
Alcuni principi fondamentali

1. La sintassi e le regole di base

Il notebook è un foglio di lavoro interattivo; questo significa che, a differenza di quanto avviene in un comune editor di testi, ciò che l'utente scrive nel notebook deve avere un significato logico e semantico non solo per l'utente stesso ma anche e soprattutto per il sistema. In particolare, questo è esplicitamente richiesto per le linee di input dei comandi (celle di Input). In altre parole, se si desidera ricevere una risposta dal sistema, la domanda formulata deve rispettare una serie di regole di sintassi e semantica imposte dal sistema.

Mathematica è interamente scritto in Inglese; dunque, richiede una certa dimestichezza con la terminologia scientifica (in particolare matematica) in lingua Inglese. Da questa considerazione si può dedurre una prima indicazione su come ricordare i nomi delle funzioni. Si supponga di voler individuare le radici di un'equazione ma di non conoscere o ricordare il nome della funzione che in *Mathematica* effettua tale calcolo; in italiano si dice *risoluzione di una equazione*, in inglese si dice *equation solve* quindi da quanto detto ci si aspetta che la funzione in questione si chiami Solve; è proprio così: la funzione Solve calcola le radici di un'equazione (o di più equazioni nel caso dei sistemi); in inglese risolvere un integrale si dice *integrate*, da cui la corrispondente funzione in *Mathematica* si chiama Integrate.

Tutte le parole chiave del sistema vanno scritte con la **prima lettera maiuscola** (se hanno una sola lettera questa sarà maiuscola, ovviamente); nel caso di funzioni con nomi composti, entrambi i nomi devono essere scritti con la prima lettera maiuscola.

Mathematica ha un interprete **case sensitive**, ossia fa distinzione tra lettere minuscole e maiuscole. È bene ricordare sempre questa caratteristica, perchè spesso può capitare che un errore sia difficilmente individuato in quanto dipende da una semplice lettera maiuscola messa al posto di una minuscola (o viceversa). Ovviamente questo vale sia per i nomi degli oggetti nativi del sistema sia per quelli definiti dall'utente.

Nome	Significato
Limit	Limite di una funzione
N[abbreviazione di Numerical]	Valore numerico di un'espressione simbolica
GCD[Greatest Common Divisor]	Massimo comune denominatore tra interi
Derivative	Derivata prima di una funzione
CharacteristicPolynomial	Polinomio caratteristico di una matrice

Tabella 1. Nome e significato di alcune funzioni di *Mathematica*

In virtù del fatto che i notebook possono essere anche utilizzati come documenti di testo, il tasto `RET` (invio) viene interpretato dal front end come un *carriage return* (carattere di *a capo*); per questo motivo quando, invece, si intende far valutare il contenuto di una cella bisogna premere contemporaneamente i tasti `SHIFT+RET`, per indicare al front end che l'intero contenuto della cella deve essere mandato al kernel per l'interpretazione dei comandi in esso inclusi.

Si supponga di voler calcolare le radici di una equazione di secondo grado del tipo $x^2 - 4 == 0$.

Seguendo il ragionamento fatto in precedenza basta ricordare che la funzione che ci interessa è la Solve; a questo punto appare evidente che quando si chiede al sistema una certa computazione bisogna indicare almeno due cose: l'operazione da effettuare (Solve nell'esempio) e l'argomento su cui effettuarla (l'equazione $x^2 - 4 == 0$). Il passaggio degli argomenti avviene mediante la coppia di parentesi quadrate [] che ne delimitano

l'inizio e la fine.

Quindi, la sintassi generale per la chiamata di una funzione è la seguente

```
NomeFunzione[argomento1, argomento2, ..., argomenton]
```

Una volta completata la scrittura della richiesta basta premere i tasti `SHIFT+RET` (Shift + invio) contemporaneamente. Si avrà così

```
In[1]:= Solve[x^2 - 4 == 0]
Out[1]:= {{x -> -2}, {x -> 2}}
```

La prima cosa che si nota è che la cella di input con il comando `Solve` viene etichettata con una `label` (etichetta) del tipo `In[1]:=` e la successiva, quella con la risposta generata dal kernel, viene etichettata con una `label` del tipo `Out[1]:=`; questo sta ad indicare che tutte le operazioni che vengono effettuate durante una sessione di lavoro vengono etichettate e memorizzate dal sistema in ordine cronologico. Tale ordine torna utile nel momento in cui si vuole recuperare l'output di una computazione già eseguita in precedenza, senza dover ripetere le operazioni che hanno generato tale output; infatti mediante l'uso della combinazione di simboli `%n` si può ottenere nuovamente l'output numero `n`, senza che venga rielaborato il corrispondente input.

Se, in un qualunque momento della sessione di lavoro, si vogliono riutilizzare le radici dell'equazione, mostrate nella cella etichettata `In[1]:=` basta scrivere:

```
In[2]:= %1
Out[2]:= {{x -> -2}, {x -> 2}}
```

Si riporta ora una serie di informazioni utili per l'inserimento dei simboli speciali e delle espressioni bidimensionali direttamente dalla tastiera (alternativamente si possono utilizzare le palette).

Ad esempio l'espressione seguente

```
In[3]:= ∫ Log[1 + ξ] / √ξ dξ
Out[3]:= 4 ArcTan[√ξ] + 2 √ξ (-2 + Log[1 + ξ])
```

si può scrivere utilizzando la sequenza di tasti di seguito riportata (in rosso sono sottolineate le sequenze di tasti che corrispondono a simboli speciali; si noti che le sequenze del tipo `ESC` int `ESC` indicano che i tasti vanno digitati l'uno dopo l'altro, mentre quelle del tipo `SHIFT` `CTRL` `2` indicano che i tasti vanno premuti contemporaneamente nella sequenza specificata)

```
ESC int ESC Log[1+ ESC x ESC ] SHIFT CTRL / SHIFT CTRL 2 ESC x ESC SHIFT CTRL SPACE ESC dd ESC ESC x ESC
```

Ancora un'altra espressione bidimensionale

```
In[4]:= ∑_{k=1}^∞ 1/k^2
Out[4]:= π^2/6
```

ottenibile tramite la sequenza

`\ESC sum \ESC \SHIFT \CTRL = k=1 \SHIFT \CTRL % \ESC inf \ESC \SHIFT \CTRL SPACE 1 \SHIFT \CTRL / k \CTRL 6 2 \SHIFT \CTRL SPACE`

Per l'inserimento delle espressioni bidimensionali (quelle che prevedono caratteri in posizioni particolari come ad esempio apici, pedici, estremi di integrazione, indici delle sommatorie ecc.) in genere è indispensabile ricorrere all'utilizzo dei tasti di controllo quali `\SHIFT` e `\CTRL`. Per l'inserimento dei simboli, invece, oltre alla possibilità di utilizzare i tasti di controllo vi è la possibilità di utilizzare solo caratteri standard della tastiera, sfruttando la sintassi seguente

`\[Nomesimbolo]`

appena terminata di scrivere la parentesi di chiusura nella forma sopra riportata, il front end provvede a sostituire quanto scritto con il corrispondente simbolo. Si provi a completare le seguenti espressioni

`\[ControlKey`

`\[ShiftKey`

`\[Integral`

`\[ContourIntegral`

`\[Sum`

`\[Times`

`\[Product`

Sequenza	Simbolo
<code>\[EscapeKey]</code>	<code>\ESC</code>
<code>\[RightTriangle]</code>	▷
<code>\[Placeholder]</code>	□
<code>\[SelectionPlaceholder]</code>	■
<code>\[Infinity]</code>	∞
<code>\[Implies]</code>	⇒
<code>\[Epsilon]</code>	ε
<code>\[Element]</code>	∈
<code>\[ForAll]</code>	∀
<code>\[Exists]</code>	∃
<code>\[And]</code>	∧
<code>\[Or]</code>	∨
<code>\[CirclePlus]</code>	⊕
<code>\[CircleTimes]</code>	⊗

Tabella 0.2. La sequenza di caratteri corrispondente ad alcuni simboli

Infine, si ricorda che le lettere dell'alfabeto greco si ottengono anche tramite la sequenza di `\ESC` lettera `\ESC`

Sequenza	Simbolo
<code>ESC a ESC</code>	α
<code>ESC b ESC</code>	β
<code>ESC c ESC</code>	χ
...	...
<code>ESC y ESC</code>	ψ

Tabella 0.3. L'alfabeto greco

Per un elenco completo dei caratteri e simboli speciali si veda il *The Mathematica Book* all'Appendice A.12.1.

Un'altra regola di sintassi da ricordare è l'utilizzo delle parentesi. *Mathematica* interpreta le parentesi in maniera differente rispetto alla notazione tradizionale scientifica. Infatti, ad esempio, per indicare una funzione di una variabile bisognerà scrivere $f[x]$ e non $f(x)$. La tabella che segue riepiloga il significato delle tre forme di parentesi per *Mathematica*

Forma	Significato	Esempio
()	Raggruppamento nelle espressioni algebriche	$x(y+z) \rightarrow xy + xz$
[]	Passaggio di argomenti per le funzioni	<code>Solve[x² - 4 == 0]</code>
{}	Delimitatore delle liste	{1, 2, 3}, {4, 5, 6}, {6, 7, 8}
[[]]	Recupera la parte di un oggetto (ad esempio lista)	<code>{a, b, c}[[2]]</code> \rightarrow a
(* *)	Delimitatore dei commenti nelle righe di input	<code>var = 2 + 3 (* var memorizza il risultato *)</code>

Tabella 0.4. Significato delle parentesi in *Mathematica*

Dunque, scrivere $f(2)$ significa moltiplicare la funzione f per 2, mentre $f[2]$ significa valutare la funzione f nel punto 2.

L'utilizzo ed il significato delle parentesi viene anche riportato nel *The Mathematica Book* alle sezioni 1.2.5 e A.2.6.

Alcuni principi fondamentali

2. L'help in linea

Per un sistema come *Mathematica*, che dispone di migliaia di oggetti di cui ricordarne la sintassi, la funzionalità, il numero degli argomenti, eccetera, è ovviamente indispensabile offrire anche un manuale o guida in linea che supporti l'utente nel rivedere le specifiche di ciascun oggetto, ogni volta che lo si ritiene opportuno. *Mathematica* offre tre differenti livelli di help (aiuto), differenziati per il livello di informazioni che forniscono. Vi sono due tipi di help in linea, ottenibile direttamente nel notebook di lavoro. Se si vuole sapere semplicemente il comportamento di una funzione e quali argomenti utilizzare si può chiedere un help di primo livello utilizzando il simbolo ?

```
In[5]:= ?Solve
Solve[eqns, vars] attempts to solve an equation or set of equations
for the variables vars. Solve[eqns, vars, elims] attempts to solve
the equations for vars, eliminating the variables elims. More...
```

Se invece si vogliono sapere ulteriori informazioni sulla funzione si può chiedere un help più descrittivo utilizzando il simbolo ?? che fornisce, oltre al messaggio precedente, anche qualche indicazione aggiuntiva sulle caratteristiche della funzione

```
In[6]:= ?? Solve
Solve[eqns, vars] attempts to solve an equation or set of equations
for the variables vars. Solve[eqns, vars, elims] attempts to solve
the equations for vars, eliminating the variables elims. More...

Attributes[Solve] = {Protected}

Options[Solve] = {InverseFunctions → Automatic, MakeRules → False, Method → 3,
Mode → Generic, Sort → True, VerifySolutions → Automatic, WorkingPrecision → ∞}
```

I comandi ? e ?? permettono anche l'uso del carattere jolly, l'asterisco *. Se non si ricorda il nome completo della funzione di cui si cercano spiegazioni, si può utilizzare anche solo una parte del nome, sostituendo la parte non nota con il carattere jolly *.

Se ad esempio non si ricorda per esteso il nome Solve, basta scrivere:

```
In[3]:= ?? Sol*
System`
Solve SolveAlways SolveDelayed
```

ovviamente in tal caso la risposta non è più la spiegazione sul funzionamento della Solve, perchè non avendo specificato per esteso il suo nome, l'help in linea risponde con un elenco di nomi di funzioni e/o oggetti di *Mathematica* che soddisfano il criterio di ricerca impostato, ossia nomi che cominciano con Solv, in questo caso. Con la release 4.1 si ha una funzionalità in più, ossia i nomi delle funzioni che soddisfano il criterio di ricerca impostato (nell'esempio Solve, SolveAlways e SolveDelayed) vengono riportati come collegamenti ipertestuali ai corrispondenti messaggi di help in linea. Basta cliccare su uno dei nomi sopra comparsi per visualizzarne l'help corrispondente.

L'help in linea è rivolto ad utenti con una certa esperienza, in quanto non fornisce spiegazioni aggiuntive oltre alla sintetica descrizione sull'uso standard dei vari comandi. Per chi ha ancora poca dimestichezza, spesso può risultare indispensabile leggere ulteriori spiegazioni sui comandi o esempi pratici sul loro utilizzo. Per questo, *Mathematica* mette a disposizione una serie di help maggiormente descrittivi, ossia dei veri e propri manuali in formato elettronico.

