Capitolo 1

■ 1.1 Calcolo numerico

Innanzitutto *Mathematica* e' una calcolatrice...ma molto, molto piu' potente di una calcolatrice tradizionale. I simboli di addizione +, sottrazione -, divisione / , moltiplicazione *, elevamento a potenza ^, e il fattoriale ! hanno l'usuale significato. Il prodotto si puo' indicare anche con uno spazio: per esempio **2 3** e' lo stesso di **2*3**, ma e' diverso da **23**. La radice quadrata si denota con **Sqrt[*expr*]**. Per esempio

```
1 / 15 + 1 / 35 + 1 / 63
321 * 23 + 567 * 12 + 134 * (231 - 21)
2 6
(2 / 13) ^ 30
2 ^ 300
```

Il simbolo ¨· alla fine di ogni riga indica che l'espressione non e' finita ma si va a capo

```
30 !
Sqrt[75]
Binomial[9, 5](*calcola il coefficiente binomiale*)
```

Mathematica interpreta i numeri 6 oppure 1/15 come numeri "esatti" laddove con la notazione 6. oppure 1./15 oppure 1/15. denota i numeri "approssimati".

Mathematica e' progettato per il calcolo "esatto": non ritorna mai il risultato numerico a meno che non sia richiesto esplicitamente usando il comando **N[*expr*]** (o anche l'operatore *postfisso* ***expr*//N)** che richiede la valutazione numerica dell'espressione tra parentesi quadre. Per esempio

```
6. - 6
6 - 6
1. / 15 + 1 / 35 + 1 / 63

N[2 ^ 300]
2 ^ 300 // N
2 ^ 300.

N[Sqrt[75]]
Sqrt[75.]

Timing[N[30000 !]]
(*misura il tempo impiegato per il calcolo e scrive il risultato*)
```

Mathematica conosce un gran numero di costanti matematiche predefinite. Per esempio

```
Pi (* Pi greco*)

Pi // N

E (*E numero di Nepero*)

E // N
```

```
EulerGamma (*costante di Eulero*)

EulerGamma // N

Infinity

1 / Infinity

E^-Infinity

1^Infinity
```

Quando si usa il comando **//N** *Mathematica* visualizza 6 cifre dopo la virgola (anche se in realta' tiene conto di piu' cifre). E' possibile specificare la precisione e quindi il numero di cifre del risultato. Per esempio se vogliamo calcolare π con 35 cifre o con 200 cifre dovremo scrivere

```
N[Pi, 35]

N[Pi, 200]
```

Se provate a calcolare 2^10000000, rischiate di aspettare invano...e' bene sapere che in alcuni casi il calcolo si deve interrompere. Per far questo occorre utilizzare, nel menu' **Kernel,** l'opzione **Interrupt Evaluation**. Purtroppo a volte anche questo non basta e bisogna chiudere il **Kernel** (con il comando **Quit Kernel** nel menu **Kernel**) e, se anche questo non funziona, bisogna riavviare il computer . Per questo e' importante salvare sempre il file prodotto con il comando **Save** (oppure **Save as** se il *notebook* ancora non ha un nome) nel menu **File**, per non rischiare di perdere il lavoro fatto.

Con il simbolo **%** si indica l'ultimo output prodotto, **%%** indica il penultimo output prodotto mentre con il comando **%n** si indica l'output numero **n**. Questa notazione consente di abbreviare calcoli molto lunghi. Eseguite i seguenti calcoli

```
Sqrt[4]

%

%3 (* terzo output *)

%%% (* terzultimo output *)

(%3 + %5) * %%%

Numerator[%](*numeratore dell' espressione precedente *)

Denominator[%%](*denominatore della penultima espressione *)
```

1.1.1 Tipi di "espressioni"

• *Mathematica* conosce quattro tipi di numeri:

Integer : numeri interi esatti, per esempio, 123 ,

Rational: quoziente di due interi esatti,

Real: numeri reali "approssimati" (floating point, in virgola mobile) che si distinguono dagli interi per la presenza di un punto, per esempio 123. ,

