disposizioni possibili raggruppando 20 oggetti a 4 per volta.

7 Programmare con GP

GP, oltre che essere un linguaggio adatto ad utilizzare le funzioni di PARI per ottenere risultati come se utilizzassimo una calcolatrice, è un vero e proprio linguaggio di programmazione: per essere più esatti, di scripting, in quanto i suoi programmi non sono compilati ma vengono interpretati nella shell di GP per essere eseguiti.

Uno script GP ha bisogno di essere redatto secondo una impostazione ben precisa: se non si rispetta questa impostazione gli script non funzionano o forniscono risultati strampalati.

A questo fine ricordiamo che:
. uno script GP inizia con una parentesi graffa aperta (`{`) e termina con una parentesi graffa chiusa (`}`);
. ogni riga di istruzione termina con il punto e virgola (`;`),
. prima della parentesi graffa di apertura non ci deve essere nessuna riga vuota e dopo la parentesi graffa di chiusura può esserci una sola riga vuota, corrispondente alla pressione del tasto INVIO dopo aver scritto questa parentesi graffa di chiusura.

Lo script si salva in un file con estensione .gp e lo si lancia nella shell con il comando `\r "<percorso_al_file.gp>"`.

Questo piccolo script che ci chiede il nome per salutarci

```
{
print("Come ti chiami?");
nome=input();
print("Ciao, ",nome);
}
```

7.1 Variabili

In informatica una variabile è una locazione di memoria in cui teniamo a disposizione un valore, un oggetto, per utilizzarlo quando serve nel corso di un programma. Si chiama variabile in quanto possiamo modificarne il contenuto.

L'attribuzione del valore iniziale o la sua modifica si fa con l'operatore di assegnazione = con la seguente sintassi:
<nome_variabile> = <valore>.

Nel piccolo esempio di prima l'istruzione
`nome=input();`
crea la variabile nome e le assegna il valore letto dallo standard input (tastiera).

Con l'istruzione
`a = 7.5`
si crea la variabile `a` assegnandole il valore `7.5` e la variabile assume il tipo del dato assegnato, in questo caso `t_REAL`.

Con l'istruzione
`a = 8`
modifichiamo il valore della variabile `a` assegnandole il valore `8` e il tipo della variabile diventa `t_INT`.

Se, al di fuori di uno script, scriviamo le istruzioni come fatto sopra, oltre all'assegnazione dei valori, questi vengono anche scritti. Se chiudiamo le istruzioni con un punto e virgola (;) i valori vengono assegnati alle variabili senza essere scritti.

All'interno di uno script, come ho già detto, è d'obbligo il punto e virgola alla fine di ogni istruzione.

7.2 Interattività con l'utente

In uno script che si rispetti, prima o poi è necessario interfacciarsi con l'utente, vuoi per acquisire dati vuoi per scrivere risultati.

L'acquisizione di dati digitati dallo standard input (tastiera) avviene attraverso la funzione
`input()`
che legge una stringa e la interpreta come espressione riconosciuta da GP.

La scrittura sullo standard output (schermo) avviene attraverso la funzione
`print(`<cosa_scrivere>`)`
che scrive in modo grezzo ciò che è indicato come argomento e va a capo.
L'argomento <cosa_scrivere> può essere una stringa, il valore di una variabile, il risultato di una espressione.
Se non si indica l'argomento viene prodotto un a capo oppure, se già si è a capo, viene prodotta una riga vuota.

Se vogliamo scrivere in modo formattato abbiamo a disposizione la funzione
`printf("direttiva_formattazione>", `<cosa_scrivere>`)`
ereditata tale e quale dal linguaggio C.
La stringa della direttiva di formattazione comincia con il carattere %, prosegue con un intero indicante l'ampiezza in caratteri del campo di allineamento a destra, dopo un punto di separazione (`.`) prosegue con un intero indicante l'eventuale numero di cifre decimali e si chiude con il carattere `s` se si tratta di scrivere una stringa, con il carattere `d` se si tratta di scrivere un numero intero e con il carattere `f` se si tratta di scrivere un numero decimale.

