a cura
di Francesco Romani

TRUCCHI DEL MESTIERE, II

In questa puntata cerco di rispondere pubblicamente ad alcune delle domande più frequenti che ricevo per e-mail. Inoltre presento un nuovo strabiliante algoritmo per π dovuto a David Bailey, Peter Borwein e Simon Plouffe, trovato in rete grazie alla segnalazione di un lettore di MCmicrocomputer. Ai lettori a cui non piace la matematica prometto che il prossimo articolo sarà completamente dedicato ai calcoli non numerici

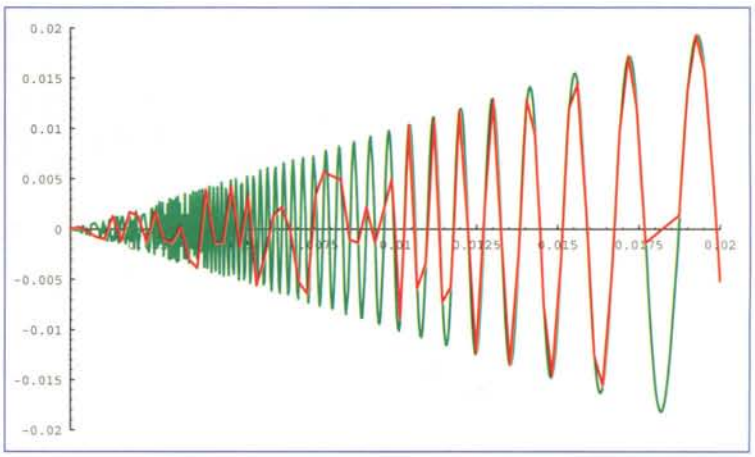


Figura 1

1. I grafici che vengono male

A volte capita che un grafico risulti completamente sballato. *Mathematica* contiene un algoritmo di plottaggio adattivo che cerca di intuire il comportamento della funzione per renderne il grafico con meno tratti possibile. Se la funzione è molto complicata con pendenze elevate e frequenti oscillazioni può capitare che pezzi di curva vengano saltati.

Esistono molte opzioni per controllare l'algoritmo di *plotting*, il più semplice consiste nel forzare il numero minimo di punti ad un valore elevato. Nell'esempio che segue una funzione altamente oscillante viene disegnata con il comando **Plot** rispettivamente con e senza l'opzione **PlotPoints ->200**.

In[1]:=

```
f[x_]:=x Sin[1/x]
pl1=Plot[f[x],{x,-0.1,0.1},
  PlotStyle->Red];
pl2=Plot[f[x],{x,-0.1,0.1},
  PlotPoints->200,PlotStyle->Green];
```

Il grafico viene ingrandito in vicinanza del punto critico x=0 e si nota la differenza tra i due grafici risultanti **(Figura 1)**.

In[4]:=

```
Show[pl2,pl1,
  PlotRange->{{0,0.02},{-0.02,0.02}}];
```

2. Regole di sostituzione

Con *Mathematica* è facile risolvere equazioni; meno banale è utilizzare i risultati ottenuti. Prendiamo una semplice equazione parametrica di II grado.

In[1]:=

```
regole=Solve[x^2-(c^2-2)x+1==0,x]
```

Out[1]=

$$\{\{x \to \frac{-2 + c^2 - \text{Sqrt}[-4 + (2 - c^2)^2]}{2}\},\ \{x \to \frac{-2 + c^2 + \text{Sqrt}[-4 + (2 - c^2)^2]}{2}\}\}$$

Sostituendo la **x** secondo le **regole** si ottengono le due soluzioni.