Ad esempio, se si desidera apprendere maggiori dettagli sull'uso della funzione Solve basta andare nel menu

Help > HelpBrowser e comparirà la finestra di visualizzazione/consultazione dell'help come riportato nella seguente figura

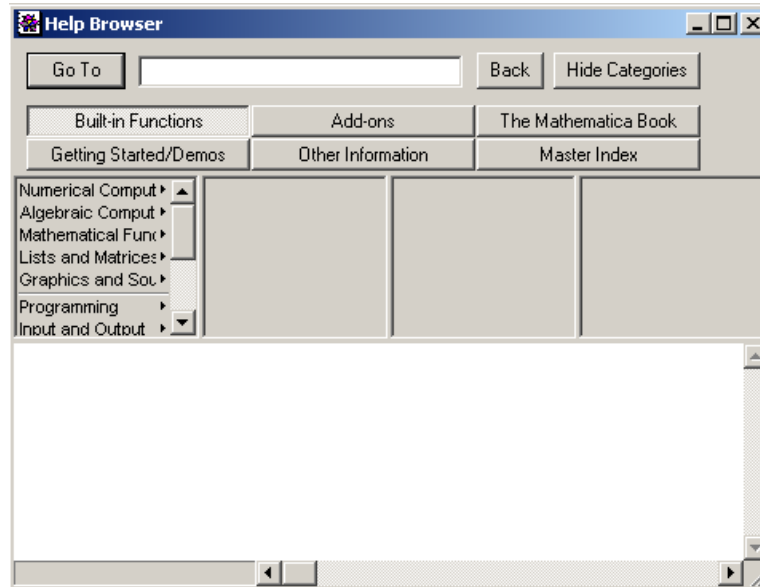


Figura 0.1. La finestra di dialogo dell'HelpBrowser.

all'interno della quale si ha un campo di testo (in alto a destra del bottone con la scritta Go To)



Figura 0.2. Il riquadro dove digitare le parole da ricercare nella manualistica.

dove l'utente può scrivere una qualunque parola chiave di *Mathematica* e successivamente vederne la guida disponibile che apparirà nel riquadro principale.

I manuali di guida all'utente forniti dall'help browser sono sei, come si evidenzia dalla presenza di altrettante etichette selezionabili (una alla volta)

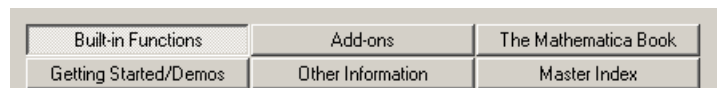


Figura 0.3. I sei differenti tipi di manuali disponibili nell'HelpBrowser.

La tabella che segue riepiloga il contenuto di ciascuna categoria di guida

Categoria	Contenuto
Built – in Function	Utilizzo delle funzioni native
Add – ons	Utilizzo dei package aggiuntivi forniti con Mathematica
The Mathematica Book	Manuale completo sul sistema Mathematica
Getting Started/Demos	Guida introduttiva a Mathematica con esempi e spiegazioni sui comandi più comuni
Other information	Utilizzo del front end ed informazioni dell' ultima ora non presenti nelle altre guide
Master Index	Indice analitico di tutti gli oggetti di Mathematica

Tabella 0.5. Descrizione dei contenuti delle guide in linea

Nella figura che segue viene riportato il risultato della ricerca del comando Solve nell'help browser dove si è selezionato il manuale Built-in Function

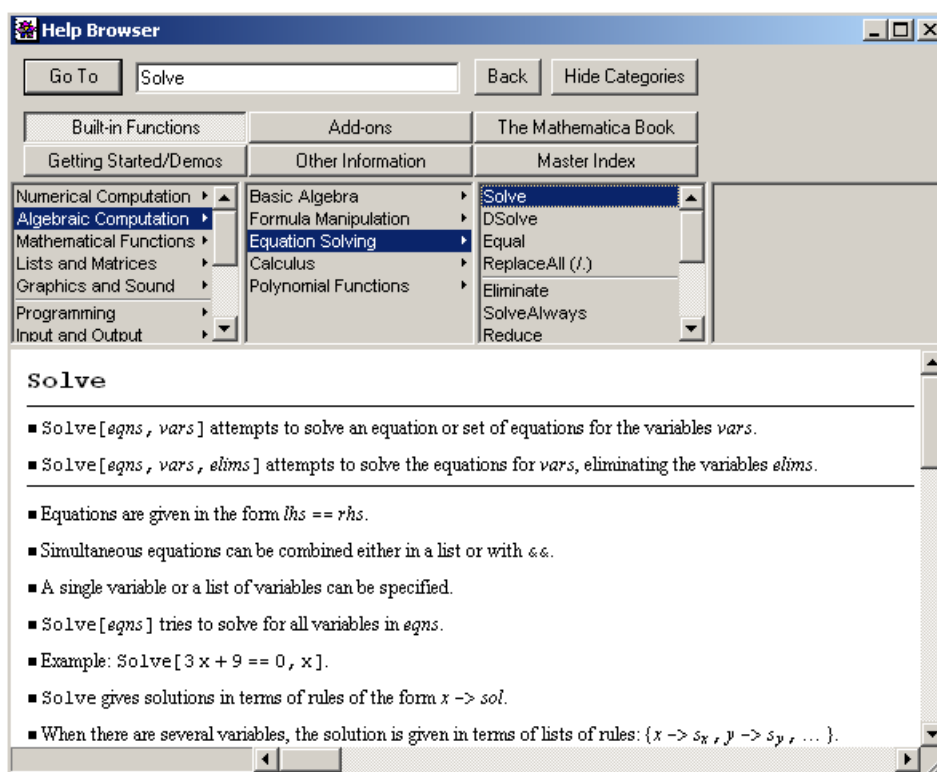


Figura 0.4. Le finestre di navigazione riportano gli argomenti corrispondenti alla parola inserita.

Un altro modo per accedere in maniera immediata alla descrizione di un comando nella categoria Built-in Function è l'utilizzo del tasto funzione F1. Si scrive il nome di una funzione di *Mathematica*, ci si posiziona immediatamente alla fine della parola e si preme il tasto F1. Si provi sulla riga successiva

Limit

L'uso dell'help browser è importante non solo per rivedere cose già note, ma è un ottimo strumento anche per studiare le funzionalità del sistema, in quanto le spiegazioni inserite non si fermano alla semplice regola di sintassi per le funzioni di sistema, ma ne spiegano anche i vantaggi, i limiti, e soprattutto mostrano esempi aggiuntivi sul loro uso. Se, ad esempio, si vuole avere una panoramica su quanto è possibile realizzare in termini di grafica con il *Mathematica* si può sfogliare il paragrafo Getting Started/Demos > Graphics gallery. Infine, è da notare che la finestra principale dell'help browser è in realtà uno speciale notebook, quindi è possibile sperimentare i comandi letti nel manuale direttamente all'intero delle pagine mostrate dall'help (non ci si deve preoccupare delle modifiche e/o cancellazioni delle linee di codice e/o dei testi dell'help in quanto all'uscita il notebook non viene salvato ed è anche protetto da eventuali salvataggi accidentali).

Un altro piccolo aiuto è fornito dal completamento automatico delle parole e delle espressioni. Dopo aver iniziato a scrivere parte del nome di una funzione si può digitare la sequenza di caratteri `[CTRL] {k}`. Se esiste una sola parola riconosciuta da *Mathematica* che inizia con le stesse lettere digitate, allora viene automaticamente completata la parola; se, invece, vi sono più parole che iniziano con tali lettere, allora viene mostrato un riquadro con l'elenco di tali parole, da cui si può scegliere quella desiderata.

Integ

Per completare automaticamente la parola basta indicare con il puntatore del mouse la parola desiderata o selezionare la parola utilizzando le frecce di scorrimento della tastiera e successivamente dando digitando `[RET]`.

Un'altra funzionalità del front end utile come aiuto durante la scrittura dei comandi è il completamento automatico dell'espressione, analogo al completamento precedente. Una volta completata la parola codice si può chiedere il completamento dell'espressione da utilizzare con la sequenza di tasti `[SHIFT] [CTRL] {k}`. Questa

funzionalità è molto utile ad esempio per i comandi della grafica, che solitamente richiedono una struttura abbastanza articolata dell'input. Provare a digitare la sequenza di caratteri sopra indicata sulla linea di input seguente.

```
ParametricPlot3D
```

Si noti che il completamento automatico del nome funziona anche per i simboli speciali, quelli che possono essere introdotti con la forma `\[NomeSimbolo]`. Si provi ad esempio a digitare `CTRL+k` sulla linea seguente

```
\[Conto
```

Alcuni principi fondamentali

3. I messaggi di errore

Un aspetto fondamentale per una buona comprensione del comportamento del sistema è l'interpretazione delle risposte ed i messaggi di errore.

Il concetto fondamentale è che il sistema tenta in ogni modo di soddisfare le richieste dell'utente, operando sugli argomenti nel miglior modo possibile, e comunicando il tipo di errore all'utente quando non è in grado di portare a termine la computazione o quando durante l'elaborazione della risposta il kernel rileva problemi o situazioni anomale. *Mathematica* è un linguaggio interpretato, ossia una volta indicata una sequenza di linee di codice dalla riga di input, il kernel esamina una linea alla volta. In questo modo, anche se un comando contiene un errore, il kernel finiva la sua computazione, passa automaticamente alla successiva. Questo comporta che spesso un singolo errore pregiudica il buon esito di tutta una serie di comandi, in particolare quando l'output di un comando viene richiesto come input per uno dei comandi successivi, il che causa una serie di messaggi di errore. Per questo motivo è bene sempre leggere i messaggi di errore a partire dal primo messaggio generato in quanto, quasi sempre, al primo messaggio corrisponde l'errore principale occorso (alle volte è l'unico causato dall'utente). Il primo messaggio mostra le informazioni più importanti per la correzione dell'input, gli altri sono generati automaticamente dall'esecuzione che prosegue, quindi spesso sono poco significativi.

I messaggi di errore che *Mathematica* adotta sono numerosissimi e vengono raggruppati in diverse categorie, a seconda della tipologia di problema riscontrato (sintassi non corretta, problemi dell'algoritmo interno, mancanza di dati per la computazione, ecc).

Si riporta di seguito una sequenza di operazioni esemplificative che mostra il differente comportamento del sistema, a seconda degli errori occorsi. Si osservi cosa succede quando viene digitata una sequenza di caratteri totalmente insignificante dal punto di vista dei comandi di *Mathematica*

```
In[7]:= oiehof wefo wijogiesjgo jfv g dskfv skdmòl
Out[7]= dskfv jfv g oiehof skdmòl wefo wijogiesjgo
```

Probabilmente ci si aspettava che il sistema restituisse un messaggio di errore e che non effettuasse alcuna operazione: è accaduto esattamente l'opposto!

In effetti, sebbene l'espressione non assume nessun significato dal punto di vista dell'utente, per il kernel essa corrisponde ad un'operazione ben precisa (scritta correttamente dal punto di vista della sintassi), ossia alla moltiplicazione delle variabili *oiehof*, *wefo*, *wijogiesjgo*, *jfv g*, *dskfv* e *skdmòl* dunque l'output riporta il risultato, ovviamente simbolico, della moltiplicazione. Si noti che l'ordinamento delle variabili viene eseguito automaticamente dalla funzione Times (la moltiplicazione).

Cosa succede quando se per caso si introduce, oltre alle lettere, qualche carattere che il sistema attribuisce a particolare funzioni ?

```
In[8]:= dfasfaskfl : dasda po. * djwoidjup oproqwporiwp
Syntax::sntxf : "dasda po." cannot be followed by "*djwoidjupoproqwporiwp". More...
dfasfaskfl : dasda po. * djwoidjup oproqwporiwp
```

```
In[8]:= ajsdf /. dfhhd o
ReplaceAll::reps : {dfhhd o} is neither a list of replacement rules
nor a valid dispatch table, and so cannot be used for replacing. More...
Out[8]:= ajsdf /. dfhhd o
```

Si può intuire così la differenza del comportamento del kernel rispetto all'esecuzione precedente. In quest'ultimo caso tra le varie parole sono stati inseriti alcuni simboli, in particolare il punto . e la sequenza backslash punto / . , quindi il kernel ha tentato di eseguire le operazioni associate a tali simboli, andando chiaramente in errore perché gli argomenti passati a tali funzioni non sono nella forma corretta.

Sintetizzando, gli esempi precedenti lasciano intuire che, nella maggior parte dei casi, il sistema è in grado di interpretare l'input, capire gli errori e quindi aiutare l'utente nella correzione tramite l'uso di messaggi appropriati.

