Guida a *Mathematica*

Parte prima. Le funzioni

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

Come è strutturato *Mathematica*

Mathematica è costituito da due componenti: il **Kernel** e il **Front End**.
Il Kernel (nucleo) è il motore di calcolo del programma, mentre il Front End è l'interfaccia utente. Durante una sessione di lavoro l'utente interagisce con il Kernel attraverso il Front End.
A loro volta, Kernel e Front End scambiano dati mediante un protocollo di comunicazione denominato *MathLink*. L'interazione tra il Kernel e il Front End avviene tramite le cosiddette "celle" di input. Al termine della computazione, il Kernel restituisce il risultato attraverso celle di output.

Sintassi

Mathematica è case-sensitive, cioè distingue le maiuscole dalle minuscole. Ad esempio, i nomi delle funzioni built-in sono maiuscoli, come pure i comandi. In generale, le istruzioni hanno la seguente sintassi: **`Command[expr1,expr2,...]`**, dove **`Command`** è un comando generico, mentre **`expr1`**,... sono espressioni che verrano valutate. È buona norma scrivere le funzioni definite dall'utente con lettere minuscole, in modo da distinguerle dalle funzioni built-in. Per quanto riguarda le funzioni, gli argomenti sono racchiusi tra parentesi quadre, anziché tonde. Ad esempio per la funzione sin(x) la sintassi è:

```
Sin[x]
```

```
Sin[x]
```

Una sessione di lavoro compone un *notebook* ovvero un set di celle di input e output. Se in una data fase di sessione caratterizzata dalla *k-esima* cella di output, si desidera avere in output il contenuto della cella *n-esima* con $n<k$, dobbiamo utilizzare il comando **`Out[n]`**, che restituisce l'*n-esima* cella. Alternativamente si può utilizzare la forma **`%%...%`**, dove il numero di puntini dipende da *n*. Ad esempio, volendo richiamare la funzione scritta in precedenza:

```
Out[1]
```

```
Sin[x]
```

Oppure:

```
%%
```

```
Sin[x]
```

La sintassi per le righe di commento è:

```
(*commento*)
```

Operatori aritmetici

Focalizziamo la nostra attenzione sulle operazioni aritmetiche ordinarie `+`, `-`, `*`, `/`, `^`. Il simbolo * utilizzato nella moltiplicazione può essere omesso a patto di lasciare uno spazio. Ad esempio:

```
a b
```

```
a b
```

è equivalente a:

```
a * b
```

```
a b
```

In alcuni casi particolari, lo spazio non è necessario. Ad esempio:

```
4 Sin[x]
```

```
4 Sin[x]
```

è equivalente a:

```
4 * Sin[x]
```

```
4 Sin[x]
```

È comunque preferibile lasciare uno spazio o utilizzare il simbolo *.

Costanti built-in

Le principali costanti built-in sono:

```
Infinity
```

```
∞
```

```
Pi
```

```
π
```

```
E
```

```
ⅇ
```

```
I
```

```
i
```

Per passare dal valore simbolico al valore numerico è necessario utilizzare la funzione **N[]**. Ad esempio

```
N[π]
```

```
3.14159
```

Operatori booleani

I simboli adoperati per i principali operatori booleani sono:
&& (And)
|| (Or)
! (Not)
Xor

Funzioni built-in

Passiamo in rassegna alcune funzioni built-in:

```
(*funzione logaritmo (in base e)*)
```

```
Log[x]
```

```
Log[x]
```

```
(*funzione logaritmo in base a*)
```

```
Log[a, x]
```

$$\frac{\text{Log}[x]}{\text{Log}[a]}$$

```
(*alcune funzioni trigonometriche*)
```

```
Sin[x]; Cos[x]; Tan[x]; Cot[x];
```

Per le altre funzioni built-in si consiglia di navigare nell'help in linea.