In[2]:=

```
soluzioni=x/.regole
```

Out[2]=

$$\{\frac{-2 + c^2 - \text{Sqrt}[-4 + (2 - c^2)^2]}{2},$$

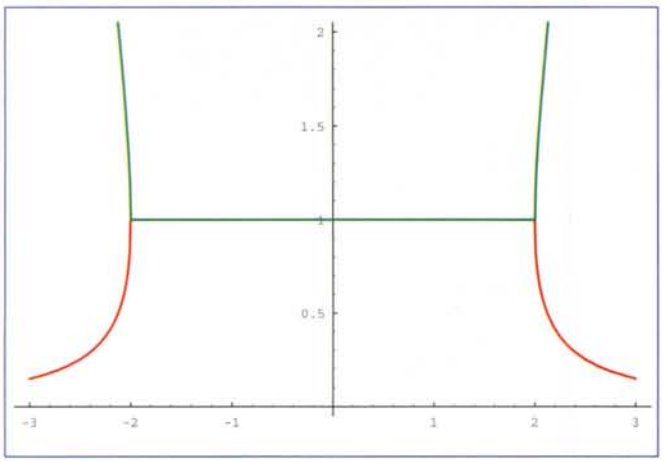


Figura 2

```
-2 + c^2 + Sqrt[-4 + (2 - c^2)^2]
-----------------------------------}
                 2
```

Le soluzioni possono quindi essere disegnate in funzione di **c** (poiché possono assumere valori complessi ne disegniamo il valore assoluto).

In[3]:=
```
Plot[Evaluate[Abs[soluzioni]],{c,-3,3},
        PlotStyle->{Red,Green}];
```

(Figura 2)

Per chiarirsi le idee su cosa accade nel piano complesso è possibile fare un grafico parametrico della parte reale e della parte immaginaria delle soluzioni (per c che varia fra -2 e 2, valori a cui corrispondono radici complesse) e combinare i risultati con colori diversi.

In[4]:=
```
{x1,x2}=soluzioni;
```

In[5]:=
```
ParametricPlot[
 {Re[x1],Im[x1]},{c,-2,2},
   AspectRatio->1,
   PlotStyle->Red,
   PlotRange->{{-1.5,1.5},{-1.5,1.5}}];
ParametricPlot[
 {Re[x2],Im[x2]},{c,-2,2},
   AspectRatio->1,
   PlotStyle->Green,
   PlotRange->{{-1.5,1.5},{-1.5,1.5}}];
Show[%,%%];
```

Si scopre che il modulo delle soluzioni è 1 e che nell'intervallo studiato ciascuna percorre una metà della circonferenza unitaria **(Figura 3)**.

3. Alcune sottigliezze sul calcolo di una somma

Nel numero 159 di MCmicrocomputer abbiamo visto alcuni algoritmi per sommare 1 a tutti gli elementi di un vettore; stavolta tocca al calcolo della somma di n valori. Parte del testo è ispirato al capitolo 2 dell'articolo elettronico di Adamchik e Wagon citato in bibliografia. Consideriamo 4 metodi diversi e distinguiamo 4 casi: somma di un vettore numerico già presente in memoria, somma di un vettore simbolico già presente in memoria, somma di n elementi numerici da calcolare, somma di n elementi simbolici da valutare.

La differenza tra il caso simbolico e quello numerico è sostanziale: una somma parziale tra 1000 valori numerici è **UN** numero, una somma parziale tra 1000 valori simbolici è (a meno di possibili semplificazioni) una complicata espressione.

Somma degli elementi di un vettore

Consideriamo di dover calcolare la somma degli elementi di un vettore **lista**; i metodi che prendiamo in considerazione sono:

1) l'uso del **For**

In[1]:=
```
metodo1:=
  (somma=0;
   For[i=1,i<=Length[lista],i++,
     somma=somma+lista[[i]]];
   somma)
```

2) l'uso del **Do**

In[2]:=
```
metodo2:=
 (somma=0;
  Do[somma=somma+
    lista[[i]],{i,Length[lista]}];
  somma)
```