Si riporta ora una serie di esempi in cui l'interpretazione dei messaggi guida l'utente ad individuare gli errori occorsi.

```
In[9]:= Solve[x2 + 3 x - 2]
Solve::eqf : -2 + 3 x + x2 is not a well-formed equation. More...
Solve::eqf : -2 + 3 x + x2 is not a well-formed equation. More...
Out[9]:= Solve[-2 + 3 x + x2]
```

In questo caso il messaggio parla chiaramente di un errore nella scrittura dell'equazione, quindi va interpretato nel modo seguente: la funzione Solve è stata riconosciuta (il messaggio porta l'etichetta `Solve::eqf`) quindi la sua sintassi è corretta; è il suo argomento che non è scritto nel modo giusto perché l'espressione $x^2 + 3x - 2$ non è affatto un'equazione ma una semplice espressione algebrica, mancando del simbolo che identifica le equazioni (`==`) ed il secondo membro dell'equazione stessa. In questo caso, il messaggio di errore suggerisce in maniera evidente la correzione da apportare.

Si supponga, invece, di scrivere la seguente espressione

```
In[10]:= Solvi[x2 + 3 x - 2 == 0]
General::spell1 : Possible spelling error: new
symbol name "Solvi" is similar to existing symbol "Solve". More...
Out[10]:= Solvi[-2 + 3 x + x2 == 0]
```

Questa volta il messaggio parla di una nuova parola (Solvi) che è molto simile a quella che il kernel conosce (Solve) quindi il sistema non è in grado di stabilire se è stato commesso un errore di scrittura della parola Solve oppure se si voleva effettivamente utilizzare (o creare) una nuova funzione dal nome Solvi. Infatti, il messaggio etichettato `General::spell1` si riferisce ad un *Possible spelling error* (quindi ipotetico) errore di spelling. Si ricorda che *Mathematica* è anche un linguaggio di programmazione, quindi permette di definire nuove variabili e/o funzioni; questo spiega perché il sistema non riesce ad identificare con certezza una situazione di errore come nell'esempio.

Se l'intenzione era quella di calcolare le radici dell'equazione, dunque di utilizzare effettivamente la funzione Solve, verificando la sintassi della cella di input ci si rende conto che al posto della *e* si è scritta la *i*.

Infine, una volta eliminati tutti gli errori, si ottiene la risposta corretta:

```
In[11]:= Solve[x^2 + 3 x - 2 == 0]
Out[11]:= {{x -> 1/2 (-3 - Sqrt[17])}, {x -> 1/2 (-3 + Sqrt[17])}}
```

Si riportano altri esempi di messaggi

```
In[12]:= Sin[x
Syntax::bktmcp : Expression "Sin[x" has no closing ". More...
Sin[x
```

Si noti che alcune volte, soprattutto nel caso degli errori di sintassi, il messaggio di errore si conclude con un'espressione che replica l'input errato, sottolineata in rosso nella parte dove si è verificato l'errore.

Esempi di messaggi che mostrano una particolare condizione o caratteristica dei dati in ingresso alla funzione, che viene riscontrata durante l'elaborazione. Il messaggio in questo caso non è tanto di errore quanto di attenzione nell'interpretazione della risposta fornita dal kernel.

```
In[12]:= Integrate[1/x, {x, -1, 2}]
Integrate::idiv : Integral of 1/x does not converge on {-1, 2}. More...
Out[12]:= Integrate[1/x, {x, -1, 2}]
```

```
In[13]:= DSolve[theta''[t] + Sin[theta[t]] == 0, theta, t]
Solve::ifun : Inverse functions are being used by Solve, so some solutions
may not be found; use Reduce for complete solution information. More...
Out[13]:= {{theta -> Function[{t}, 2 JacobiAmplitude[-1/2 Sqrt[(2 + C[1]) (t + C[2])^2], 4/(2 + C[1])]],
{theta -> Function[{t}, 2 JacobiAmplitude[1/2 Sqrt[(2 + C[1]) (t + C[2])^2], 4/(2 + C[1])]]}}
```

```
In[14]:= NIntegrate[1/Sqrt[Abs[x]], {x, -1, 1}]
Out[14]:= 4.
```

In quest'ultimo esempio il problema evidenziato dal messaggio e che causa la non valutazione dell'integrale può essere aggirato indicando esplicitamente il punto di singolarità nell'intervallo di integrazione.

```
In[15]:= NIntegrate[1/Sqrt[Abs[x]], {x, -1, 0, 1}]
Out[15]:= 4.
```

Mathematica permette anche di inibire i messaggi di errore se lo si desidera, ad esempio come nel caso dei messaggi di *General::spell*, che possono risultare inutili qualora si riferiscono alla definizione di variabili con nomi simili e l'utente ne è consapevole. Ad esempio

```
In[16]:= variabileA = 0
Out[16]:= 0
```

```
In[17]:= variabileB = 1
General::spell1 : Possible spelling error: new symbol
name "variabileB" is similar to existing symbol "variabileA". More...
Out[17]= 1
```

```
In[18]:= variabileC = 2
General::spell : Possible spelling error: new symbol name "variabileC" is
similar to existing symbols {variabileA, variabileB}. More...
Out[18]= 2
```

Per inibire la generazione di un certo messaggio basta utilizzare la funzione `Off`

```
In[19]:= Off[General::spell]
In[20]:= variabileD = 3
Out[20]= 3
```

Infine, un'ultima considerazione sul modo in cui *Mathematica* supporta l'utente nella scrittura delle espressioni input. Quando si scrive un'espressione le parentesi aperte per le quali non si è digitata ancora la corrispondente parentesi chiusa, viene mostrata con un colore diverso, come nell'esempio che segue

```
Solve[{x2 - 4 y == 0, x2 + 2 y == -1
```

Questa tecnica, molto utile quando si scrivono espressioni contenenti più funzioni nidificate, permette di evidenziare all'utente le parentesi che devono ancora essere chiuse, ma serve anche ad evidenziare eventuali mancanze. Infatti, se si dimentica di chiudere una parentesi ad un certo livello, tutte le parentesi ai livelli superiori rimarranno colorate, come si mostra nell'esempio che segue

```
Solve[{x2 - 4 y == 0, x2 + 2 y == -1, {x, y}}
```

avendo dimenticato di inserire la parentesi di chiusura della lista di equazioni, la successiva parentesi di chiusura della `Solve` rimane colorata in rosso, sebbene sia abbinata a quella di apertura. Questo suggerisce all'utente di verificare la correttezza dell'espressione prima di mandarla in esecuzione e generare un errore di sintassi.

```
In[21]:= Solve[{x2 - 4 y == 0, x2 + 2 y == -1, {x, y}}
Syntax::bktmcp :
Expression "{x2 - 4 y == 0, x2 + 2 y == -1, {x, y}" has no closing "}". More...
Solve[{x2 - 4 y == 0, x2 + 2 y == -1, {x, y}}]
In[21]:= Solve[{x2 - 4 y == 0, x2 + 2 y == -1}, {x, y}]
Out[21]= {{y -> -1/6, x -> -i Sqrt[2/3]}, {y -> -1/6, x -> i Sqrt[2/3]}}
```

Nota

Il colore delle parentesi non ancora chiuse, definite in *Mathematica* come *Unmatched brackets*, è personalizzabile così come si può impostare di non visualizzare affatto tale colore. Queste personalizzazioni rientrano nelle caratteristiche del front end.

Alcuni principi fondamentali

4. Qualsiasi cosa è un'espressione

Un altro principio fondamentale riportato in molti testi della letteratura di *Mathematica* viene spesso sintetizzato con la frase: "Qualsiasi cosa è un'espressione".

Esasperando il concetto si potrebbe affermare che *Mathematica* è interamente costruito sul semplice concetto di espressione.

Tutto ciò che si utilizza e si manipola in *Mathematica* è riconducibile ad una espressione del tipo

```
(1) Oggetto[qualcosa definito dall'utente, parametri richiesti, parametri opzionali]
```

E questo è vero non solo a livello di kernel ma anche a livello di front end. Ad esempio la cella di input

```
In[1]:= 2 + 2
```

ha un'espressione interna del tipo

```
(2) Cell[ BoxData[ RowBox[{ "2", "+", "2" }], "Input", CellLabel->"In[1]:=" ]
```

con la seguente interpretazione

```
Cell [BoxData[RowBox[{"2", "+", "2"}]], "Input", CellLabel -> "In[1]:="]
Oggetto          Qualcosa definito dall'utente      Parametri richiesti      Parametri opzionali
```

Così come un notebook ha un'espressione interna del tipo

```
Notebook[{
  Cell[
    BoxData[ \ (2 + 2)\ ], "Input"
  ],
  FrontEndVersion->"4.1 for Microsoft Windows",
  ScreenRectangle->{{0, 1024}, {0, 695}},
  WindowSize->{496, 599},
  WindowMargins->{{18, Automatic}, {Automatic, 23}}
]}
```

Quindi risulta indispensabile acquisire il concetto di espressione al fine di poter meglio comprendere poi il comportamento del sistema. Associato al concetto intrinseco di espressione vi è un altro concetto di carattere generale molto importante, che non va mai trascurato quando si lavora con *Mathematica*: Qualsiasi oggetto mostrato all'utente dal front end è una rappresentazione sintetica e standardizzata della reale espressione che *Mathematica* conserva internamente per lo stesso oggetto. Ogni oggetto reca con se una serie di informazioni aggiuntive necessarie al sistema per la corretta interpretazione dei dati e delle eventuali operazioni che l'utente richiede su di essi. Ad esempio in molti casi all'utente non è evidente la forma (1) sopra riportata, perché *Mathematica* interpreta tale forma e riconosce che ne esiste una rappresentazione equivalente di più immediata

comprensione per l'utente. Banalmente basti pensare all'espressione della cella (2), che ovviamente a video viene mostrata come una semplice cella di input con la richiesta di somma $2+2$.

Alcuni principi fondamentali

5. Le opzioni

Il meccanismo delle opzioni interviene in due momenti diversi: quando si utilizzano le funzioni di carattere prettamente computazionale, per indicare opportune informazioni aggiuntive all'algoritmo che deve risolvere un problema di calcolo; e, come nel caso della grafica e della programmazione del front end, quando si desidera personalizzare il modo in cui appare un oggetto. Ovviamente, vi sono molte altre occasioni in cui l'utilizzo delle opzioni assume significato e ruolo diverso, come per le istruzioni del set di programmazione funzionale o rule-based, ma per il momento questa generalizzazione ben si presta all'introduzione del concetto di opzioni.

Si riportano di seguito alcuni esempi di entrambi i casi sopra citati, rimandando ai capitoli successivi gli approfondimenti sulla struttura e sul significato di alcune opzioni, con particolare attenzione al caso della grafica.

Quando si chiede un integrale definito (*Integrate*), *Mathematica* cerca di restituire un valore sempre valido, se è necessario indicando le condizioni di validità.

```
In[22]:=  $\int_0^{\infty} \text{Exp}[a x] dx$ 
Out[22]:= If[Re[a] < 0, -1/a, Integrate[e^{ax}, {x, 0, \infty}, Assumptions -> Re[a] >= 0]]
```

Se il risultato ottenuto non è proprio come ci si aspettava, si può provare a visualizzare le opzioni della funzione utilizzata, per tentare di modificare il comportamento della *Integrate*.

```
In[23]:= Options[Integrate]
Out[23]:= {Assumptions -> $Assumptions, GenerateConditions -> Automatic, PrincipalValue -> False}
```

Se si vuole conoscere l'uso ed il significato di una particolare opzione, basta utilizzare uno degli help disponibili.

```
In[24]:= ?? Assumptions
Assumptions is an option for functions such as Simplify, Refine and Integrate which
specifies default assumptions to be made about symbolic quantities. More...
Attributes[Assumptions] = {Protected}
```

Se è la prima volta che si utilizza un'opzione, è preferibile, comunque, visualizzare il manuale in linea Built-inFunction, che offre spiegazioni dettagliate sia sul significato e comportamento delle opzioni sia sui valori ammissibili che si possono utilizzare. Il significato delle opzioni di *Integrate* viene riepilogato in tabella

Nome	Valore di Default	Descrizione
Assumptions	{}	Condizioni sugli eventuali parametri della funzione integranda
GenerateConditions	Automatic	Generare le condizioni sui parametri nell'integrale definito
PrincipalValue	False	Individuare il valore principale di Cauchy nell'integrale definito

Tabella 0.5.6. Descrizione delle opzioni della funzione Integrate.