Complex: numeri complessi della forma a+b **I**. Con **I** si denota l'unita' immaginaria tale che I^2 =-1

Il comando **Head** mi dice che tipo di "espressione" sto considerando

```
{Head[3], Head[1/3], Head[0.3], Head[0.3 + i]}
```

• *Mathematica* rappresenta ogni oggetto che si considera come una *expression*. Oltre ai numeri, *Mathematica* conosce molti altri tipi di espressioni. Per esempio *Mathematica* interpreta una sequenza finita di caratteri come un simbolo (**Symbol**). Per esempio **aaa** oppure **a1a2** sono simboli

```
Head[aaa]
Head[a1a2]
```

Una sequenza finita di caratteri, racchiusi da un doppio apice, e' una stringa (**String**)

```
Head["Buon Lavoro"]
StringLength["Buon lavoro"]
StringJoin["Buon lavoro", " ", "a casa"]
```

Incontreremo spesso una lista (**list**) cioe' una successione ordinata e finita di elementi (numeri, stringhe, simboli) racchiusi tra {} e separati tra loro da una virgola

```
lista = {a, 4, ssss, "ciao"}
Head[lista]
lista[[1]](*estraggo il primo elemento della lista*)
lista[[3]](*estraggo il terzo elemento della lista*)
```

Anche x+y e' una espressione. Quando si chiede a *Mathematica* di valutare x+y, il **Kernel** "traduce" l'espressione x+y nella forma **Plus[x,y]** cioe' nella rappresentazione interna o **FullForm.**

```
Head[x + y]
FullForm[x + y]

Head[x y]
FullForm[x y]

Head[x / y]
FullForm[x / y]
```

Il comando **Apply[NewHead,*expr*]** sostituisce il tipo di *expr* con NewHead. Per esempio

```
Head[x + y]
Apply[List, x + y]
Head[%]

Head[x^2 + y^2 z]
FullForm[x^2 + y^2 + z^2]
Apply[Times, x^2 + y^2 z]

Head[x / y]
FullForm[x / y]
Apply[Plus, x / y]
```

1.1.2 I numeri complessi

Si possono usare i numeri complessi come si usano i reali, ricordando che **I** e' l'unita' immaginaria.

```
Head[I]

I (* unita' immaginaria*)
```

```
{Sqrt[-4], Sqrt[-4.]}
```

```
(4 + 3 I) / (2 - I)
```

Si possono calcolare parte reale, parte immaginaria, coniugato, modulo, e argomento. Per esempio

```
Re[(4 + 3 I) / (2 - I)]
```

```
Im[(4 + 3 I) / (2 - I)]
```

```
Conjugate[(4 + 3 I) / (2 - I)]
```

```
Abs[(4 + 3 I) / (2 - I)]
```

```
Arg[(4 + 3 I) / (2 - I)]
```

■ 1.2 Calcolo simbolico: calcolo algebrico

Come gia' abbiamo sottolineato, una delle caratteristiche di *Mathematica* e' la possibilita' di eseguire calcoli simbolici cioe' calcoli dove l'output non e' un numero.

Nel prossimo esempio *Mathematica* esegue le semplificazioni algebriche piu' semplici

```
-1 + 2 x + x^3 - 5 x
```

Se si vuole mantenere inalterata un'espressione i comandi da usare sono **Hold[*expr*]** e **HoldForm[*expr*]**. La differenza e' che **Hold** viene scritto esplicitamente nell'output mentre **HoldForm** e' invisibile nell'output di *Mathematica*. **ReleaseHold** elimina **Hold** e **HoldForm**, in modo che le espressioni vengano calcolate. Per esempio

```
Hold[-1 + 2 x + x^3 - 5 x]
```

```
HoldForm[-1 + 2 x + x^3 - 5 x]
```

```
ReleaseHold[%]
```

Di solito *Mathematica* non sviluppa i calcoli algebrici, a meno che non glielo chiediamo esplicitamente, per esempio, con il comando **//Expand**

```
(x - y)^5
(x - y) (x + y)
```

```
(x - y)^5 // Expand
(x - y) (x + y) // Expand
```