3) l'uso di **Sum**

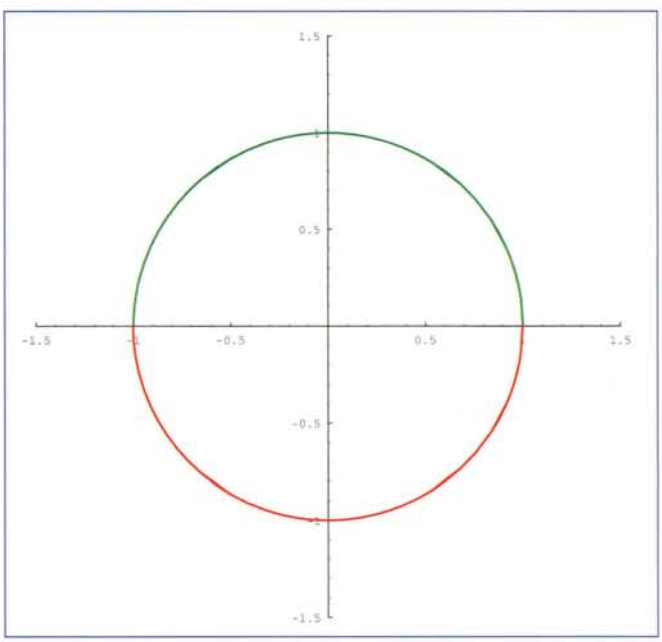


Figura 3

In[3]:=
```
metodo3:=
  Sum[lista[[i]],{i,Length[lista]}]
```

4) l'uso di **Apply[Plus,...]** abbreviato in **Plus@@...**

In[4]:=
```
metodo4:=Plus@@lista
```

Creiamo un vettore di 100000 elementi reali

In[5]:=
```
lista=Table[Random[], {100000}];
```

verifichiamo che i 4 metodi danno gli stessi risultati e vediamo i tempi di esecuzione.

In[6]:=
```
metodo1==metodo2==metodo3==metodo4
```

Out[6]=

True

In[7]:=
```
{Timing[metodo1][[1,1]],
 Timing[metodo2][[1,1]],
 Timing[metodo3][[1,1]],
 Timing[metodo4][[1,1]]}
```

Out[7]=
```
{35.4, 22.1333, 14.8667, 6.6}
```

La stessa prova per una lista di 10000 elementi simbolici (**f[]** non deve essere definita)

In[8]:=
```
lista=Table[f[i], {i,2000}];
```

ha dato i risultati

Out[10]=

Out[8]=
```
{28.6333, 28.15, 0.216667,  0.0333333}
```

Poiché **Plus** riordina i propri argomenti, calcolare la somma in un botto solo è infinitamente più veloce. Inoltre il metodo 4 si limita a sostituire **List** con **Plus** e in assenza di calcoli o di semplificazioni è praticamente istantaneo.

Esercizio: Il lettore provi a vedere cosa succede con
```
lista=Table[Sign[i]f[Abs[i]],
   {i,-1000,1000}];
```
(il risultato deve essere 0).

Somma di elementi calcolati

Supponiamo di dover calcolare la somma di espressioni. Prendiamo in considerazione i metodi 2, 3 e 4 ed iniziamo con il caso numerico sommando l'espressione **Random[]**.

In[11]:=
```
n=100000;
```

In[12]:=
```
Timing[
  somma=0;
  Do[somma=somma+Random[],{i,n}];
  somma]
```

Out[12]=
```
{17.6167 Second, 50020.9}
```

In[13]:=
```
Timing[Sum[Random[],{i,n}]]
```

Out[13]=
```
{12.7667 Second, 49999.3}
```

In[14]:=
```
Timing[Plus@@
  Table[Random[], {i,n}]]
```

Out[14]=
```
{12.8833 Second, 49920.3}
```

Anche in questo caso **Sum** e **Plus@@...** sono più rapidi ma richiedono la memorizzazione del vettore (**Sum** dapprima calcola tutti gli elementi da sommare, poi ne valuta la somma e infine libera la memoria utilizzata) e sono inapplicabili per sommare milioni di elementi. La situazione si ribalta totalmente nel caso di espressioni simboliche. Se non ci sono semplificazioni la memoria serve comunque e la maggiore velocità dei metodi funzionali è strabiliante.

Conclusioni

Non usate mai il **For**!

Se dovete sommare un vettore preesistente usate **Plus@@...**

Se dovete sommare molti valori numerici calcolati usate il **Do**.

Se dovete sommare espressioni simboliche usate **Sum** (che per inciso possiede potenti regole di semplificazione se si carica il pacchetto, **"Algebra`SymbolicSum`"**).