Dunque, l'opzione Assumptions permette di indicare apriori le condizioni, se note, sui parametri.

```
In[25]:= Integrate[Exp[a x], {x, 0, ∞}, Assumptions -> {a < 0}]
Out[25]= -1/a
```

In pratica le Options sono delle condizioni interne degli algoritmi associati alle varie funzioni, e pertanto devono sempre avere un valore definito. Questo conduce facilmente al concetto del valore di default per ciascuna Options, che viene impostato dal sistema ed utilizzato dal kernel nel caso in cui l'utente omette particolare indicazioni in merito. Segue ancora un esempio di opzioni nel caso della funzione Limit.

```
In[26]:= Limit[1/(x-1), x -> 1]
Out[26]= ∞
```

```
In[27]:= Options[Limit]
Out[27]= {Analytic -> False, Assumptions -> $Assumptions, Direction -> Automatic}
```

```
In[28]:= ?? Direction
Direction is an option for Limit. Limit[expr, x -> x0, Direction ->
  1] computes the limit as x approaches x0 from smaller values. Limit[
  expr, x -> x0, Direction -> -1] computes the limit as x approaches
  x0 from larger values. Direction -> Automatic uses Direction -> -1
  except for limits at Infinity, where it is equivalent to Direction -> 1.

Attributes[Direction] = {Protected}
```

come si legge dal messaggio di help precedente, se si desidera specificare la direzione con cui la variabile indipendente tende al valore limite, basta utilizzare l'opzione Direction

```
In[29]:= {Limit[1/(x-1), x -> 1, Direction -> 1], Limit[1/(x-1), x -> 1, Direction -> -1]}
Out[29]= {-∞, ∞}
```

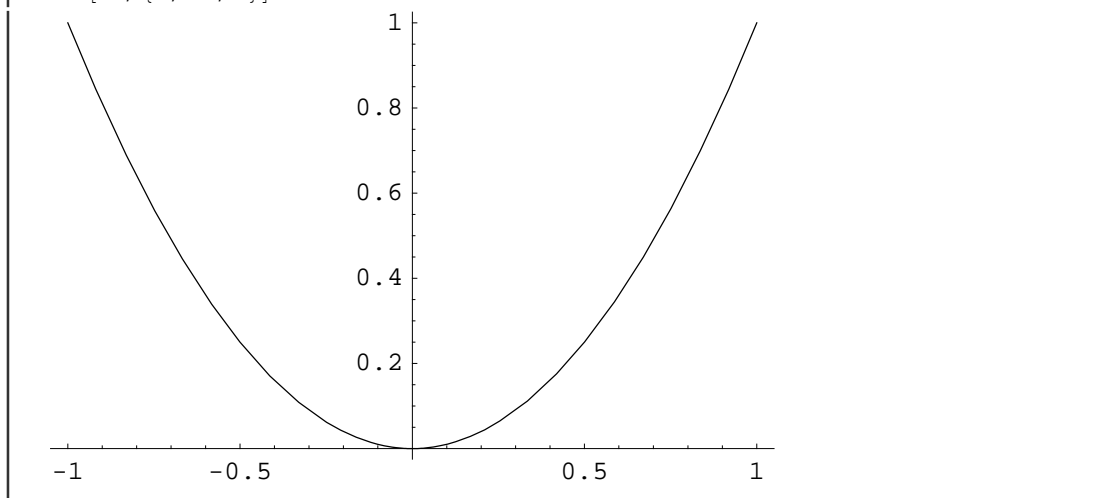
Come detto prima, l'altro caso in cui si può manipolare il comportamento del sistema tramite il meccanismo delle opzioni è quello della grafica o del front end. La tabella che segue mostra alcune opzioni inerenti le principali funzioni di grafica in due e tre dimensioni (d'ora in poi indicata come 2D e 3D).

Nome	Valore di Default	Descrizione
AspectRatio	$\frac{1}{\text{GoldenRatio}}$	Rapporto tra altezza e larghezza del grafico
Axes	True	Disegna o no gli assi
AxeseLabel	None	Mostra eventuali etichette sugli assi
AxesOrigin	Automatic	Indica il punto dove disegnare l'origine degli assi
Background	Automatic	Definisce il colore dello sfondo
Ticks	Automatic	Disegna le tacche in corrispondenza delle unità di misura sugli assi
PlotPoints	25	Numero di punti campione da calcolare per il grafico

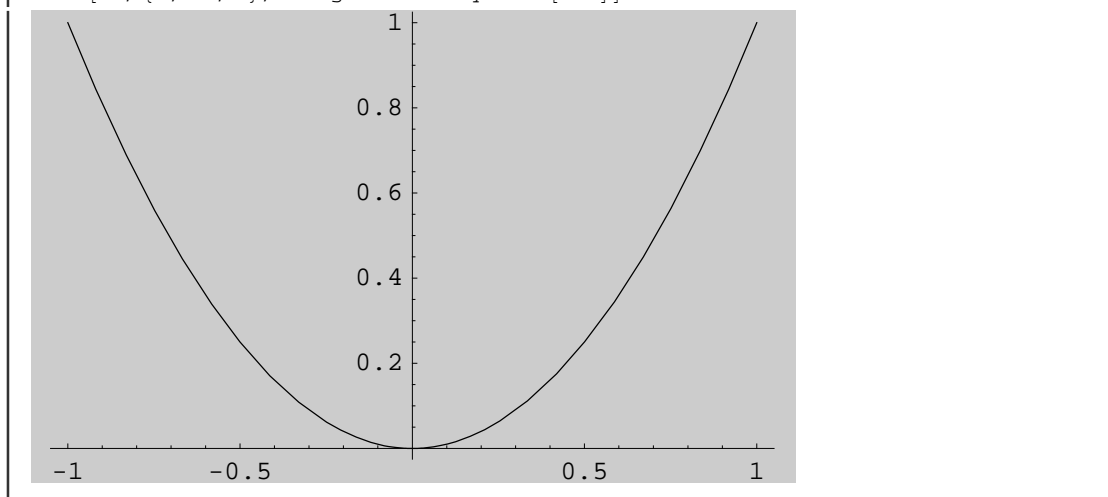
Tabella 0.5.7. Alcuni opzioni delle funzioni di grafica.

Di mostrano ora solo alcuni esempi di modifica del risultato grafico ottenuto con Plot.

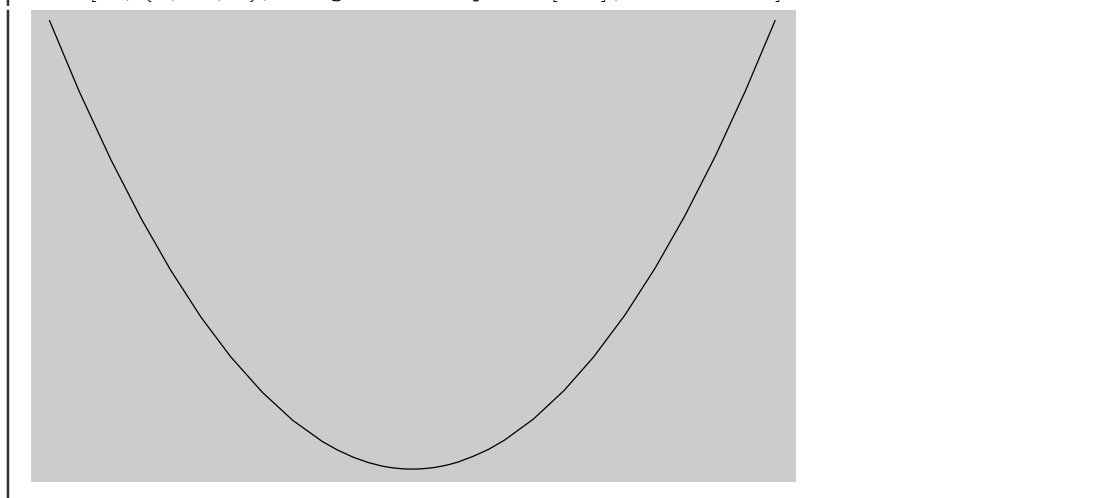
In[30]:= `Plot[x2, {x, -1, 1}];`



In[31]:= `Plot[x2, {x, -1, 1}, Background -> GrayLevel[0.8]];`



In[32]:= `Plot[x2, {x, -1, 1}, Background -> GrayLevel[0.8], Axes -> False];`



Per quanto concerne l'utilizzo delle opzioni per il front end, il discorso è abbastanza articolato, in quanto il meccanismo delle opzioni si unisce al meccanismo dell'ereditarietà dei valori impostati per le opzioni, poiché il front end è strutturato su diversi livelli (global, notebook e celle). In ogni caso si riporta qui un semplice esempio di opzione per modificare l'apparenza del testo in una cella, rimandando anche in questo caso al capitolo che tratta in dettaglio il front end per un maggiore approfondimento.

La cella che segue ha un testo scritto con carattere Comic San Serif di dimensione 14, colore nero più altre opzioni ereditate dal notebook, come ad esempio il colore dello sfondo.

Esempio di opzioni del front end per la modifica del testo

Mathematica rappresenta internamente questa cella con l'espressione

```
Cell["Esempio di opzioni del front end per la modifica del testo","Text"]
```

Se si modifica, tramite i comandi del menù Format il carattere da Comic San Serif a Times New Roman, la dimensione da 14 a 16, il colore da nero a blu e lo sfondo da bianco a grigio, l'espressione interna della cella modificata sarà come la seguente

```
Cell["Esempio di opzioni del front end per la modifica del testo", "Text",
  FontFamily->"Times",
  FontSize->16,
  FontColor->RGBColor[0,0,1],
  Background->GrayLevel[0.9]
]
```

mentre la cella apparirà nel notebook come la seguente

Esempio di opzioni del front end per la modifica del testo

Si valuti la cella seguente per vedere un effetto particolare sul notebook

```
In[15]:= Do[
  SetOptions[EvaluationNotebook[],
    Background->GrayLevel[Cos[2 π x]],
    FontColor->Hue[Sin[2 π x]], {x, 0, 2, .02}
  ];
  SetOptions[EvaluationNotebook[], FontColor->GrayLevel[0]];
```

In conclusione, il meccanismo delle opzioni permette all'utente di modificare il comportamento del sistema sia nell'esecuzione dei calcoli sia nella presentazione a video dei risultati.

6. Gli attributi

Mentre le opzioni di una funzione offrono la possibilità di variare il suo comportamento in funzione del valore ad esse assegnato, gli Attributes di una funzione indicano alcune sue caratteristiche preimpostate, che possono al più essere inibite, ma non modificate nella loro influenza sul comportamento della funzione stessa.

La tabella riporta un breve elenco di alcuni attributi che si possono incontrare associati alle funzioni di *Mathematica*

Nome	Descrizione
Constant	Tutte le derivate di f sono nulle, f è una costante
HoldAll	Gli argomenti di f non sono valutati quando f viene eseguita
Listable	f viene automaticamente applicata a tutti gli elementi di una lista
Locked	Gli attributi di f non possono essere modificati
Orderless	f è commutativa (implica che viene eseguito l'ordine canonico dei suoi argomenti)
Protected	I valori assegnati ad f non possono essere modificati
ReadProtected	Le assegnazioni di f sono protette dalla lettura

Tabella 0.6.8. Alcuni attributi comuni.

Seguono degli esempi che chiariscono il senso di alcuni attributi sopra riportati; gli altri attributi verranno trattati con maggiore dettaglio in seguito, quando si affronteranno funzioni che assumo un comportamento specifico proprio in funzione dei rispettivi attributi.

Se si deve eseguire la derivata di un'espressione con variabili e parametri, potrebbe essere utile indicare che i parametri vanno considerati come costanti, ossia a derivata nulla.

```
In[33]:= Dt[3 x^3 + c x^2 - 8 c x, x]
Out[33]= -8 c + 2 c x + 9 x^2 - 8 x Dt[c, x] + x^2 Dt[c, x]
```

A meno di opportune specifiche sul parametro c , *Mathematica* indica il risultato della precedente derivata in funzione della derivata di c rispetto ad x , essendo c non definito. Se deve essere inteso come costante, basta impostare l'attributo Constant per il simbolo c .

```
In[34]:= SetAttributes[c, Constant]
```

```
In[35]:= Dt[3 x^3 + c x^2 - 8 c x, x]
Out[35]= -8 c + 2 c x + 9 x^2
```

L'attributo Protected, assegnato di default a tutti gli oggetti nativi di *Mathematica*, serve per prevenire eventuali ri-assegnazioni di simboli o funzioni protette. Ad esempio potrebbe capitare di voler utilizzare le variabili D o I , che sono rispettivamente il simbolo di derivata parziale e di unità immaginaria

```
In[36]:= D = 0
Set::wrsym : Symbol D is Protected. More...
Out[36]= 0
```

```
In[37]:= I = 0
Set::wrsym : Symbol i is Protected. More...
Out[37]= 0
```

Come richiamato dai messaggi di errore, tali simboli hanno l'attributo Protected, che evita appunto eventuali assegnazioni dell'utente che potrebbero causare malfunzionamenti al sistema. Ovviamente anche per le proprie routine o variabili si può impostare tale l'attributo così da evitare errori involontari da parte di chi poi le utilizza.

```
In[38]:= Attributes[D]
Out[38]= {Protected, ReadProtected}
```

```
In[39]:= a = 10
Out[39]= 10
```

```
In[40]:= SetAttributes[a, Protected]
```

```
In[41]:= a = 11
Set::wrsym : Symbol a is Protected. More...
Out[41]= 11
```

```
In[42]:= Clear[a]
Clear::wrsym : Symbol a is Protected. More...
```

```
In[43]:= ?? a
Global`a

Attributes[a] = {Protected}

a = 10
```

```
In[44]:= Remove[a]
Remove::rmptc : Symbol a is Protected and cannot be removed. More...
```

Come si vede un simbolo protetto non può essere modificato, cancellato o rimosso.