Se non si vuole visualizzare il risultato di una operazione si mette **;** *Mathematica* esegue il comando ma non lo visualizza. Cio' puo' essere molto utile se il risultato di un'espressione e' molto lungo e non e' necessario visualizzarlo. Assegnamo all'espressione razionale seguente il nome *expr*. Quando eseguiamo il comando, *Mathematica* valuta l'espressione e il suo risultato viene assegnato al simbolo *expr*.

```
expr = (x - 1)^2 (2 + x) / ((1 + x) (x - 3)^2);
```

Notate la differenza tra il comando **Expand** e **ExpandAll**

```
expr // Expand
```

```
expr // ExpandAll
```

Il comando **//Apart** riduce alla somma di espressioni con singoli denominatori

```
expr // Apart
```

Il comando **//Factor** e' sostanzialmente l'inverso dei comandi **//Expand** e **//ExpandAll**

```
4 + 4 x + x^2 // Factor
```

Il comando **Together** riduce a denominatore comune

```
4 / x + 7 / (x - 1) - 8 x^2 / (x + 2) // Together
```

Il modo piu' semplice di semplificare una espressione e' usare il comando **//Simplify** (che spesso ha lo stesso effetto di **//Factor**) o **//FullSimplify** (piu' potente, ma che spesso richiede piu' tempo)

```
pol = 32 + 16 x - 48 x^2 - 40 x^3 + 10 x^4 + 21 x^5 + 8 x^6 + x^7
```

```
pol // Simplify
```

```
pol // Factor
```

```
Simplify[ Sqrt[(2-x)/(3+x)] / Sqrt[2 - x] ]
```

```
FullSimplify[ Sqrt[(2-x)/(3+x)] / Sqrt[2 - x] ]
```

```
Simplify[Gamma[z] Gamma[1 - z]]
```

```
FullSimplify[Gamma[z] Gamma[1 - z]]
```

Entrambi questi comandi possono eseguire le semplificazioni sotto opportune condizioni con la sintassi **Full-Simplfy[expr,ass]** oppure **Simplify[expr,ass].** Per esempio

```
Simplify[Sqrt[(1 - x^2)^2]]
```

```
Simplify[Sqrt[(1 - x^2)^2], 0 ≤ x ≤ 1]
```

Definiamo **pol** in modo diverso cioe'

```
pol = Expand[(1 + 3 x + 4 y^2)^2]
```

Il comando **Coefficient[pol,x^2]** fornisce il coefficiente di **x** nel polinomio **pol** mentre il comando **Part[pol,n]** o anche **pol[[n]]** da' il termine nella posizione **ennesima** che compare in **pol**

```
Coefficient[pol, x^2]
Part[pol, 4]
pol[[4]]
```

Consideriamo i due polinomi nella variabile **x** dipendenti dal parametro a

```
p1 = 1 - 3 a^2 + (2 - a^2) x + x^3;
p2 = x + a;
```

Calcoliamo, rispettivamente, quoziente e resto della divisione dei polinomi in **x**, **p1** con **p2**

```
q = PolynomialQuotient[p1, p2, x]
r = PolynomialRemainder[p1, p2, x]
```

Verifichiamo il risultato

```
Simplify[q p2 + r]
% == p1 // Simplify
```

Abbiamo appena usato l'operatore di relazione **==** sul quale torneremo piu' avanti.

Esercizo. Eseguite la divisione di p1=x^3+12 x^2+41 x+34 per p2=x+5.

Naturalmente non e' importante ricordare tutte le operazioni algebriche disponibili (qui ne abbiamo richiamate solo una parte!!): queste sono tutte riportate nell'**Help Browser** alla voce **Algebraic Computation** oppure nel submenu **Palettes** del menu **File.**

Attenzione: se chiediamo a Mathematica di calcolare **Expand[(1+x)^100(^1000)]** rischiamo di aspettare anche un anno...a meno che *Mathematica* non ci invii il messaggio **out of memory.** In tal caso l'unica alternativa e' riavviare il **Kernel.**

■ 1.3 Rappresentazione degli Input e degli Output

E' possibile rappresentare un'espressione matematica nel *notebook* con diverse notazioni:

InputForm[expr] forma usata comunemente per rappresentare *expr* usando solo i caratteri della tastiera su una riga;

StandardForm[expr] visualizza i dati di input e di output usando caratteri speciali (vedi nella barra principale il menu' **File** e il sottomenu' **Palettes**) e la notazione matematica usuale;

OutputForm[expr] forma per l' output che usa solo i caratteri della tastiera ma con la notazione matematica usuale (per esempio per le potenze etc.) ;

TraditionalForm[expr] visualizza gli output in maniera "elegante" molto simile alla tradizionale notazione matematica, anche con stili diversi.