Notiamo di passaggio che il terminatore **;** cancella l'output. Diversamente avremmo visualizzato i singoli elementi

```
Sin[x];
```

```
Sin[x]
```

$$\text{Sin}[x]$$

Funzioni definite dall'utente

Assegnazione immediata

Esistono diversi modi per definire una funzione. L'approccio più immediato (*assegnazione immediata*), ha il seguente costrutto:

```
f[x_] = expr
```

$$\text{expr}$$

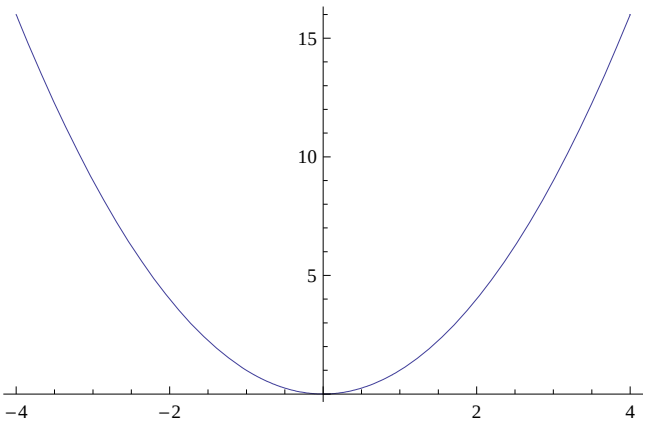
dove <expr> è l'espressione analitica della funzione. Ad esempio:

```
f[x_] = x^2
```

$$x^2$$

Una volta definita la funzione, possiamo determinare i valori assunti per opportuni valori della variabile indipendente, oppure possiamo graficarla in un dato sottoinsieme del campo di esistenza, calcolarne la derivata prima, seconda, etc. Ad esempio (studieremo più avanti l'istruzione **Plot**):

```
Plot[
 f[x],
 {x, -4, 4}
]
```



Assegnazione ritardata

Nell' *assegnazione ritardata* la funzione viene valutata solo quando viene richiamata dall'utente. La sintassi è:

```
f[x_] := expr
```

In output non viene restituito nulla. Per visualizzare l'espressione della funzione, dobbiamo dare in input **f[x]**

```
Clear[f]
```

```
f[x_] := x^2
```

```
f[x]
```

x^2

```
Clear[f]
```

Nel caso di più variabili:

```
f[x_, y_, z_] := Sin[x + y] - Exp[z]
```

etc.

Assegnazione condizionata

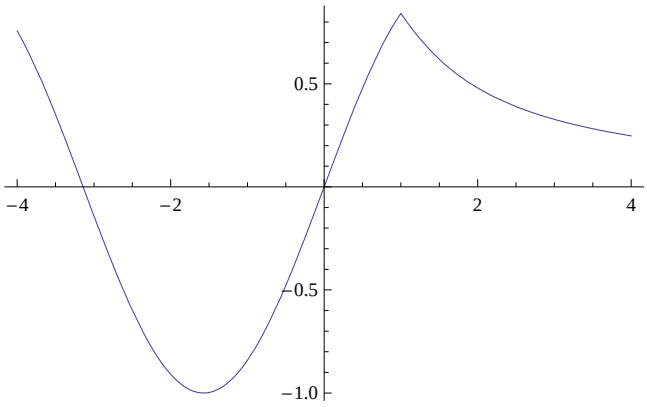
L'*assegnazione condizionata* si utilizza quando dobbiamo definire una funzione che ha espressioni diverse per differenti sottoinsiemi dell'insieme di definizione. Ad esempio, consideriamo la funzione reale di una variabile reale: $f(x) = \sin(x)$, se $x \in$ (– ∞ , 1] ; $f(x) = \sin(\frac{1}{x})$, se $x \in$ (1,+∞). Utilizziamo l'istruzione **/;** la cui sintassi è:

```
f[x_] := Sin[x] /; x ≤ 1
```

```
f[x_] := Sin[1/x] /; x > 1
```

Grafichiamo f

```
Plot[
 f[x],
 {x, -4, 4}
]
```

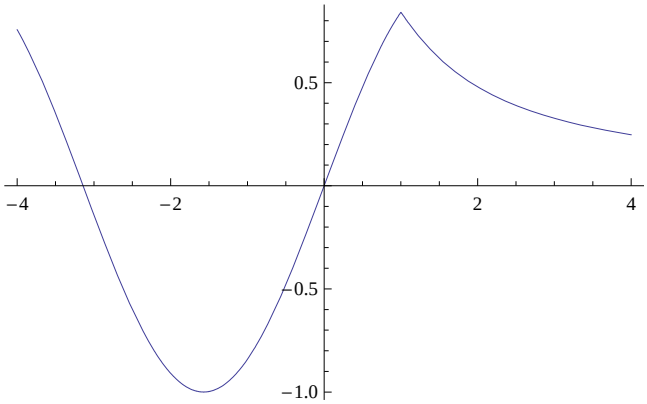


L'istruzione **Piecewise** ha lo stesso effetto:

```
Clear[f]
```

```
f[x_] := Piecewise[
  {
   {Sin[x], x ≤ 1},
   {Sin[1/x], x > 1}
  }
 ]
```

```
Plot[
 f[x],
 {x, -4, 4}
]
```