L'attributo Protected offre la possibilità di rimuovere la protezione di un oggetto tramite la funzione Unprotect

```
In[45]:= Unprotect[a]
Out[45]= {a}
```

```
In[46]:= a = 0
Out[46]= 0
```

```
In[47]:= ? a
Global`a

a = 0
```

```
In[48]:= Remove[a]
```

```
In[49]:= ? a
Information::notfound : Symbol a not found. More...
```

La possibilità di assegnare un'espressione ad un'altra espressione è davvero una notevole potenzialità del sistema, ma può portare ad alcuni problemi. Si supponga di voler assegnare ad una somma di variabili un determinato valore, come nell'esempio che segue

```
In[50]:= a + b = c
Set::write : Tag Plus in a + b is Protected. More...
Out[50]= c
```

Il messaggio dice che la funzione Plus è protetta, essendo una funzione di sistema. E' facile scoprire perché l'assegnazione fatta si riconduce alla funzione Plus.

```
In[51]:= FullForm[a + b]
Out[51]//FullForm= Plus[a, b]
```

Dunque l'espressione $a + b = c$ viene intesa come

```
In[52]:= Plus[a, b] = c
Set::write : Tag Plus in a + b is Protected. More...
Out[52]= c
```

il che mostra come si è tentato di assegnare il valore c alla funzione Plus, il che non è permesso a causa dell'attributo Protected. Come si verifica di seguito, l'assegnazione non risulta associata con le variabili a o b e nemmeno con la loro somma.

```
In[53]:= ?? a
Global`a
```

```
In[54]:= ?? b
Global`b
```

```
In[55]:= ?? a+b
Information::notfound : Symbol a+b not found. More...
```

Se si desidera, è possibile eseguire tale assegnazione rimuovendo la protezione alla funzione Plus.

```
In[56]:= Unprotect[Plus]
Out[56]= {Plus}
```

```
In[57]:= a + b = c
Out[57]= c
```

```
In[58]:= Protect[Plus]
Out[58]= {Plus}
```

```
In[59]:= a + f + t + c + g + b
Out[59]= 2 c + f + g + t
```

```
In[60]:= a + b
Out[60]= c
```

Come si osserva ora la regola di sostituzione è stata registrata. Ma dove ?

```
In[61]:= ?? a
Global`a
```

```
In[62]:= ?? b
Global`b
```

```
In[63]:= ?? c
Global`c
Attributes[c] = {Constant}
```

Le variabili che intercorrono nella regola (a, b, c) non riportano alcuna informazione, questo implica che vi solo un altro posto dove si è potuto registrare la regola, all'interno della Plus.

```
In[64]:= ?? Plus
x + y + z represents a sum of terms. More...
Attributes[Plus] = {Flat, Listable, NumericFunction, OneIdentity, Orderless, Protected}

a + b = c

Default[Plus] := 0
```

Quindi ora si comincia a comprendere che le regole sono soggette anche al posto in cui vengono memorizzate. In particolare, nell'esempio appena riportato il posto è davvero significativo, perchè avendo aggiunto una regola al comando Plus, ogni qualvolta il *Mathematica* dovrà effettuare delle somme andrà ad eseguire il controllo sulle sue regole interne e su quelle aggiunte dall'utente. Questo, vista la frequenza delle somme che si hanno in qualunque algoritmo, comporta un appesantimento dei tempi di esecuzione.

Mathematica permette all'utente anche di gestire il posto in cui memorizzare le regole.

Probabilmente, se si ha l'esigenza di ricordare la regola $a + b = c$, si può presumere che tale regola debba essere adottata solo quando intervengono a o b in un certo calcolo, ed è quindi inutile chiedere al sistema di verificarne l'occorrenza per qualsiasi somma.

La soluzione è di caricare la regola a bordo delle variabili che ne sono interessate, ossia a e b . Per fare questo si sfrutta la possibilità di assegnare alle variabili un valore aggiuntivo a quello normalmente attribuito con il semplice $=$.

Per ciascuna variabile esistono due tipi di valori che possono essere assegnati, chiamati UpValue e DownValue.

L'UpValue è il valore normalmente assegnato con l'operatore $=$

Il DownValue è un valore assegnato alla variabile mediante l'operatore $\wedge=$

Si prova ora a registrare la regola di prima su $a + b$.

Innanzitutto eliminiamo la precedente regola dalla Plus per essere certi che tutto funzioni.

```
In[65]:= Unprotect[Plus]
Out[65]:= {Plus}
```

```
In[66]:= a + b = .
```

```
In[67]:= Protect[Plus]
Out[67]:= {Plus}
```

```
In[68]:= a + b
Out[68]:= a + b
```

```
In[69]:= ?? Plus
x + y + z represents a sum of terms. More...

Attributes[Plus] = {Flat, Listable, NumericFunction, OneIdentity, Orderless, Protected}

Default[Plus] := 0
```

Ora si ridefinisce la stessa regola, questa volta memorizzandola su a e b

```
In[70]:= a + b ^= c
Out[70]:= c
```

```
In[71]:= a + b
Out[71]= c
```

```
In[72]:= a + f + t + c + g + b
Out[72]= 2 c + f + g + t
```

```
In[73]:= ?? Plus
x + y + z represents a sum of terms. More...

Attributes[Plus] = {Flat, Listable, NumericFunction, OneIdentity, Orderless, Protected}

Default[Plus] := 0
```

```
In[74]:= ?? a
Global`a

a /: a + b = c
```

```
In[75]:= ?? b
Global`b

b /: a + b = c
```

```
In[76]:= {UpValues[b], UpValues[b]}
Out[76]= {{HoldPattern[a + b] => c}, {HoldPattern[a + b] => c}}
```

Si noti che il fatto che a e b abbiano degli UpValues non implica che esse non possano avere dei DownValues

```
In[77]:= a = 10
Out[77]= 10
```

```
In[78]:= a + b
Out[78]= 10 + b
```

```
In[79]:= a = .
```

```
In[80]:= a + b
Out[80]= c
```

```
In[81]:= a = 1
Out[81]= 1
```

```
In[82]:= b = 9
Out[82]= 9
```

```
In[83]:= a + b
Out[83]= 10
```

```
In[84]:= ?? a
Global`a

a = 1

a /: a + b = c
```

```
In[85]:= ?? b
Global`b

b = 9

b /: a + b = c
```

```
In[86]:= Trace[a + b]
Out[86]= {{a, 1}, {b, 9}, 1 + 9, 10}
```

In questo caso eventuali valori di c non influenzano il risultato perché la regola non viene proprio applicata

```
In[87]:= c = 11
Out[87]= 11
```

```
In[88]:= a + b
Out[88]= 10
```

```
In[89]:= a = .
```

```
In[90]:= Trace[a + b]
Out[90]= {{b, 9}, a + 9, 9 + a}
```

```
In[91]:= a = .
```

```
In[92]:= b = .
```

```
In[93]:= c = 10
Out[93]= 10
```

```
In[94]:= a + b
Out[94]= 10
```

Dagli ultimi esempi si deduce che se una delle due variabili a o b assume valore diverso dal simbolo stesso, la regola non viene più applicata, in quanto non ricorre più l'espressione $a + b$. Infatti *Mathematica* valuta sempre prima la singola variabile a , poi la singola variabile b ed infine controlla se rispettano la regola assegnata.

Alcuni principi fondamentali

7. I pacchetti aggiuntivi (Standard Add On Packages)

La vasta gamma di funzioni native messe a disposizione di *Mathematica* è ampliata da una serie di pacchetti aggiuntivi forniti a corredo del sistema e chiamati Standard Add On Packages. Tali funzioni non sono state inglobate nel kernel per diverse ragioni. Da una parte esse rappresentano, nella maggior parte dei casi, implementazioni di funzioni specifiche per certi ambiti scientifici di interesse non generale (ad esempio funzioni statistiche quali le distribuzioni continue e discrete, gli intervalli di confidenza, il test delle ipotesi, la statistica descrittiva, così come funzioni tipiche della Teoria dei Numeri, ecc.). In altri casi perché si tratta di funzioni implementate con il linguaggio stesso di *Mathematica* e dunque non vengono compilate all'interno del kernel. In ogni caso il criterio adottato è quello di non appesantire la sessione di lavoro con numerose definizioni di funzioni che probabilmente non vengono utilizzate da una gran parte di utenti. Ovviamente, qualora si ha la necessità di eseguire calcoli specifici per cui esistono delle funzioni nei pacchetti standard aggiuntivi basta semplicemente caricare in memoria il pacchetto corrispondente.

Ciascun pacchetto aggiuntivo ha una dettagliata documentazione che spiega l'utilizzo delle funzioni in esso definite. Per visualizzare tale documentazione basta aprire l'Help Browser alla voce

Add-ons > Standard Packages > The Standard Add-on Packages.

Di seguito si riporta una tabella riepilogativa delle categorie di pacchetti disponibili con la versione 4.2 di *Mathematica*.

Categoria	Breve descrizione
Algebra	Estende le potenzialità di Mathematica nella manipolazione di espressioni algebriche e polinomiali. In questa categoria vi sono funzioni per la risoluzione di disequazioni polinomiali e per la rappresentazione di polinomi in termini di residui e polinomi simmetrici. Inoltre, vengono definite le funzioni <code>PolynomialExtendedGCD</code> e <code>PolynomialPowerMod</code> , che vanno ad aggiungersi alle funzioni native <code>PolynomialGCDePolynomialMod</code> .
Calculus	Mathematica dispone di molteplici funzioni native per il calcolo, quali ad esempio il calcolo degli integrali, derivate, soluzioni di equazioni differenziali e limiti. Questa categoria di pacchetti estende tali funzionalità, fornendo funzioni per il calcolo degli integrali completi di equazioni differenziali, il calcolo dell'approssimazione di Padé e le operazioni vettoriali in differenti sistemi di coordinate tridimensionali.
DiscreteMath	La Matematica Discreta fa riferimento alle strutture matematiche enumerabili, come ad esempio studiate nell'analisi combinatoria, nella teoria dei grafi e nella geometria computazionale. Questa categoria include, tra gli altri, il pacchetto <code>Combinatorica</code> , che fornisce oltre 200 funzioni per l'analisi combinatoria e la teoria dei grafi; <code>ComputationalGeometry</code> , che fornisce molte importanti funzioni geometriche utili nell'analisi dei dati non parametrici.
Graphics and Geometry	Questa categoria offre molti pacchetti che incrementano le potenzialità di rappresentazione grafica dei dati di Mathematica. Include, tra gli altri, pacchetti per grafici su scala logaritmica, in coordinate polari, campi vettoriali, superfici di rivoluzione, curve di livello in tre dimensioni, funzioni definite implicitamente, funzioni complesse di variabili complesse. Altri pacchetti forniscono, ancora, potenzialità per la realizzazione di grafici a barre o a torte, inserimento di legende all'interno di grafici, frecce o incrementano le direttive di colore per la manipolazione degli stili dei grafici. Infine, il pacchetto <code>Geometry</code> fornisce funzioni che restituiscono la caratteristica dei polinomi regolari, dei poliedri e le rotazioni in due e tre dimensioni.
Linear Algebra	La categoria <code>Linear Algebra</code> fornisce funzioni per la generazione di vettori ortonormali, la risoluzione di sistemi con matrice dei coefficienti tridiagonale, la computazione della fattorizzazione LU e la decomposizione di Cholesky, oltre ad altre potenzialità per la manipolazione di matrici.

Categoria	Breve descrizione
Miscellaneous	Comprende diversi pacchetti che includono funzioni non inserite nelle altre categorie perché non facilmente classificabili, ma di utilità generale. Ad esempio vi sono pacchetti per esplorare le potenzialità di Mathematica su argomenti quali il suono tramite la modulazione di onda o le scale musicali. Altre funzioni forniscono, ad esempio, la creazione di calendari, la tabella degli elementi chimici, delle costanti fisiche, le unità di conversione, le misure geodetiche ed le rappresentazioni grafiche di cartine geografiche. Infine, il pacchetto <code>RealOnly</code> è utile per gli studenti di algebra che non desiderano considerare il campo dei complessi nelle soluzioni fornite da Mathematica.
Number Theory	Importanti funzioni per la teoria dei numeri sono già presenti in Mathematica, come ad esempio <code>PrimePi</code> , <code>EulerPhi</code> , <code>MoebiusMu</code> , <code>DivisorSigma</code> . Questa categoria amplia tali funzionalità fornendo pacchetti per la primalità e per l'esplorazione dei metodi delle curve ellittiche per la fattorizzazione di interi. Vi sono, inoltre, funzioni per l'approssimazione di numeri reali tramite razionali e per l'approssimazione di polinomi con radici reali tramite polinomi a coefficienti interi. Sono anche supportate funzioni quali Ramanujan τ e Siegel Θ .
NumericalMath	Questa categoria estende le potenzialità native di Mathematica per il calcolo numerico, traendo ulteriori vantaggi dalla aritmetica in precisione arbitraria, che è l'elemento fondamentale per le computazioni numeriche con Mathematica. Vi sono pacchetti che forniscono: funzioni di fitting (polinomiali, spline, trigonometriche); versioni numeriche di funzioni simboliche del kernel (<code>ND</code> , <code>NLimit</code> , <code>NResidue</code> , <code>NSeries</code>); funzioni di integrazione numerica <code>CauchyPrincipalValue</code> , <code>ListIntegrate</code> , <code>NIntegrateInterpolationFunction</code> ; supporto per la soluzione numerica di equazioni differenziali (<code>BesselZeros</code> , <code>Butcher</code> , <code>OrderStar</code>); alternative per la <code>FindRoot</code> che utilizzano l'interpolazione o i metodi degli intervalli.
Statistics	Questa categoria offre numerose funzioni utili per molti utenti che utilizzano Mathematica per calcoli statistici. I pacchetti disponibili offrono funzioni per: distribuzioni continue e discrete univariate e multivariate; statistica descrittiva per dati univariati e multivariati; manipolazione di dati e smoothing; test dell'ipotesi e stime degli intervalli di confidenza. Infine, funzioni per la regressione lineare e non lineare permettono agli utenti di trarre vantaggio dai numerosi strumenti di diagnosi.
Utilities	Questa categoria raccoglie una piccola serie di pacchetti di utilità generale.