Nel menu a barra della finestra principale, selezionando **Cell** e poi **Convert To** e' possibile convertire le celle da una forma all'altra.

Vediamo la differenza con alcuni esempi. Nelle righe seguenti vediamo lo stesso input scritto in **InputForm** e **StandardFrom**, rispettivamente. Notate come cambia la parentesi quadra nella cella a destra.

```
Pi (*InputForm scritto solo con l' ausilio della tastiera*)
```

π `(*Standard Form scritto con i caratteri della Palette BasicInput*)`

```
x^2 + y^2/z (*InputForm *)
```

$x^2 + \frac{y^2}{z}$ `(*StandardForm scritto con i caratteri della Palette BasicInput *)`

Un altro esempio:

```
Sqrt[x] + 1/(2 + Sqrt[y])(*InputForm*)
```

$\sqrt{x} + \frac{1}{2 + \sqrt{y}}$ `(*StandardForm*)`

Nei prossimi esempi generiamo l'output, rispettivamente, in **InputForm**, **OutputForm**, **TraditionalForm** e **StandardForm**. Per *default* l'output e' rappresentato in **StandardForm**.

```
Exp[I Pi x] // InputForm
E^(I*Pi*x)
```

```
Exp[I Pi x] // OutputForm
```

```
E^(I Pi x)
```

```
Exp[I Pi x] // TraditionalForm
```

$$e^{i\pi x}$$

```
Exp[I Pi x] // StandardForm
```

$$e^{i\pi x}$$

Questi non sono le uniche possibilita' per generare l'output. Per esempio e' possibile rappresentare l'output con la sintassi del linguaggio C, del TeX e del Fortran con i comandi **CForm[expr]**, **TeXForm[expr]** e **FortranForm[expr]**, rispettivamente.

■ 1.4 Regola di sostituzione immediata

Consideriamo un polinomio (o una qualsiasi altra funzione) della **x**. Se voglio valutare il polinomio quando **x** vale x_0 (senza pero' assegnare a **x** il valore x_0), si puo' usare il comando **polinomio/.x->**x_0 cioe' **x** viene sostituito da $\boldsymbol{x_0}$ nell'espressione **polinomio** scritta prima di **/. (ReplaceAll o regola di sostituzione immediata)**

```
x + 2 /. x → 3
```

```
x + 2 /. x → a
```

```
x + 2 /. x → 3 + y
```

```
(x - 1) ^ 4 + y ^ 2 /. y → 1 / x
```

Nella seguente riga di comando prima sostituisco **a** con **b** e successivamente **b** con **d**:

```
a + 2 b + c /. a → b /. b → d
```

oppure posso sostituire contemporaneamente **a** con **b** e **b** con **d...**attenzione che ottengo un risultato diverso

```
a + 2 b + c /. {a → b, b → d}
```

```
x + x^2 /. {x → a, x^2 → x}
x + x^2 /. x → a /. x^2 → x
x + x^2 /. x^2 → x /. x → a
```

```
(x - 1) ^ 4 + y ^ 2 /. {x → b, y → a}
```

```
x ^ 2 + y ^ 2 + z ^ 2 /. {x → a + b, y → c + d, z → e + f}
```

```
Log[x] /. x → 5
```

Usando | posso specificare piu' alternative. Nelle seguenti righe sostituiamo entrambe **x** e **y** con **a**

```
{x, 0, 1} /. x | y → a
```

```
{x, y, 1} /. x | y → a
```

Esercizio 1. Dimostrare, usando *Mathematica*, che x^3+y^3+z^3=3 x y z se x+y+z=0.