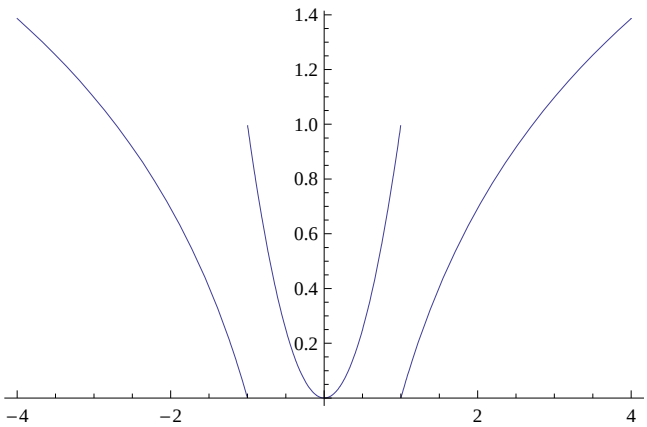


Spesso è richiesta l'azione degli operatori AND (**&&**) oppure OR (**||**) visti in una sezione precedente. Ad esempio:

```
Clear[f]

f[x_] := x^2 /; (x ≥ -1 && x < 1)
f[x_] := Log[Abs[x]] /; (x < -1 || x > 1)

Plot[
 f[x],
 {x, -4, 4},
 Exclusions →
  {
   x == 1, x == -1
  }
]
```

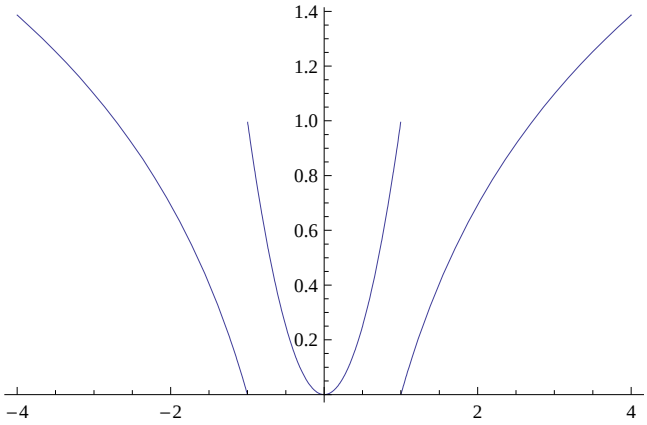



```
Clear[f]
```

La stessa funzione si definisce con l'istruzione **Piecewise:**

```
f[x_] := Piecewise[
  {
   {x^2, x ≥ -1 && x < 1},
   {Log[Abs[x]], x < -1 || x > 1}
  }
 ]
```

```
Plot[
 f[x],
 {x, -4, 4},
 (*escludiamo i punti di discontinuità
  di prima specie*)
 Exclusions →
  {
   x == 1, x == -1
  }
]
```




```
Clear[f]
```

Definizione di funzioni composte

Si consideri la funzione reale di variabile reale $f(x) = x^5 + 6x^4 - 2x + \ln(x^2 + 1)$. Per ipotesi x è una funzione della variabile reale t, cioè $x = x(t)$, per cui abbiamo la funzione composta

```
f[x_] := x^5 + 6 x^4 - 2 x + Log[x^2 + 1]
```

```
ReplaceAll[f[x], x → Sin[t]]
```

```
Log[1 + Sin[t]^2] - 2 Sin[t] + 6 Sin[t]^4 + Sin[t]^5
```

Tuttavia esiste una modalità postfissa per definire una funzione composta. Si tratta della potente istruzione **/.**

```
f[x] /. x → Sin[t]
```

```
Log[1 + Sin[t]^2] - 2 Sin[t] + 6 Sin[t]^4 + Sin[t]^5
```

```
Clear[f]
```

Nel caso di più variabili, in **/.** vanno utilizzate le parentesi graffe. Ad esempio, supponiamo di avere la funzione:

```
f[x_, y_] := Sqrt[x^2 - y^2]
```

con $x = \cos(t)$, $y = \sin(t)$, onde la funzione composta:

```
f[x, y] /. {x → Cos[t], y → Sin[t]}
```

```
Sqrt[Cos[t]^2 - Sin[t]^2]
```

