Guida a Mathematica

Parte seconda. Il formato dei numeri

Marcello Colozzo - http://www.extrabyte.info

Numeri interi, razionali, reali, complessi

Quando si introduce un numero, bisogna informare *Mathematica* sul tipo di numero introdotto, cioè se si tratta di un intero (**Integer**), di un razionale (**Rational**), di un reale (**Real**) o di un numero complesso (**Complex**). Ad esempio, scriviamo

```
2.0
```

```
2.
```

In tal modo abbiamo dato in input in numero reale 2. Se invece avessimo scritto:

```
2
```

```
2
```

Mathematica avrebbe interpretato tale input come l'intero naturale 2.

Il numero complesso $z = 7 + 2\,i$ si scrive:

```
7. + 2. I
```

```
7. + 2. i
```

o direttamente digitando l'unità immaginaria con le opportune scorciatoie da tastiera (si veda la guida in linea di Mathematica):

```
7. + 2. ⅈ
```

```
7. + 2. i
```

Una funzione built-in utilizzata per esplicitare la forma di un'espressione è **FullForm**:

```
FullForm[12 + 5 I]
```

```
Complex[12, 5]
```

È possibile controllare il formato dei numeri attraverso il comando **Head[]**. Precisamente, **Head[*x*]** restituisce il tipo di numero a cui appartiene *x*. Ad esempio:

```
Head[√3.]
```

```
Real
```

Se lasciamo inespresso **x**, **Head[x]** restituisce **Symbol**:

```
Head[x]
```

```
Symbol
```

La funzione **NumberQ[]** restituisce il valore logico **True** se il suo argomento è un qualunque formato numerico. Restituisce **False** in tutti gli altri casi.

```
NumberQ[√2.]
```

```
True
```

```
NumberQ[√2]
```

```
False
```

In questo caso, restituisce **False**, poichè *Mathematica* interpreta $\sqrt{2}$ alla stregua di un simbolo e non di un numero. In maniera simile, agisce **IntegerQ[x]** che controlla se **x** è un intero o meno.

```
IntegerQ[4]
```

```
True
```

```
IntegerQ[-4]
```

```
True
```

```
IntegerQ[4.]
```

```
False
```

Riconoscimento della parità di un intero naturale

La parità di un intero naturale è controllata dai comandi **EvenQ[]** e **OddQ[]**. Precisamente, il primo controlla se è pari, il secondo se è dispari.

```
EvenQ[2]
```

```
True
```

```
EvenQ[3]
```

```
False
```

```
OddQ[2]
```

```
False
```

Riconoscimento di numeri primi

Per il riconoscimento del tipo di numero (**Integer, Real, Complex, Prime**), *Mathematica* utilizza un comando del tipo ***Q[]**, dove al posto dell'asterisco va messo **Integer**, etc. Quindi, per i numeri primi il comando è **PrimeQ[]**. Ad esempio:

```
PrimeQ[11 003]
```

```
True
```

```
PrimeQ[11 009]
```

```
False
```

Approssimazioni numeriche

Dato un numero reale x, denotiamo con $[x]$ la sua parte intera, cioè se $x = n_1\, n_2 \ldots n_r.m_1\, m_2 \ldots$, si ha $[x] = n_1\, n_2 \ldots n_r$. Troncando la rappresentazione decimale di x alla $a-$esima cifra decimale, cioè ponendo $x = n_1\, n_2 \ldots n_r.m_1\, m_2 \ldots m_a$, si definisce *accuratezza* di tale nuova rappresentazione decimale, il numero intero positivo a, mentre $p = r + a$ definisce la *precisione* della suddetta rappresentazione

Ciò premesso, per default la precisione di *Mathematica* installato su una macchina Windows, è $p = 16$. Per visualizzarla, basta usare il comando:

```
$MachinePrecision
```

```
15.9546
```

La precisione può essere aumentata con il comando **SetPrecision[x,n]** che incrementa la precisione di x di n cifre. Ad esempio:

```
√2.
```

```
1.41421
```

```
SetPrecision[√2., 4]
```

```
1.414
```