Tabella 0.10. Le categorie di pacchetti standard aggiuntivi presenti di default con *Mathematica* 4.2.

Una volta individuata una funzione presente in uno dei pacchetti aggiuntivi, per poterla utilizzare è sufficiente caricare il corrispondente pacchetto. Per fare ciò bisogna conoscere il percorso completo del pacchetto, che si compone utilizzando il nome della categorie ed il nome del pacchetto stesso. È bene ricordare che la struttura Categoria \triangleright Pacchetto deve sempre essere rispettata rigorosamente nel caricamento dei pacchetti, in quanto essa corrisponde alla struttura delle cartelle (directory) nelle quali sono memorizzati i file di codice sorgente relativi ai pacchetti.

Si mostra ora un esempio di utilizzo dei pacchetti aggiuntivi. Se si deve risolvere una disequazione polinomiale, consultando l'help si vede che esiste il pacchetto Algebra \triangleright InequalitySolve. Dunque, si deve caricare tale pacchetto. L'istruzione che esegue tale caricamento in memoria si chiama `Get` o, equivalentemente, si può utilizzare il simbolo `<<` (un doppio segno di minore). Per separare il nome della categoria dal nome del pacchetto bisogna utilizzare il carattere ``` (back quote - non presente su alcune tastiere, per inserirlo digitare `[ALT+96]` dal tastierino numerico). Tale carattere deve essere anche incluso anche dopo il nome del pacchetto. Si ricordi di non lasciare mai alcuno spazio tra i nomi delle categorie o dei pacchetti.

```
In[95]:= << Algebra`InequalitySolve`
```

```
In[96]:= ?InequalitySolve
InequalitySolve[expr, x] gives the solution set of an expression containing
logical connectives and univariate polynomial equations and inequalities
in the variable x. InequalitySolve[expr, {x1, ..., xn}] gives the
solution set of an expression containing logical connectives and
linear equations and inequalities in the variables {x1, ..., xn}. More...
```

```
In[97]:= InequalitySolve[x^2 + y^2 - 3 < 0 && x < 2 y - 1, {x, y}]
Out[97]:= (-sqrt(3) < x <= 1/5 (-1 - 2 sqrt(14)) && -sqrt(3 - x^2) < y < sqrt(3 - x^2) ||
(1/5 (-1 - 2 sqrt(14)) < x < 1/5 (-1 + 2 sqrt(14)) && (1+x)/2 < y < sqrt(3 - x^2))
```

```
In[98]:= InequalitySolve[x^2 + y^2 - 3 < 0 && x < 2 y - 1, {x, y}] // TraditionalForm
Out[98]//TraditionalForm=
```

$$\left(-\sqrt{3} < x \leq \frac{1}{5}(-1 - 2\sqrt{14}) \wedge -\sqrt{3-x^2} < y < \sqrt{3-x^2} \right) \vee$$

$$\left(\frac{1}{5}(-1 - 2\sqrt{14}) < x < \frac{1}{5}(-1 + 2\sqrt{14}) \wedge \frac{x+1}{2} < y < \sqrt{3-x^2} \right)$$

Un modo semplice per verificare quali pacchetti sono stati già caricati nella sessione corrente è fare riferimento alla variabile di ambiente `$ContextPath`, citata nel paragrafo precedente.

```
In[99]:= $ContextPath
Out[99]:= {Algebra`InequalitySolve`, Global`, System`}
```

In alcuni casi potrebbe risultare utile caricare un'intera categoria di pacchetti, ad esempio nel caso della categoria `Statistics`, dove sono presenti numerosi pacchetti spesso tutti utili per una serie di computazioni. Per fare ciò basta invocare il comando `<<` con il solo nome della categoria seguito dal carattere ```, come nell'esempio seguente:

```
In[100]:= << Statistics`
```

```
In[101]:= $ContextPath
Out[101]:= {Statistics`StatisticsPlots`, Statistics`NormalDistribution`,
Statistics`NonlinearFit`, Statistics`MultinormalDistribution`,
Statistics`MultiDiscreteDistributions`, Statistics`MultiDescriptiveStatistics`,
Statistics`LinearRegression`, Statistics`HypothesisTests`,
Statistics`DiscreteDistributions`, Statistics`DescriptiveStatistics`,
Statistics`DataSmoothing`, Statistics`DataManipulation`,
Statistics`ContinuousDistributions`, Statistics`ConfidenceIntervals`,
Statistics`Common`RegressionCommon`, Statistics`Common`PopulationsCommon`,
Statistics`Common`MultivariateCommon`, Statistics`Common`DistributionsCommon`,
Statistics`ClusterAnalysis`, Utilities`FilterOptions`,
Statistics`ANOVA`, Algebra`InequalitySolve`, Global`, System`}
```

Di seguito si riporta un'altra tabella riepilogativa che elenca tutti i pacchetti di ciascuna categoria.

Categoria	Breve descrizione
Algebra	AlgebraicInequalities, FiniteFields, Horner, InequalitySolve, PolynomialExtendedGCD, PolynomialPowerMod, Quaternions, ReIm, RootIsolation, SymmetricPolynomials
Calculus	DSolveIntegrals, FourierTransform, Integration, Pade, VariationalMethods, VectorAnalysis
DiscreteMath	CombinatorialFunctions, Combinatorica, ComputationalGeometry, DiscreteStep, Permutations, RSolve, Tree
Geometry	Polytopes, Rotations
Graphics	Animation, ArgColors, Arrow, Colors, ComplexMap, ContourPlot3D, FilledPlot, Graphics, Graphics3D, ImplicitPlot, InequalityGraphics, Legend, MultipleListPlot, ParametricPlot3D, PlotField, PlotField3D, Polyhedra, Shapes, Spline, SurfaceOfRevolution, ThreeScript, Common
Linear Algebra	Cholesky, FourierTrig, GaussianElimination, MatrixManipulation, Orthogonalization, Tridiagonal
Miscellaneous	Audio, BlackBodyRadiation, Calendar, ChemicalElements, CityData, Geodesy, Music, PhysicalConstants, RealOnly, ResonanceAbsorptionLines, StandardAtmosphere, Units, WorldData, WorldNames, WorldPlot
Number Theory	ContinuedFractions, FactorIntegerECM, NumberTheoryFunctions, PrimeQ, PrimitiveElement, Ramanujan, Rationalize, Recognize, SiegelTheta
NumericalMath	Approximations, BesselZeros, Butcher, CauchyPrincipalValue, ComputerArithmetic, GaussianQuadrature, InterpolateRoot, IntervalRoots, ListIntegrate, Microscope, NIntegrateInterpolatingFunct, NLimit, NMinimize, NResidue, NSeries, NewtonCotes, PolynomialFit, SplineFit, TrigFit
Statistics	ANOVA, ConfidenceIntervals, ContinuousDistributions, DataManipulation, DataSmoothing, DescriptiveStatistics, DiscreteDistributions, HypothesisTests, LinearRegression, MultiDescriptiveStatistics, MultiDiscreteDistributions, MultinormalDistribution, NonlinearFit, NormalDistribution, Common
Utilities	BinaryFiles, FilterOptions, MemoryConserve, Package, ShowTime

Tabella 0.11. Elenco dei pacchetti disponibili per ciascuna categoria di Standard Add Ons Packages.

Alcuni principi fondamentali

8. I macro solutori

Una delle caratteristiche di Mathematica che lo hanno reso utile ed indispensabile in molti contesti scientifici è la disponibilità di un ampio insieme di macro solutori, ossia funzioni in grado di risolvere problemi complessi, come il calcolo analitico di un integrale, un limite o un sistema di equazioni differenziali, con una sola istruzione. La tabella che segue riporta i principali macro solutori.

Nome	Descrizione
Solve	Soluzione di equazioni o sistemi
NSolve	Soluzione di equazioni o sistemi con metodi numerici
DSolve	Soluzione con metodi analitici di equazioni differenziali o sistemi
NDSolve	Soluzione con metodi numerici di equazioni differenziali o sistemi
Integrate	Integrazione con metodi analitici
NIntegrate	Integrazione con metodi numerici
Limit	Calcolo di limiti
Series	Sviluppo in serie di potenze
D	Derivata parziale
Dt	Derivata totale

Tabella 12. Altre funzioni di uso comune in *Mathematica*

La presenza di tali funzioni comporta un grande vantaggio per gli utenti, che non devono preoccuparsi di implementare alcuna routine per risolvere sofisticati problemi. Basta fornire alla funzione i valori necessari, quali, ad esempio, l'espressione analitica di una funzione ed il nome della variabile indipendente, per calcolare un integrale in maniera analitica. Dietro ciascuna di tali funzioni si trova un algoritmo robusto, ottimizzato e testato ormai da molti anni, che in alcuni casi si articola in centinaia o migliaia di pagine di codice sorgente compilato nel kernel. Mathematica 5.0 riporta, tra le altre novità, un notevole aggiornamento di alcune di tali funzioni, quali ad esempio la `DSolve` e la `NDSolve`, per le quali gli algoritmi sottostanti sono stati completamente riscritti secondo nuovi paradigmi che ora permettono un calcolo più rapido, più affidabile ed in grado di gestire numerose nuove funzioni matematiche speciali.

Oltre alla potenzialità di calcolo, tali macro solutori offrono un ulteriore aiuto. Operando nel caso simbolico, essi permettono anche di esplorare le regole generali che si applicano ai vari calcoli. Ad esempio, nel caso dell'operatore di derivata `D`, la sua caratteristica di operare anche in maniera simbolica, permette di rivedere le regole generali per il calcolo delle derivate. Chiamando la funzione `D` con una qualsiasi espressione e/o composizione di funzioni, senza peraltro specificare l'espressione analitica delle funzioni stesse, si ottiene la forma generale della derivata.

```
In[102]:= D[f[x] + g[x], x] (* Derivata della somma di due funzioni *)
Out[102]= f'[x] + g'[x]
```

```
In[103]:= D[f[x] g[x], x] (* Derivata del prodotto di due funzioni *)
Out[103]= g[x] f'[x] + f[x] g'[x]
```

In[104]:= $D\left[\frac{f[x]}{g[x]}, x\right]$ (* Derivata del rapporto di due funzioni *)
 Out[104]:= $\frac{f'[x]g[x] - f[x]g'[x]}{g[x]^2}$

In questo ultimo caso la forma data in output potrebbe essere diversa da quella usualmente nota, dunque si può modificarla per ricondurla a quella tipica, tramite l'uso del comando `Together` che riscrive un'espressione algebrica di più razionali in un'unica espressione con denominatore comune.