4. Ancora Pigreco!!!

Il calcolo della espansione decimale di π è stato diffusamente trattato su queste colonne. Pochi giorni fa Riccardo Bernardini, un lettore di Losanna, mi ha segnalato alcuni articoli che descrivono un nuovo approccio al calcolo delle espansioni delle costanti matematiche (espresse in una qualche base) che ha confutato una legge non scritta che era stata osservata per secoli.

Quello che segue è stato adattato dall'articolo di Adamchik e Wagon a cui si rimandano i lettori più esperti.

Si è sempre creduto che l'unico modo per calcolare una parte della espansione decimale di π fosse di calcolarne tutte le cifre fino a quel punto. Recentemente David Bailey, Peter Borwein, e Simon Plouffe (B.B.P. nel seguito) hanno trovato la seguente formula:

$$\pi = \sum_{k=0}^{\infty} \frac{1}{16^k}\left(\frac{4}{8k+1} - \frac{2}{8k+4} - \frac{1}{8k+5} - \frac{1}{8k+6}\right)$$

che con i trucchi che spiegheremo nel seguito permette di calcolare un gruppo di cifre **esadecimali** di π (per esempio 10 cifre a partire dalla miliardesima) senza calcolare le cifre precedenti e lavorando con una precisione limitata. La formula è stata trovata con un programma di ricerca euristica basato sull'algoritmo PSLQ (implementato in *Mathematica* da Richard Crandall).

Dimostrazione della formula per π

La dimostrazione della formula B.B.P. si può fare banalmente con *Mathematica*. Verifichiamo dapprima, attraverso una integrazione simbolica ed una semplificazione, l'identità:

$$(*)\qquad \int_0^{1/\sqrt{2}} \left(\sqrt{2}\right)^i z^{8k+i-1}\,dx = \frac{1}{16^k(8k+i)}$$

In[1]:=
```
2^(i/2) Integrate[z^(8 k+i-1), z]
```

Out[1]=
```
   i/2  i + 8 k
  2    z
  ------------
     i + 8 k
```

In[2]:=
```
%/.z->1/Sqrt[2]//Simplify
```

Out[2]=
```
        1
  --------------
   4 k
  2    (i + 8 k)
```

Definiamo ora le tre funzioni ad argomenti interi:

In[3]:=
```
f[i_] := Sum[1/(16^k (8 k + i)),
 {k, 0, Infinity}]
```

In[4]:=
```
g[i_] := 2^(i/2) Integrate[
  Sum[z^(8 k + i - 1),
    {k,0,Infinity}],{z,0,1/Sqrt[2]}]
```

In[5]:=
```
h[i_] := 2^(i/2) Sum[
  Integrate[z^(8 k + i - 1),
    {z,0,1/Sqrt[2]}],{k,0,Infinity}]
```

Si vede che **g[i]** e **h[i]** hanno lo stesso valore perché ottenute scambiando la sommatoria con l'integrale. Inoltre, grazie all'identità (*), **f[i]** ha lo stesso valore di **h[i].** Allora la nostra espressione di π si può scrivere

```
4g[1]-2g[4]-g[5]-g[6]
```

e verificare usando il *package* per le somme simboliche citato in precedenza.

In[6]:=
```
Needs["Algebra`SymbolicSum`"]
Simplify[PowerExpand[
   4 g[1] - 2 g[4] - g[5] - g[6]]]
```

Out[7]=
```
Pi
```

La milionesima cifra esadecimale di π

Dato un numero reale s (espresso per semplicità in base 10) la sua espansione decimale a partire dalla d-esima cifra è uguale alla parte frazionaria di s 10^(d-1). La parte frazionaria si può calcolare facendo il modulo rispetto a 1 (in *Mathematica* la funzione **Mod** è definita anche per numeri reali).

In[10]:=
```
Frac[x_]:=Mod[x,1]
```

Proviamo con π.

In[11]:=
```
pi=N[Pi,30]
```

Out[12]=
```
3.14159265358979323846264338328
```

In[13]:=
```
Frac[pi 10^9]
```

Out[14]=
```
0.589793238462643383 28
```

Venendo alla nostra espressione, bisogna calcolare la parte frazionaria della formula:
```
Sum[16^(d-k)(4/(8k+1)-2/(8k+4)-
 1/(8k+5)-1/(8k+6)),{k,0,Infinity}]
```

Supponendo di contentarci di 5 cifre a partire dalla d-esima si può dimostrare che basta sommare fino a d+15.