Questo piccolo script esemplifica l'uso di `printf()`:

```
{
a = 17.74636748;
b = 2.54637628;
printf("a vale:  " "%15.4f",a); print();
printf("b vale:  " "%15.4f",b);
}
```

Eseguendolo otteniamo a terminale quanto segue:

```
a vale:         17.7464
b vale:          2.5464
```

che è un po' meglio di ciò che otterremmo utilizzando la funzione `print()` con quest'altro script:

```
{
a = 17.74636748;
b = 2.54637628;
print("a vale:  ", a);
print("b vale:  ", b);
}
```

eseguendo il quale avremmo a terminale quanto segue:

```
a vale: 17.746367480000000000000000000000000000
b vale: 2.546376280000000000000000000000000000
```

Per l'uso combinato delle funzioni `input()` e `print()` valga il piccolo esempio dello script del saluto visto prima.

7.3 Strutture di controllo

Le strutture di controllo sono delle particolari funzioni attraverso le quali possiamo condizionare l'esecuzione di certe istruzioni al verificarsi di determinate condizioni oppure possiamo ottenere la ripetizione di una o più istruzioni.

Nel primo caso parliamo di esecuzione condizionale e nel secondo parliamo di ripetizione.

7.3.1 Esecuzione condizionale

Otteniamo l'esecuzione condizionale di certe istruzioni utilizzando la funzione `if` con la seguente sintassi:

`if(<condizione>,<istruzioni_per_vero>,<istruzioni_per_falso>)`

Questo piccolo script ci chiede di indicare un numero intero. Se quello indicato è un numero primo ci viene confermata tale circostanza, altrimenti ci viene detto che non si tratta di un numero primo e ce ne viene indicata la scomposizione in fattori.

```
{
print("Scrivi un numero intero:  ");
n = input();
if(isprime(n), print("PRIMO"), print("NON PRIMO. Fattori:  " factor(n)));
}
```

Se eseguiamo lo script e indichiamo il numero 37 otteniamo:

```
Scrivi un numero intero:
37
PRIMO
```

Se eseguiamo lo script e indichiamo il numero 18 otteniamo:

```
Scrivi un numero intero:
18
NON PRIMO. Fattori: [2, 1; 3, 2]
```

Non necessariamente devono essere indicate entrambe le istruzioni.

Può infatti bastare l'indicazione di cosa fare se la condizione è vera con la sintassi

`if(<condizione>,<istruzioni_per_vero>)`

oppure di cosa fare se la condizione è falsa con la sintassi

`if(<condizione>, ,<istruzioni_per_falso>)`

Possiamo anche costruire la struttura in altri linguaggi denominata switch o case con la sintassi

```
if (<condizione>, <istruzioni>,
  <condizione>, <istruzioni>,
  <condizione>, <istruzioni>,
  ...., ....
  <istruzioni_di_default>);
```

7.3.2 Ripetizione

Le strutture di ripetizione nel linguaggio GP sono moltissime e rimando al manuale per chi voglia approfondire.

Qui ricordo quelle più semplici.

Abbiamo la funzione `for()` per ripetere l'esecuzione di una istruzione un numero di volte predefinito, con la seguente sintassi

`for(<variabile>=<inizio>,<fine>,<istruzioni>)`

Con l'istruzione

```
for(x = 1, 5, print("Ciao"))
```

scriviamo cinque volte la parola Ciao.

Abbiamo la funzione `while()` per ripetere l'esecuzione di una istruzione fino a quando si verifica una certa condizione, con la seguente sintassi

`while(<condizione>,<istruzioni>)`

Se la condizione è espressa attraverso l'uso di un contatore otteniamo gli stessi risultati che otteniamo con la funzione for() vista prima.

Con le istruzioni

```
i = 1
while(i<=5, print("Ciao"); i=i+1)
```

scriviamo cinque volte la parola Ciao.

* * *

Quanto visto in questo Capitolo descrive le possibilità che abbiamo di costruire script per utilizzare il linguaggio GP.

Non facciamoci tuttavia prendere da troppo facili entusiasmi.

Con questo linguaggio, infatti, non possiamo andare oltre un certo livello di complicazione.

Il limite deriva dal fatto che all'interno delle parentesi graffe di apertura e di chiusura dello script non è possibile gestire altre parentesi graffe, cioè non è possibile annidare cicli di operazioni nello script.