Sia

```
w = x ^ 3 + y ^ 3 + z ^ 3;
```

Sostituiamo a z l'espressione -(x+y) con la regola di sostituzione immediata

```
w /. z → -(x + y)
```

```
Simplify[%]
```

```
% /. x + y → -z
```

■ 1.5 Assegnazioni (operatori Set e SetDelayed)

Se si vuole assegnare il valore 3 al simbolo **x** si usa l'operatore di **assegnazione immediata = (Set)**

```
Head[x]
```

```
x = 3
```

```
Head[x]
```

```
x^3
```

```
x = 4; x^2
```

Si possono assegnare piu' comandi su una stessa riga, separandoli da **;**

```
y = 2; z = 4; Sqrt[y^2 + z^2]
```

Per cancellare il valore attribuito alla variabile x si usa il comando **Clear[x]** oppure il comando **x=.** Per ripulire contemporaneamente x, y, z si puo' fare

```
Clear[x, y, z]
```

Oppure, per ripulire tutte le variabile usate fino ad ora, si puo' usare il comando drastico

```
Clear["Global`*"]
```

In realta' per assegnare dei valori a delle variabili (anzi simboli) si puo' usare, oltre al segno = anche il comando di **assegnazione in differita := (SetDelayed)**. Se si scrive **x=***valore* allora il valore viene immediatamente calcolato e il valore e' assegnato a **x** una volta per tutte. Se si scrive **x:=***valore*, allora il valore non e' immediatamente calcolato ma viene calcolato ogni volta che **x** viene richiamata.

```
a = 2; x := a; x
```

```
a = 3; x
```

```
a = 6; x
```

Provate invece a fare

```
a = 2; x = a; x
```

```
a = 3; x
```

```
a = 6; x
```

Notate la differenza?

```
Clear[a, x]
```

Mathematica contiene la funzione **Random** che genera numeri casuali (in realta' pseudocasuali, perche' prodotti applicando un algoritmo matematico partendo da un dato valore iniziale)

```
Random[Real, {0, 5}](*genera un reale tra 0 e 5*)

Random[Integer, {0, 100}](*genera un intero tra 0 e 100*)
```

Se non si specifica il tipo e il range *Mathematica* genera un numero reale tra 0 e 1.

```
Random[]
```

Per capire meglio la differenza tra **Set** e **SetDelayed**, provate a eseguire i seguenti comandi

```
x = Random[]

y := Random[]

{x, y}

{x, y}

{x, y}

Clear[x, y]
```

■ 1.6 Operatori di relazione e Operatori Logici

Gli operatori di relazione sono usati per confrontare due espressioni e hanno come valore di output *True* o *False*. Con *Mathematica* gli operatori di relazione sono **Equal**[x,y] (**==**), **Unequal**[x,y](**!=**), **Greater**[x,y](**>**), **Less**[x,y](**<**), **GreaterEqual**[x,y] (**>=**), **LessEqual**[x,y](**<=**).

```
7<5
5<7≤8
Equal[3,7-4,6/2]
Equal[3,7-3]
```

Ci chiediamo se le seguenti espressioni sono vere o false

```
3 == 4
4 x - x^2 == -x^2 + 4 x
```

Gli operatori logici (o Booleani) stabiliscono la validita' di una relazione basata sulla aritmetica booleana.

L'operatore **And[x,y,...]** si denota anche con **&&** : valuta i suoi argomenti in ordine ed ha valore False se uno dei suoi argomenti e' falso ed ha valore *True* se tutti i suoi argomenti sono veri.

L'operatore logico **Or[x,y]** si denota anche con || : valuta i suoi argomenti in ordine ed ha il valore *True* se almeno uno dei suoi argomenti e' vero ed ha il valore False se sono tutti falsi.

L'operatore **Not[x]** si denota anche con **!** . **!expr** ha valore *False* se *expr* e' vera e *True* se *expr* e' falsa.

```
9 > 4 && 5 == 3 + 2
9 < 4 || 5 < 6
(1 < 2) && ! (4 < 2)
```