Concentriamoci sul caso d=99 e vediamo varie soluzioni. Innanzitutto calcoliamo le cifre esadecimali esatte di π e mettiamole da parte.

In[15]:=
```
d=99;
BaseForm[Frac[N[16^d Pi,130]],16]
```

Out[16]//=
```
0.c29b7c98_16
```

N.B. come in ogni rappresentazione in base che si rispetti l'ultima cifra è arrotondata.

Effettuiamo ora il calcolo brutale, sommando le parti frazionarie dei vari addendi.

In[17]:=
```
BaseForm[Frac[Sum[Frac[N[
 16^(d-k)(4/(8k+1)-2/(8k+4)-
          1/(8k+5)-1/(8k+6)),
    130]],{k,0,d+15}]],16]
```

Out[17]=
```
0.c29b7c97c_16
```

Si ottengono 7 cifre esatte.

ATTENZIONE, fino a qui non c'è niente di nuovo! Questo metodo funziona per qualunque formula per π e con qualunque base. L'inconveniente è che il calcolo dei singoli addendi deve essere fatto in precisione piena.

Con la formula B.B.P. è possibile una notevole semplificazione: il calcolo di **Frac[16^k/h]** per k≥0, può essere ottenuto come **Frac[Mod[16^k,h]/h]** e questo ultimo calcolo si può fare in modo molto efficiente attraverso l'operazione **PowerMod**:

In[18]:=
```
?PowerMod
PowerMod[a,b,n] gives Mod[a^b,n]. For negative b,
PowerMod[a,b,n] gives modular inverses.
```

La somma si spezza in due parti: fino a **d** si usa **PowerMod**; gli addendi tra **d+1** e **d+15** si sommano brutalmente in semplice precisione. **SetPrecision** serve per forzare 7 cifre in uscita.

In[19]:=
```
BaseForm[SetPrecision[Frac[
 Sum[Frac[
  4. PowerMod[16,d-k,(8k+1)]/(8k+1) -
  2. PowerMod[16,d-k,(8k+4)]/(8k+4) -
  1. PowerMod[16,d-k,(8k+5)]/(8k+5) -
  1. PowerMod[16,d-k,(8k+6)]/(8k+6)],
      {k,0,d}]+
 Sum[16^(d-k)(4/(8k+1)-2/(8k+4)-
     1/(8k+5)-1/(8k+6)),
      {k,d+1,d+15}]],11],16]
```

Out[19]=
```
0.c29b7c97c16
```

Con una tecnica simile Wagon, Adamchik e Sofroniou hanno scritto un programma *Mathematica* più complicato ed efficiente che, sul mio PowerPc 601 a 80 MHz, ha calcolato la milionesima cifra di π in 18.000 secondi, lavorando con 16 cifre di precisione.

B.B.P. presentano programmi Fortran e C che calcolano la miliardesima cifra di π lavorando con 64 bit di aritmetica.

Una formula analoga per log 10/9

I (pochi) lettori che hanno resistito fino qua si saranno domandati: perche base16? Non esiste una formula per le cifre DECIMALI di π? Per ora nessuno sa se sia possibile derivare una formula in base 10 per π. L'unico esempio conosciuto è una semplice formula per log 10/9 che ha permesso a B.B.P. di battere il record per la cifra decimale più lontana in una espansione decimale calcolando la 5-miliardesima cifra. Vediamo la formula e facciamo qualcosa nel nostro piccolo.

```
Log[10/9]==
     Sum[10^-k/k, {k,1,Infinity}]
```

La funzione **long[d]** effettua il calcolo brutale di 5 cifre decimali a partire dalla (d+1)-esima.