Esistono diverse funzioni built-in che permettono di aver un controllo sul grado di approssimazione. La prima è **Floor**, che è la funzione parte intera: $f : x \to [x]$. La funzione **Ceiling** approssima all'intero superiore, mentre **Round** approssima all'intero superiore se l'ultima cifra è 5. Ad esempio:

```
a = 1.49;
```

```
Floor[a]
```

```
1
```

```
Ceiling[a]
```

```
2
```

```
Round[a]
```

```
1
```

```
b = 1.5;
```

```
Floor[b]
```

```
1
```

```
Ceiling[b]
```

```
2
```

```
Round[b]
```

```
2
```

Formato dei numeri

Le principali funzioni built-in che controllano il formato dei numeri sono: **ScientificForm[x]** e **EngineeringForm[x]**. La prima restituisce la classica notazione scientifica di un numero. Esempio:

```
ScientificForm[21.121]
```

$$2.1121 \times 10^{1}$$

EngineeringForm[x] è simile alla funzione precedente, con la differenza che l'esponente è divisibile per 3. Esempio:

```
EngineeringForm[81 200.00141]
```

$$81.2 \times 10^{3}$$

Approssimazione dei numeri reali con numeri razionali

Mathematica dà la possibilità di approssimare un numero reale attraverso un numero razionale. La funzione da utilizzare è **Rationalize[x,Δ]**, essendo **x** il numero e Δ l'errore (al più) commesso. Esempio:

```
Rationalize[√3, 10^-3]
```

$$\frac{71}{41}$$

Se richiediamo la rappresentazione decimale, dobbiamo applicare la funzione **N[]**; ricordando che ogni funzione richiedente un solo argomento può

essere richiamata in notazione postfissa, si ha:

```
Rationalize[Sqrt[3], 10^-3] // N
```

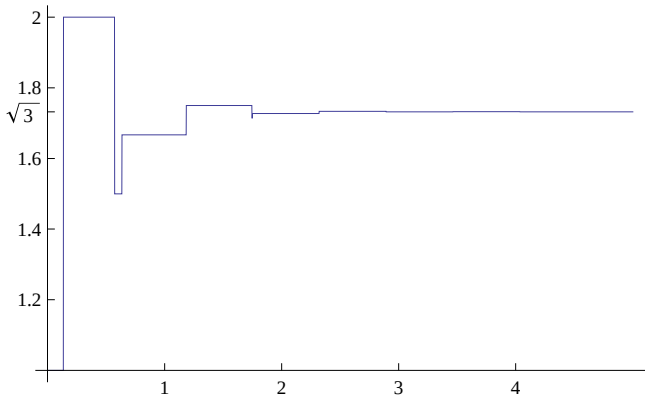
```
1.73171
```

È interessante studiare l'approssimazione di $\sqrt{3}$ al variare di Δ. A tale scopo utilizziamo il comando Evaluate[], che ci pemetterà di definire una funzione della precisione richiesta. Scriviamo:

```
f[n_] := Rationalize[Sqrt[3], 10^-n]
```

Tracciamo un grafico di tale funzione (vedremo più avanti il significato delle varie opzioni utilizzate in **Plot**):

```
Plot[
 (*espressione della funzione*)
 f[n],
 (*intervallo in cui plottiamo*)
 {n, 0, 5},
 (*codominio*)
 PlotRange → All,
 Ticks →
  {
   (*punti sull'asse delle ascisse *)
   {1, 2, 3, 4},
   (*punti sull'asse delle ordinate*)
   {1.2, 1.4, 1.6, 1.8, Sqrt[3], 2}
  }
]
```



La funzione **Chop[x,Δ]** restituisce 0 se **x** è un numero prossimo allo zero e se ne discosta di più di Δ. Ad esempio:

```
x = 5 - 4.999999999999
```

$$1.00009 \times 10^{-12}$$

```
Chop[x]
```

```
0
```

in quanto per default è $\Delta = 10^{-10}$.

```
Chop[x, 10^-12]
```

$$1.00009 \times 10^{-12}$$

L'uso di **Chop** risulta molto utile quando è necessario valutare espressione molto complicate. In tal caso **Chop** elimina gli errori di arrotondamento.