8 Espressioni e calcolo simbolico

In PARI/GP una sequenza di valori e di operatori forma un'espressione.

I valori possono essere indicati direttamente o attraverso il nome di una variabile che li contiene e l'espressione viene immediatamente valutata appena scritta.

Se scriviamo `3*4` e diamo INVIO, vediamo immediatamente il risultato della valutazione `12`.

Se abbiamo la variabile `v` che contiene il valore `3.5`, scriviamo `v+2` e diamo INVIO vediamo il risultato della valutazione `5.5`.

Se al posto dei valori o di alcuni valori scriviamo simboli otteniamo come risultato una nuova espressione in cui, per quanto possibile, compaiono risultati di operazioni aritmetiche e rimangono i simboli di ciò che non è stato possibile valutare.

Se scriviamo `3+2*x-1` e diamo INVIO otteniamo come risultato `2*x+2`.

Se scriviamo `(a+b)^2` otteniamo il risultato `a^2+2*b*a+b^2`.

Se scriviamo `(a+b)*(a-b)` otteniamo il risultato `a^2-b^2`.

La stessa cosa succede se abbiamo a che fare con tipi contenitori.

Dato il vettore orizzontale `v1 = [2, 3]` e il vettore verticale `v2 = [5, 2]~` il prodotto `v1*v2` fornisce il risultato `16`.
Dato il vettore orizzontale `v1 = [x1, x2]` e il vettore verticale `v2 = [y1, y2]~` il prodotto `v1*v2` fornisce il risultato `y1*x1 + y2*x2`.
Date le matrici `m1` = $\begin{vmatrix} 2 & 3 \\ 6 & 1 \end{vmatrix}$ e `m2` = $\begin{vmatrix} 4 & 2 \\ 5 & 3 \end{vmatrix}$ il prodotto `m1*m2` fornisce il risultato $\begin{vmatrix} 23 & 13 \\ 29 & 15 \end{vmatrix}$.
Date le matrici `m1` = $\begin{vmatrix} x_{11} & x_{12} \\ x_{21} & x_{22} \end{vmatrix}$ e `m2` = $\begin{vmatrix} y_{11} & y_{12} \\ y_{21} & y_{22} \end{vmatrix}$ il prodotto `m1*m2` fornisce il risultato
$\begin{vmatrix} y_{11} * x_{11} + y_{21} * x_{12} & y_{12} * x_{11} + y_{22} * x_{12} \\ y_{11} * x_{21} + y_{21} * x_{22} & y_{12} * x_{21} + y_{22} * x_{22} \end{vmatrix}$
Non ci ricordassimo come si calcola il prodotto matriciale potremmo ripassare così.

Addirittura possiamo ottenere l'espressione simbolica applicando certe funzioni.
Abbiamo, per esempio la funzione `matdet()` che calcola il determinante di una matrice.
Data la matrice `M` = $\begin{vmatrix} 5 & 3 \\ 2 & 4 \end{vmatrix}$, `matdet(M)` fornisce il risultato `14`.
Data la matrice in simboli M = $\begin{vmatrix} x_{11} & x_{12} \\ x_{21} & x_{22} \end{vmatrix}$ con `matdet(M)` otteniamo $x_{22} * x_{11} - x_{21} * x_{12}$.
Non ci ricordassimo come si calcola il determinante potremmo ripassare così.

Abbiamo una funzione per calcolare il valore numerico della derivata di una funzione in un certo punto. E' la funzione `derivnum()` con la sintassi
`derivnum(x =` <valore>, <espressione>`)`.
`derivnum(x = 0.5, x^2)` fornisce il risultato `1`, che è il valore numerico della derivata prima della funzione $y = x^2$ in corrispondenza all'ascissa 0.5.

Ma abbiamo anche una funzione per calcolare la derivata simbolica di una funzione. E' la funzione `deriv()` con la sintassi
`deriv(`<espressione>`)`.
`deriv(x^2)`
fornisce il risultato `2*x`, che è l'espressione analitica della derivata prima della funzione $y = x^2$.
Così come `deriv(1/x)` fornisce il risultato `-1/x^2`, ecc.