In[1]:=
```
long[d_]:=Frac[N[Log[10/9],d+6] 10^d]
```

La funzione **fast[d]** effettua lo stesso calcolo con la tecnica

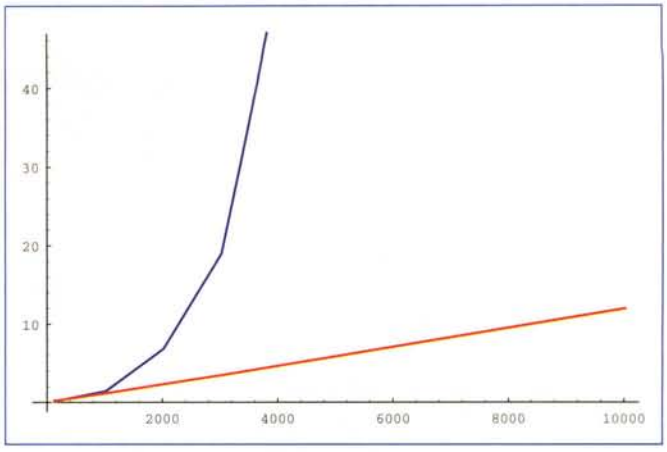


Figura 4

B.B.P. Stavolta è tutto più semplice perché la somma contiene un solo termine e non ci sono conversioni di base.

In[2]:=
```
fast[d_]:= Frac[
 Sum[Frac[N[
   PowerMod[10,d-k,k]/k]], {k,1,d}]+
 Sum[N[10^(d-k)/k], {k,d+1,d+5}]];
```

La seguente tabella mostra i tempi di esecuzione in secondi dei due algoritmi per d fino a 50000.

d	fast	long
100	0.10	0.03
1000	1.08	1.33
2000	2.25	6.76
3000	3.40	18.91
5000	5.83	88.03
10000	11.91	556.75
50000	62.53	60614.10

e un grafico mostra che **fast[d]** ha complessità lineare mentre **long[d]** diviene rapidamente inutilizzabile. **(Figura 4)**. MC

Bibliografia

V. Adamchik, S. Wagon. **π: A 2000-Year Search Changes Direction**, reperibile, all'indirizzo:

`http://www.wri.com /~victor/articles/pi/pi.html`

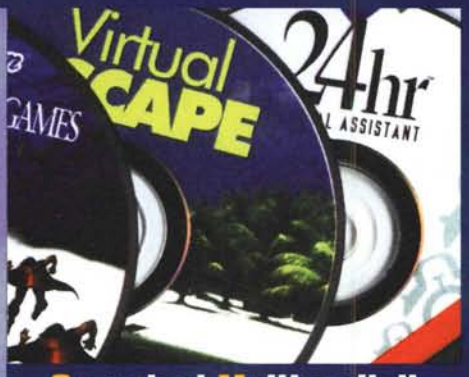
Bailey-Borwein-Plouffe **On the rapid computation of various polylogarithmic constants**, reperibile all'indirizzo:

`http://www.cecm.sfu.ca /personal/pborwein/PISTUFF/Apistuff.html`

R.E.Crandall, **Topics in Advanced Scientific Computation**, Springer/TELOS, New York, 1995.

S. Wolfram, *Mathematica.* **A System for Doing Mathematics by Computer, II Edition**, Addison Wesley, 1991.

MULTIMEDIA APPLICATIONS PRODUCTION & TECHNOLOGY



Creazioni Multimediali

Servizi Internet

Rendering e Animazioni 3D

Pubblicità, Grafica

L'universo della nuova comunicazione offre all'azienda possibilità difficilmente prefigurabili nella loro interezza e negli esiti dello sviluppo futuro. Il moltiplicarsi dei nuovi media, e la loro rapida evoluzione tecnologica, rendono evidente la necessità di un partner/consulente che sia in grado di controllare tutte le fasi della costruzione del messaggio. La possibilità di accedere ai servizi in tempi brevissimi, avvalendosi di operatori affidabili e professionali, costituisce parte dell'ampio ventaglio di proposte della Microforum Italia nell'ambito della multimedialità.

SMAU '96
Pad. 12
Stand A-27

Microforum® ITALIA

00161 Roma - Via Antonio Musa 13 - Tel. 06/44243033 - Fax 06/44242836 - email: italia@microforum.com
Ufficio di Milano: Tel. 02/22473137 - Fax 02/26226742
Http: //www.microforum.com