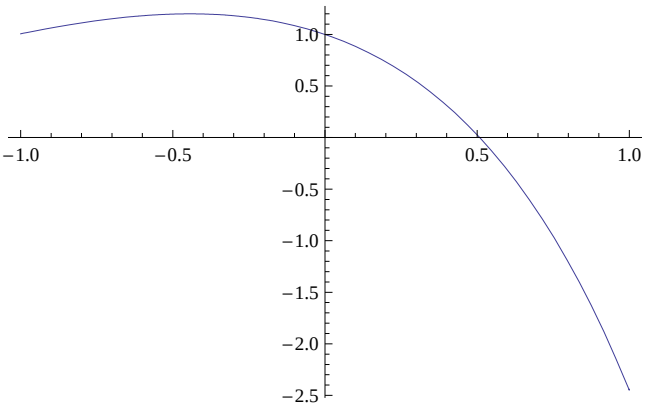
L'istruzione **/.** è estremamente efficace nel caso di plotting di integrali di equazioni differenziali. Vedremo più avanti che la soluzioni di equazioni differenziali (ODE) si effettua con l'istruzione **DSolve**. Supponiamo di avere l'equazione differenziale del secondo ordine: $y'' - 2y' + y = \sin(x)$. Determiniamo un integrale particolare che soddisfa le condizioni iniziali $y(0) = 1$, $y'(0) = -1$

```
sol = DSolve[
  {
   (*ODE*)
   y''[x] - 2 y'[x] + y[x] == Sin[x],
   (*condizioni iniziali*)
   y[0] == 1,
   y'[0] == -1
  },
  (*funzione incognita*)
  y[x],
  (*variabile indipendente*)
  x
 ]
```

$$\left\{\left\{y[x] \to \frac{1}{2}\left(e^{x} - 3\, e^{x}\, x + \text{Cos}[x]\right)\right\}\right\}$$

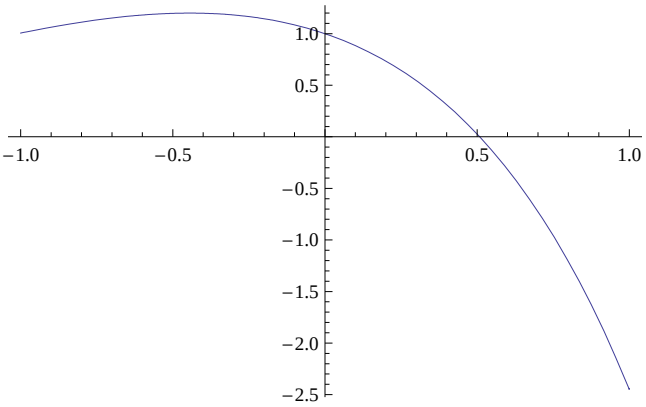
Grafichiamo tale integrale particolare:

```
Plot[
 y[x] /. sol,
 {x, -1, 1}
]
```



Utilizzando **ReplaceAll**:

```
Plot[
 ReplaceAll[y[x], sol],
 {x, -1, 1}
]
```



Operatori funzionali

In *Mathematica* le funzioni possono avere un qualunque argomento. Ciò si realizza attraverso l'underscore _ che restituisce un *pattern*. In parole povere, un simbolo del tipo **x_** rappresenta *qualunque cosa* e non necessariamente una variabile indipendente del tipo $x \in \mathbb{R}$ o $x \in \mathbb{C}$. Dal momento che l'argomento di una funzione può essere "qualunque cosa", possiamo definire una funzione in cui l'argomento è un'altra funzione:

```
Clear[f, f1]
```

```
f1[f_, x_] := f[x] + ArcTan[x]
```

```
f1[Sinh, x]
```

```
ArcTan[x] + Sinh[x]
```

```
Clear[f, f1]
```

```
f1[f_, x_] := f[Abs[x]] + ArcSin[x]
```

```
f1[Sqrt, ξ]
```

$$\sqrt{\text{Abs}[\xi]} + \text{ArcSin}[\xi]$$

Gli esempi appena visti sono abbastanza banali, giacché bastava definire direttamente la funzione. Tuttavia essi mostrano tutta l'efficacia dell'ambiente di calcolo *Mathematica*, nel senso che tale software è in grado di manipolare qualsiasi oggetto matematico, inclusi gli enti geometrici implementabili attraverso le cosiddette *direttive grafiche*. Ci occuperemo di ciò nella sezione dedicata all'insieme di Cantor. Per ora limitiamoci al calcolo di integrali indefiniti, anticipando che la ricerca di una primitiva di un'assegnata funzione $f(x)$ è implementata dall'istruzione **Integrate** che accetta come argomento principale la funzione di cui si vuole conoscere una delle infinite primitive, mentre il secondo argomento è la variabile di integrazione. A questo punto definiamo l'operatore funzionale:

```
F[f_] := Integrate[
   (*funzione da integrare*)
   f[x],
   (*variabile di integrazione*)
   x
  ];
```

```
F[Cos]
```

```
Sin[x]
```

```
F[Exp]
```

$$e^x$$

etc.