In[105]:= `Together[%]`
 Out[105]:= $\frac{g[x]f'[x] - f[x]g'[x]}{g[x]^2}$

In[106]:= $D[f[g[x]], x]$ (* Derivata di funzioni composte *)
 Out[106]:= $f'[g[x]]g'[x]$

In[107]:= $D[f[x]g[x], x]$
 Out[107]:= $f[x]g'[x] \left(\frac{g[x]f'[x]}{f[x]} + \text{Log}[f[x]]g'[x] \right)$

Per calcolare le derivate successive alla prima, basta indicare la variabile indipendente in una lista, il cui secondo argomento è l'ordine di derivazione

In[108]:= $D[f[x]g[x], \{x, 2\}]$
 Out[108]:= $f[x]g'[x] \left(\frac{g[x]f'[x]}{f[x]} + \text{Log}[f[x]]g'[x] \right)^2 +$
 $f[x]g'[x] \left(-\frac{g[x]f'[x]^2}{f[x]^2} + \frac{2f'[x]g'[x]}{f[x]} + \frac{g[x]f''[x]}{f[x]} + \text{Log}[f[x]]g''[x] \right)$

In[109]:= $D[x^2 \text{Log}[x], \{x, 3\}]$
 Out[109]:= $\frac{2}{x}$

In[110]:= $Dt[x y + \text{Log}[x] \text{Log}[y], x]$
 Out[110]:= $y + x Dt[y, x] + \frac{Dt[y, x] \text{Log}[x]}{y} + \frac{\text{Log}[y]}{x}$

In[111]:= $Dt[x y + \text{Log}[x] \text{Log}[y], \{x, 2\}]$
 Out[111]:= $2 Dt[y, x] + \frac{2 Dt[y, x]}{x y} + x Dt[y, \{x, 2\}] + \left(-\frac{Dt[y, x]^2}{y^2} + \frac{Dt[y, \{x, 2\}]}{y} \right) \text{Log}[x] - \frac{\text{Log}[y]}{x^2}$

Alle volte può essere utile rivedere il risultato con la notazione scientifica tradizionale. Per fare ciò basta aggiungere, dopo il comando, una specifica di visualizzazione di tipo `TraditionalForm`

```
In[112]:= Dt[x y + Log[x] Log[y], {x, 2}] // TraditionalForm
Out[112]//TraditionalForm=
```

$$\frac{2 \frac{dy}{dx}}{x y} + 2 \frac{dy}{dx} + x \frac{d^2 y}{dx^2} + \left(\frac{d^2 y}{dx^2} - \frac{\left(\frac{dy}{dx}\right)^2}{y^2} \right) \log(x) - \frac{\log(y)}{x^2}$$

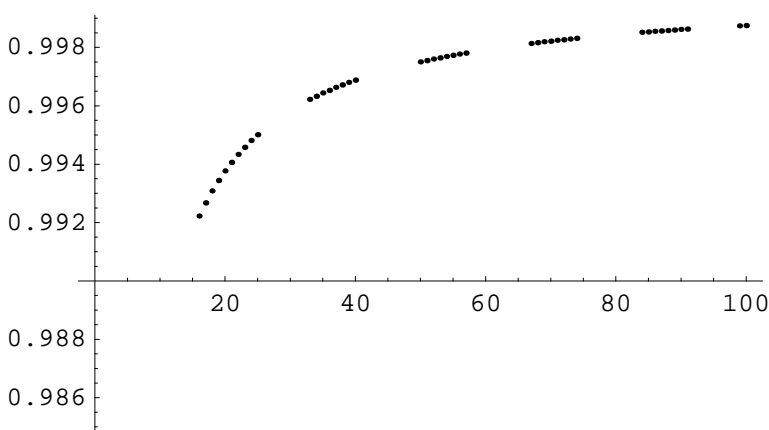
Ora seguono due esempi che illustrano situazioni abbastanza frequenti in Mathematica, ossia casi in cui i macro solutori non sono in grado, apparentemente, di fornire una risposta a determinati input. Il concetto sottostante è che molte volte prima di usare una funzione c'è bisogno di manipolare i dati di partenza in maniera tale da riscriverli in una forma non necessariamente più semplice ma sicuramente più adatta e riconoscibile dagli algoritmi interni ai macro solutori.

Il primo esempio mostra il caso di un limite di una funzione che comprende un coefficiente binomiale nella sua espressione analitica.

Sia data la funzione $f(n) = (-1)^n \binom{-\frac{1}{2}}{n} \sqrt{n\pi}$. Andando a calcolare una sequenza di punti della $f(n)$ e visualizzandoli con la funzione `ListPlot`, si osserva graficamente che essa tende ad 1 per n che tende all'infinito.

```
In[113]:= punti = Table[(-1)^n Binomial[-1/2, n] Sqrt[n Pi], {n, 100}];
```

```
In[114]:= ListPlot[punti];
```



Se si tenta di calcolare tale limite direttamente con la funzione `Limit`, si osserva che `Limit` non è in grado di fornire una risposta.

```
In[115]:= Limit[(-1)^n Binomial[-1/2, n] Sqrt[n Pi], n -> Infinity]
```

```
Out[115]:= Limit[(-1)^n Sqrt[n] Sqrt[Pi] Binomial[-1/2, n], n -> Infinity]
```

Occorre, dunque, tentare di riscrivere l'espressione della $f(n)$ in maniera differente, applicando qualche regola teorica che permette la semplificazione della sua espressione. Un primo passo è quello di utilizzare la funzione `FunctionExpand` che modifica l'espressione fornita, mediante l'applicazione di regole e funzioni speciali. In questo caso, `FunctionExpand` tenta di sostituire il coefficiente binomiale con una combinazione di funzioni Gamma di Eulero $\Gamma(n)$.

```
In[116]:= f1 = FunctionExpand[(-1)^n (n choose -1/2) sqrt(n) pi]
Out[116]= (-1)^n pi / (sqrt(n) Gamma[1/2 - n] Gamma[n])
```

Il passo successivo sarà quello di semplificare ulteriormente tale risultato applicando la formula

$$\Gamma(n) \Gamma(1-n) = \pi \csc(\pi n)$$

Per applicare tale regola, si utilizza la funzione `ReplaceAll`. In particolare, per sostituire il termine $\Gamma(\frac{1}{2}-n)$ si moltiplica e divide per $\Gamma(1 - (\frac{1}{2}-n))$. In tal modo comparirà il termine $\Gamma(n) \Gamma(1-n)$ che potrà essere semplificato mediante l'uso della funzione `FullSimplify`.

```
In[117]:= f2 = ReplaceAll[f1, Gamma[1/2 - n] -> FullSimplify[Gamma[1/2 - n] Gamma[1/2 + n]] / Gamma[1/2 + n]]
Out[117]= (-1)^n Cos[n pi] Gamma[1/2 + n] / (sqrt(n) Gamma[n])
```

Ora si semplifica ulteriormente tale risultato, mediante la funzione `Simplify` con la specifica che $n \in \mathbb{Z}$.

```
In[118]:= f3 = Simplify[f2, n ∈ ℤ]
Out[118]= Gamma[1/2 + n] / (sqrt(n) Gamma[n])
```

A questo punto si prova a calcolare il limite per n che tende ad infinito

```
In[119]:= Limit[f3, n -> ∞]
Out[119]= 1
```

Come si vede dal risultato ottenuto, se ci si ferma ad un uso immediato dei macro solutori con i dati disponibili in partenza, si potrebbe erroneamente pensare che Mathematica non sia in grado di manipolare espressioni complesse. Viceversa, andando ad indagare sulle possibili rappresentazioni alternative dei dati disponibili, spesso mediante l'applicazione di regole teoriche o di funzioni di semplificazione disponibili quali, ad esempio, `Simplify`, `FullSimplify`, `FunctionExpand`, eccetera, si ottiene il risultato desiderato.

Il secondo esempio che si riporta di seguito, tratta il caso dell'integrazione di polinomi di Chebyshev.

In *Mathematica* vi sono due funzioni che restituiscono, rispettivamente, l'espressione di un polinomio di Chebyshev di primo e secondo tipo: `ChebyshevT` e `ChebyshevU`.

```
In[120]:= ChebyshevT[12, x]
Out[120]= 1 - 72 x^2 + 840 x^4 - 3584 x^6 + 6912 x^8 - 6144 x^10 + 2048 x^12
```

```
In[121]:= ChebyshevU[12, x]
Out[121]= 1 - 84 x^2 + 1120 x^4 - 5376 x^6 + 11520 x^8 - 11264 x^10 + 4096 x^12
```

In alcuni casi è indispensabile lavorare con la definizione dei polinomi di Chebyshev anche senza specificarne il grado. Ad esempio si desidera operare con la generica definizione `ChebyshevT[n, x]` senza

assegnare alcun valore ad n.

Si prova ora ad eseguire un integrale, noto in teoria, del prodotto di due polinomi di Chebyshev del primo tipo, senza specificarne l'ordine.

$$\begin{aligned} \text{In[122]} &:= \int \text{ChebyshevT}[n, x] \text{ChebyshevT}[k, x] dx \\ \text{Out[122]} &:= \int \text{ChebyshevT}[k, x] \text{ChebyshevT}[n, x] dx \end{aligned}$$

Anche in questo caso, apparentemente, sembrerebbe che il macro solutore Integrate non sia in grado di risolvere tale problema. Si provi ad usare la FunctionExpand per riscrivere il prodotto dei due polinomi in termini di funzioni trigonometriche e successivamente si esegua l'integrale.

$$\begin{aligned} \text{In[123]} &:= \text{FunctionExpand}[\text{ChebyshevT}[n, x] \text{ChebyshevT}[k, x]] \\ \text{Out[123]} &:= \text{Cos}[k \text{ArcCos}[x]] \text{Cos}[n \text{ArcCos}[x]] \end{aligned}$$

$$\begin{aligned} \text{In[124]} &:= \int \text{Cos}[k \text{ArcCos}[x]] \text{Cos}[n \text{ArcCos}[x]] dx \\ \text{Out[124]} &:= \frac{1}{4} \left(\frac{\text{Cos}[(1+k-n) \text{ArcCos}[x]]}{1+k-n} + \frac{\text{Cos}[(1-k+n) \text{ArcCos}[x]]}{1-k+n} - \frac{\text{Cos}[(-1+k+n) \text{ArcCos}[x]]}{-1+k+n} + \frac{\text{Cos}[(1+k+n) \text{ArcCos}[x]]}{1+k+n} \right) \end{aligned}$$

Segue un altro esempio.

$$\begin{aligned} \text{In[125]} &:= \int \text{ChebyshevT}[n, x] \text{ChebyshevU}[k, x] dx \\ \text{Out[125]} &:= \int \text{ChebyshevT}[n, x] \text{ChebyshevU}[k, x] dx \end{aligned}$$

$$\begin{aligned} \text{In[126]} &:= \text{FunctionExpand}[\text{ChebyshevT}[n, x] \text{ChebyshevU}[k, x]] \\ \text{Out[126]} &:= \frac{\text{Cos}[n \text{ArcCos}[x]] \text{Sin}[(1+k) \text{ArcCos}[x]]}{\sqrt{1-x} \sqrt{1+x}} \end{aligned}$$

$$\begin{aligned} \text{In[127]} &:= \int \% dx \\ \text{Out[127]} &:= \frac{\text{Cos}[(1+k-n) \text{ArcCos}[x]]}{2(1+k-n)} + \frac{\text{Cos}[(1+k+n) \text{ArcCos}[x]]}{2(1+k+n)} \end{aligned}$$

Un altro esempio sull'utilizzo dell'opzione Assumptions.

$$\begin{aligned} \text{In[128]} &:= \text{Timing}[\text{Integrate}[\frac{\text{Exp}[-a x] - \text{Exp}[-b x]}{x}, \{x, 0, \text{Infinity}\}, \\ &\quad \text{Assumptions} \rightarrow \text{Re}[a] > 0 \ \&\& \ \text{Re}[b] > 0]] \\ \text{Out[128]} &:= \{10.266 \text{ Second}, \text{If}[\text{b} \in \text{Reals}, \text{Log}[10-a] - \text{Log}[10-b], \\ &\quad \text{Integrate}[\frac{e^{-10x}(-e^{ax} + e^{bx})}{x}, \{x, 0, \infty\}, \text{Assumptions} \rightarrow \text{b} \notin \text{Reals}]]\} \end{aligned}$$

```
In[129]:= Timing[Integrate[ $\frac{\text{Exp}[-a x] - \text{Exp}[-b x]}{x}$ , {x, 0, Infinity}]]
Out[129]:= {6.875 Second,
  If[Re[a] < 0 && Re[b] < 0, Log[-10 + a] + Log[-a] - Log[a] - Log[-10 + b] - Log[-b] + Log[b],
    Integrate[ $\frac{e^{-10 x} (-e^{a x} + e^{b x})}{x}$ , {x, 0, \infty}, Assumptions -> Re[b] ≥ 0 || Re[a] ≥ 0]]}
```

```
In[130]:= Integrate[ $\frac{\text{Exp}[-a x^2] - \text{Exp}[-b x^2]}{x}$ , {x, 0, Infinity}]
Out[130]:= If[Re[a] > 0 && Re[b] > 0,  $\frac{1}{2} (-\text{Log}[a] + \text{Log}[b])$ ,
  Integrate[ $\frac{e^{-a x^2} - e^{-b x^2}}{x}$ , {x, 0, \infty}, Assumptions -> Re[b] ≤ 0 || Re[a] ≤ 0]]
```

```
In[131]:= Integrate[ $\frac{\text{Exp}[-a x^2] - \text{Exp}[-b x^2]}{x}$ , {x, 0, Infinity},
  Assumptions -> Re[a] > 0 && Re[b] > 0]
Out[131]:=  $\frac{1}{2} (-\text{Log}[a] + \text{Log}[b])$ 
```