Una funzione può essere definita attraverso l'istruzione **Function** che ha due argomenti: il primo è la variabile indipendente, il secondo è il modo in cui la funzione opera sulla variabile, cioè la sua espressione analitica. Ad esempio:

```
Clear[f]
```

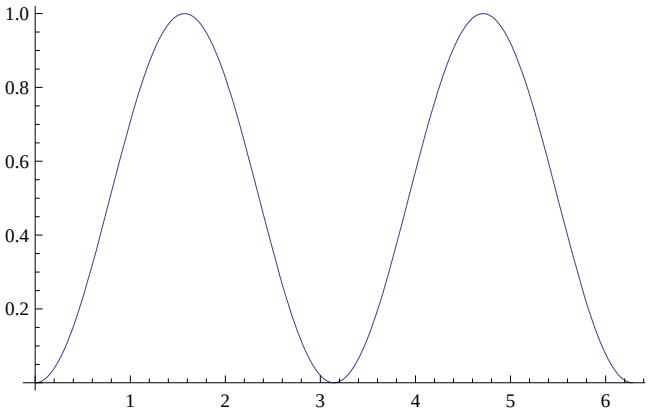
Funzioni anonime

```
f = Function[x, Sin[x]^2];
```

```
f[x]
```

$$\text{Sin}[x]^2$$

```
Plot[f[x], {x, 0, 2 π}]
```



Come è noto, la *x* è una variabile muta nel senso che può essere denotata con un qualunque simbolo:

```
f[t]
```

```
Sin[t]^2
```

```
f[ξ]
```

```
Sin[ξ]^2
```

```
Clear[f]
```

Tale circostanza suggerisce l'utilizzo di un simbolo universale per comunicare a *Mathematica* il modo in cui la funzione agisce sulla variabile indipendente. Più precisamente, usiamo **#** seguito da **&**. Consideriamo la funzione

```
f[x_] := x^3
```

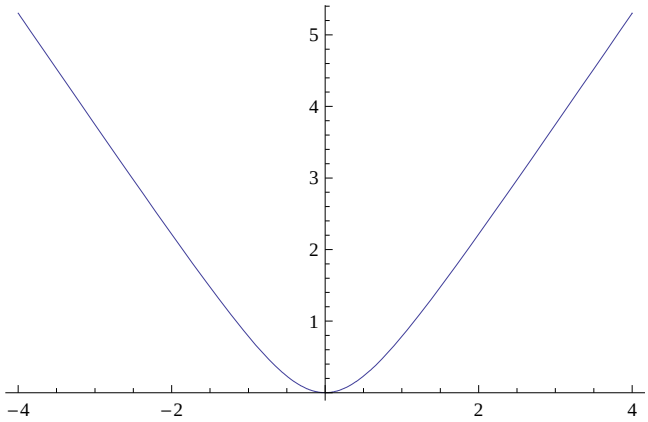
In questa nuova modalità scriviamo:

```
(#^3) &[x]
```

x^3

Abbiamo così definito una funzione "pura" o "anonima". Quest'ultimo termine è ereditato dal linguaggio di programmazione LISP.

```
Plot[# ArcTan[#] &[x], {x, -4, 4}]
```



Nel caso appena visto è preferibile definire la funzione nella modalità convenzionale. Come vedremo in seguito le funzioni pure sono vantaggiose nelle operazioni sulle liste o nella costruzione di un array. Nel caso di una funzione di più variabili si utilizzano i simboli **#1,#2,...** Ad esempio, supponiamo di voler esprimere la lunghezza dell'ipotenusa di un triangolo rettangolo in funzione delle lunghezze *x* e *y* dei cateti:

In[2]:=
```
ipotenusa[x_, y_] := √((#1^2 + #2^2) &[x, y])
```

In[3]:= `ipotenusa[2.5, √8]`

Out[3]= 3.77492