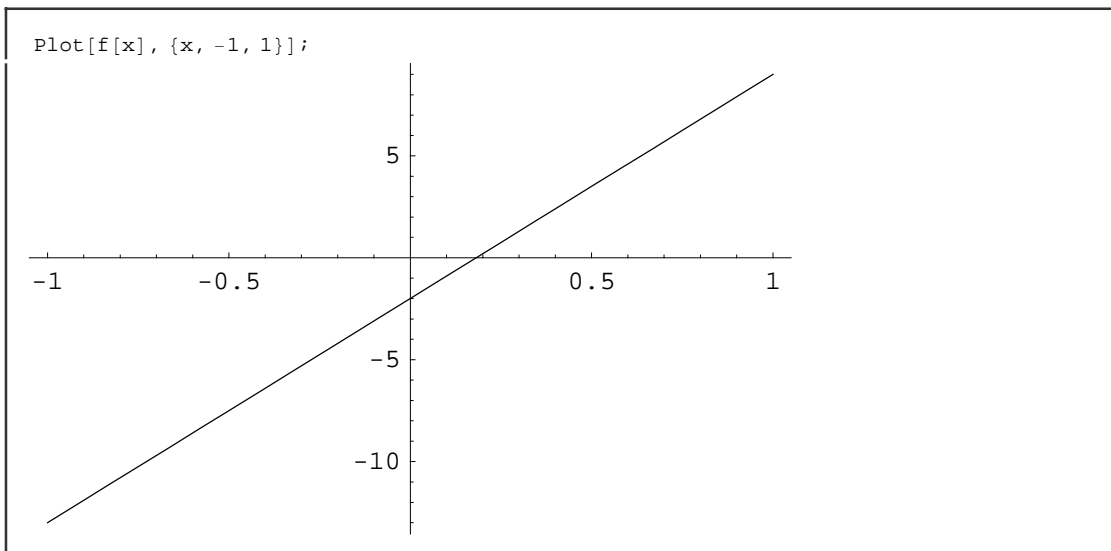
Alcuni principi fondamentali

Gestione degli errori numerici

Propagazione degli errori

```
In[132]:= f[x_] := 11 x - 2
```

```
In[133]:= Plot[f[x], {x, -1, 1}];
```



Questa funzione ha un punto fisso


```
In[140]:= Map[Precision, %]
Out[140]:= {30., 28.9586, 27.9172, 26.8758, 25.8344, 24.793, 23.7516,
22.7103, 21.6689, 20.6275, 19.5861, 18.5447, 17.5033, 16.4619,
15.4205, 14.3791, 13.3377, 12.2963, 11.2549, 10.2135, 9.17215, 8.13075,
7.08936, 6.04797, 5.00658, 3.96518, 2.92379, 1.8824, 0.841005, 0., 0.}
```

Si osserva così che il primo risultato mostra 30 decimali corretti, mentre gli ultimi due risultati sono completamente inaffidabili.

Ovviamente, con *Mathematica* si può anche usare l'aritmetica esatta.

```
In[141]:= NestList[f, 2/10, 30]
Out[141]:= {1/5, 1/5, 1/5, 1/5, 1/5, 1/5, 1/5, 1/5, 1/5, 1/5, 1/5, 1/5, 1/5, 1/5,
1/5, 1/5, 1/5, 1/5, 1/5, 1/5, 1/5, 1/5, 1/5, 1/5, 1/5, 1/5, 1/5, 1/5, 1/5, 1/5}
```

Precisione di lavoro insufficiente

Alcuni algoritmi perdono precisione per problemi malcondizionati al punto tale che risultati accurati non possono essere ottenuti senza l'impiego di una elevata precisione o dell'aritmetica esatta. Questo è un problema classico.

```
In[142]:= hilbert = Table[1/(i+j-1), {i, 10}, {j, 10}]
Out[142]:= {{1, 1/2, 1/3, 1/4, 1/5, 1/6, 1/7, 1/8, 1/9, 1/10}, {1/2, 1/3, 1/4, 1/5, 1/6, 1/7, 1/8, 1/9, 1/10, 1/11},
{1/3, 1/4, 1/5, 1/6, 1/7, 1/8, 1/9, 1/10, 1/11, 1/12}, {1/4, 1/5, 1/6, 1/7, 1/8, 1/9, 1/10, 1/11, 1/12, 1/13},
{1/5, 1/6, 1/7, 1/8, 1/9, 1/10, 1/11, 1/12, 1/13, 1/14},
{1/6, 1/7, 1/8, 1/9, 1/10, 1/11, 1/12, 1/13, 1/14, 1/15},
{1/7, 1/8, 1/9, 1/10, 1/11, 1/12, 1/13, 1/14, 1/15, 1/16},
{1/8, 1/9, 1/10, 1/11, 1/12, 1/13, 1/14, 1/15, 1/16, 1/17},
{1/9, 1/10, 1/11, 1/12, 1/13, 1/14, 1/15, 1/16, 1/17, 1/18},
{1/10, 1/11, 1/12, 1/13, 1/14, 1/15, 1/16, 1/17, 1/18, 1/19}}
```

Se non si utilizza il calcolo simbolico, tale matrice deve essere convertita una versione numerica con precisione macchina

```
In[143]:= approxhilbert = N[hilbert]
Out[143]:= {{1., 0.5, 0.333333, 0.25, 0.2, 0.166667, 0.142857, 0.125, 0.111111, 0.1},
{0.5, 0.333333, 0.25, 0.2, 0.166667, 0.142857, 0.125, 0.111111, 0.1, 0.0909091},
{0.333333, 0.25, 0.2, 0.166667, 0.142857, 0.125, 0.111111, 0.1, 0.0909091, 0.0833333},
{0.25, 0.2, 0.166667, 0.142857, 0.125, 0.111111, 0.1, 0.0909091, 0.0833333, 0.0769231},
{0.2, 0.166667, 0.142857, 0.125, 0.111111, 0.1, 0.0909091, 0.0833333, 0.0769231, 0.0714286},
{0.166667, 0.142857, 0.125, 0.111111, 0.1, 0.0909091, 0.0833333, 0.0769231, 0.0714286, 0.0666667},
{0.142857, 0.125, 0.111111, 0.1, 0.0909091, 0.0833333, 0.0769231, 0.0714286, 0.0666667, 0.0625},
{0.125, 0.111111, 0.1, 0.0909091, 0.0833333, 0.0769231, 0.0714286, 0.0666667, 0.0625, 0.0588235},
{0.111111, 0.1, 0.0909091, 0.0833333, 0.0769231, 0.0714286, 0.0666667, 0.0625, 0.0588235, 0.0555556},
{0.1, 0.0909091, 0.0833333, 0.0769231, 0.0714286, 0.0666667, 0.0625, 0.0588235, 0.0555556, 0.0526316}}
```

La cosa interessante di questo esempio, è il determinante che risulta essere una quantità molto piccola.

```
In[144]:= Det[hilbert]
Out[144]=  $\frac{1}{46206893947914691316295628839036278726983680000000000}$ 
```

```
In[145]:= N[%]
Out[145]=  $2.16418 \times 10^{-53}$ 
```

```
In[146]:= Det[approxhilbert]
Out[146]=  $2.16438 \times 10^{-53}$ 
```

Questo rende la matrice quasi singolare e difficile da invertire. Si confrontino i risultati ottenuti con l'aritmetica esatta e con un sistema numerico.

```
In[147]:= Inverse[hilbert] - Inverse[approxhilbert]
Inverse::luc : Result for Inverse of badly conditioned matrix
  {{1., 0.5, 0.333333, 0.25, 0.2, 0.166667, 0.142857, 0.125, 0.111111, 0.1},
   <<8>>, {0.1, 0.0909091, <<6>>, 0.0555556, 0.0526316}}
 may contain significant numerical errors. More...
Out[147]= {{0.00252887, -0.218769, 4.66361, -42.422,
  202.43, -556.646, 913.48, -882.905, 463.565, -101.952},
  {-0.219226, 18.9576, -404.01, 3674.18, -17529.3, 48194.8, -79079.5, 76424.1,
  -40122.3, 8823.37}, {4.68044, -404.626, 8621.15, -78389.6, 373939.,
  -1.02799  $\times 10^6$ , 1.68659  $\times 10^6$ , -1.62982  $\times 10^6$ , 855584., -188141.},
  {-42.6249, 3684.09, -78481.6, 713514., -3.40327  $\times 10^6$ , 9.35501  $\times 10^6$ ,
  -1.53473  $\times 10^7$ , 1.48298  $\times 10^7$ , -7.78457  $\times 10^6$ , 1.71173  $\times 10^6$ },
  {203.587, -17593., 374728., -3.40646  $\times 10^6$ , 1.62465  $\times 10^7$ , -4.46556  $\times 10^7$ ,
  7.32551  $\times 10^7$ , -7.07811  $\times 10^7$ , 3.71533  $\times 10^7$ , -8.16923  $\times 10^6$ },
  {-560.255, 48407.1, -1.03095  $\times 10^6$ , 9.37099  $\times 10^6$ , -4.469  $\times 10^7$ ,
  1.22829  $\times 10^8$ , -2.01484  $\times 10^8$ , 1.94671  $\times 10^8$ , -1.0218  $\times 10^8$ , 2.24665  $\times 10^7$ },
  {919.989, -79478.8, 1.69253  $\times 10^6$ , -1.53834  $\times 10^7$ , 7.33587  $\times 10^7$ ,
  -2.01614  $\times 10^8$ , 3.30706  $\times 10^8$ , -3.19512  $\times 10^8$ , 1.67702  $\times 10^8$ , -3.6872  $\times 10^7$ },
  {-889.676, 76851.8, -1.63646  $\times 10^6$ , 1.48728  $\times 10^7$ , -7.09201  $\times 10^7$ ,
  1.94903  $\times 10^8$ , -3.19688  $\times 10^8$ , 3.08858  $\times 10^8$ , -1.62106  $\times 10^8$ , 3.56407  $\times 10^7$ },
  {467.338, -40365.8, 859477., -7.81086  $\times 10^6$ , 3.72439  $\times 10^7$ , -1.0235  $\times 10^8$ ,
  1.67874  $\times 10^8$ , -1.62182  $\times 10^8$ , 8.51204  $\times 10^7$ , -1.87143  $\times 10^7$ },
  {-102.823, 8880.54, -189075., 1.71822  $\times 10^6$ , -8.19253  $\times 10^6$ , 2.25132  $\times 10^7$ ,
  -3.69249  $\times 10^7$ , 3.56723  $\times 10^7$ , -1.8722  $\times 10^7$ , 4.11608  $\times 10^6$ }}
```

Tale risultato dovrebbe essere una matrice di tutti zero, ma non lo è

```
In[148]:= Max[%]
Out[148]=  $3.30706 \times 10^8$ 
```

Se si desiderano risultati numerici più affidabili si può usare una maggiore precisione nei calcoli. In questo caso l'impiego di 20 cifre di precisione è sufficiente per ottenere risultati accettabili

```
In[149]:= Inverse[hilbert] - Inverse[N[hilbert, 50]]
Out[149]= {{0. × 10-48, 0. × 10-47, 0. × 10-46, 0. × 10-45, 0. × 10-44, 0. × 10-44,
0. × 10-44, 0. × 10-44, 0. × 10-44, 0. × 10-45}, {0. × 10-47, 0. × 10-45, 0. × 10-44,
0. × 10-43, 0. × 10-42, 0. × 10-42, 0. × 10-42, 0. × 10-42, 0. × 10-42, 0. × 10-42, 0. × 10-43},
{0. × 10-46, 0. × 10-44, 0. × 10-42, 0. × 10-42, 0. × 10-41, 0. × 10-40, 0. × 10-40,
0. × 10-40, 0. × 10-41, 0. × 10-41}, {0. × 10-45, 0. × 10-43, 0. × 10-42,
0. × 10-41, 0. × 10-40, 0. × 10-39, 0. × 10-39, 0. × 10-39, 0. × 10-40, 0. × 10-40},
{0. × 10-44, 0. × 10-42, 0. × 10-41, 0. × 10-40, 0. × 10-39, 0. × 10-39, 0. × 10-39, 0. × 10-39,
0. × 10-39, 0. × 10-39, 0. × 10-40}, {0. × 10-44, 0. × 10-42, 0. × 10-40,
0. × 10-39, 0. × 10-39, 0. × 10-38, 0. × 10-38, 0. × 10-38, 0. × 10-38, 0. × 10-39},
{0. × 10-44, 0. × 10-42, 0. × 10-40, 0. × 10-39, 0. × 10-39, 0. × 10-38, 0. × 10-38,
0. × 10-38, 0. × 10-38, 0. × 10-39}, {0. × 10-44, 0. × 10-42, 0. × 10-40,
0. × 10-39, 0. × 10-39, 0. × 10-38, 0. × 10-38, 0. × 10-38, 0. × 10-38, 0. × 10-39},
{0. × 10-44, 0. × 10-42, 0. × 10-41, 0. × 10-40, 0. × 10-39, 0. × 10-38, 0. × 10-38,
0. × 10-38, 0. × 10-39, 0. × 10-39}, {0. × 10-45, 0. × 10-43, 0. × 10-41,
0. × 10-40, 0. × 10-40, 0. × 10-39, 0. × 10-39, 0. × 10-39, 0. × 10-40}}
```

```
In[150]:= Chop[Inverse[hilbert] - Inverse[N[hilbert, 50]]]
Out[150]= {{0, 0, 0, 0, 0, 0, 0, 0, 0, 0}, {0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
{0, 0, 0, 0, 0, 0, 0, 0, 0, 0}, {0, 0, 0, 0, 0, 0, 0, 0, 0, 0}, {0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
{0, 0, 0, 0, 0, 0, 0, 0, 0, 0}, {0, 0, 0, 0, 0, 0, 0, 0, 0, 0}, {0, 0, 0, 0, 0, 0, 0, 0, 0, 0}}
```