Capitolo 2

■ 2.1 Le funzioni elementari

Le funzioni matematiche elementari piu' comuni sono **Exp[x], Sqrt[x]**, **Log[x]**, **Log[b,x]** (logaritmo in base b di x), le funzioni trigonometriche **Sin[x]**, **Cos[x]**, **Tan[x]**, e le loro inverse **ArcSin[x]**, **ArcCos[x]**, **ArcTan[x]**, le funzioni iperboliche **Sinh[x]**, **Cosh[x]**, **Tanh[x]**, e il valore assoluto **Abs[x]**. Queste funzioni sono predefinite in *Mathematica* (vedete nell'**Help Browser** la voce **Built-in functions, Mathematical Functions**, **Elementary Functions**). Per esempio eseguite i seguenti comandi

```
Clear["Global`*"]
(*con questo comando ripuliamo le variabili usate fino ad ora*)

Log[E]
Log[E] // HoldForm

Log[2, 1024]

Cos[Pi / 3]
Cos[Pi / 3] // HoldForm

ArcTan[1]

Sin[x - Pi / 2]
Sin[x - Pi / 2] // HoldForm

Abs[1 / 2 + Sqrt[3] I]
```

Esempio. *Mathematica* sa che **Log** ed **Exp** sono una l'inversa dell'altra cioe', se gli chiedo

```
Exp[Log[x]]
```

Se mi interessa che non venga eseguita la semplificazione posso usare **Hold** oppure **HoldForm**:

```
Hold[Exp[Log[x]]]
HoldForm[Exp[Log[x]]]
```

Il comando **ReleaseHold** rimuove **Hold** e **HoldForm**

```
ReleaseHold[Hold[Exp[Log[x]]]]
```

Altre funzioni elementari sono **Round[x]** (il numero intero piu' vicino a x, che non e' la parte intera di x), **Max[x,y,...], Min[x,y,...]**.

```
Round[Pi]
Round[E]
Max[Pi, E, EulerGamma]
Min[Pi, E, EulerGamma]
```

Esercizi. 1. determinare x tale che $3^x = 30$;

2. determinare x tale che sin(x)=.902;

3. determinare x tale che tg(x)=.902

2.1.1 Semplificazioni trigonometriche

Mathematica conosce (e non sbaglia) le formule di trigonometria.

TrigExpand sviluppa l'espressione trigonometrica come combinazione di funzioni trigonometriche in funzione dell'angolo piu' piccolo

```
Sin[2 x] // TrigExpand

Cos[x + y] // TrigExpand

Tan[2 x] // TrigExpand

Tan[x] Cos[2 x] // TrigExpand

Cos[x] ^ 2 - Sin[x] ^ 2 // TrigExpand
```

TrigFactor fattorizza l'espressione trigonometrica come prodotto di termini

```
Tan[x] Cos[2 x] // TrigFactor

Cos[x] ^ 2 - Sin[x] ^ 2 // TrigFactor
```

TrigReduce riduce l'espressione trigonometrica usando gli angoli multipli

```
Tan[x] Cos[2 x] // TrigReduce

Cos[x] ^ 2 - Sin[x] ^ 2 // TrigReduce
```

Con il comando **?Funzione** si ottengono informazioni sulla **Funzione**:

```
? TrigReduce

? TrigExpand
```

Possono essere utili anche i comandi **TrigToExp** e **ExpToTrig** che trasformano funzioni trigonometriche in funzioni esponenziali e viceversa. Per esempio

```
TrigToExp[Sin[x]]

ExpToTrig[Exp[I x]]

ExpToTrig[Exp[2 x]]
```

Esempio. Sviluppare $(\sin x + i \cos x)^3$ ed esprimere il risultato in termini di sin e cos di angoli multipli.

```
(Sin[x] + I Cos[x]) ^ 3 // TrigExpand

TrigReduce[%]
```

Esercizio. Sviluppare $(\sin(2x) + \cos(5x))^5$ ed esprimere il risultato in termini di sin e cos di angoli multipli.

■ 2.2 Definire funzioni

```
ClearAll["Global`*"]
```

Per definire la funzione f(x)=x^2 si usa il seguente comando

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

Notate il segno di sottolineatura _ (detta *blank*) a sinistra nella definizione di f cioe' **x_** che sostanzialmente informa *Mathematica* che la variabile **x** e' un parametro destinato ad essere sostituito di volta in volta, al momento che viene chiesta la valutazione della funzione

```
{f[x], f[a], f[a^2], f[a + 1], f[3], f[3 x + x^2]}
```

```
? f
```

```
Clear[f]
```

Se usiamo **x** invece di **x_** per definire la funzione g, il meccanismo di sostituzione non funziona

```
g[x] := (1 + x)^2
```

```
{g[x], g[a], g[a^2], g[a + 1], g[3], g[3 x + x^2]}
```

```
Clear[g]
```

Se usiamo **=** (**Set**) anziche' **:=** (**SetDelayed**) il termine a destra della definizione di funzione viene calcolato immediatamente una volta per tutte, e non ogni volta che si richiama

```
h[x_] = Expand[(1 + x)^2]
```

```
g[x_] := Expand[(1 + x)^2]
```

Notate che la definizione della funzione **h** fornisce l'output mentre la valutazione di **g** viene differita al momento in cui si applica a **x**

```
h[x + 2]
g[x + 2]
```

```
Clear[f, g]
```

Se si e' in dubbio se usare **=** oppure **:=** e' consigliabile usare **:=**.

Esempio. Valutare il tempo che *Mathematica* impiega a disegnare il grafico delle funzioni

```
f1[x_] := Expand[(Cos[x] + Sin[x])^(1000)]
f2[x_] = Expand[(Cos[x] + Sin[x])^(1000)];
```

```
Timing[Plot[f1[x], {x, -Pi / 2, Pi / 2}]]
```

```
{0.05 Second, -Graphics-}
```

```
Timing[Plot[f2[x], {x, -Pi / 2, Pi / 2}]]
```

```
{22.57 Second, -Graphics-}
```

Esempio La funzione f(x)=$\frac{\sin x}{x}$, f(0)=1 si puo' definire nel seguente modo

```
f[x_] := Sin[x] / x; f[0] = 1;
```

La definizione specifica **f[0]=1** prevale su quella generale.

```
? f
```

Nello stesso modo si possono definire funzioni di due o piu' variabili. Per esempio

```
Clear[f]; f[x_, y_] := x^2 + y^2
```

oppure funzioni a valori vettoriali $f(x, y) = (f_1(x, y), f_2(x, y), \ldots, f_n(x, y))$

```
f[x_, y_] := {x^2 + y^2, y}
```

Esempio Sia $f(x, y) = x + y$ e sia $g(x) = (\cos(x), \sin(x))$. Calcolare la funzione composta $h(x) = f(g(x))$.

```
f[{x_, y_}] := x + y
g[x_] := {Cos[x], Sin[x]}
h[x_] = f[g[x]]
```

Esempio E se volessimo definire la funzione che vale x^2 se **x≤0** e vale **x** se **x>0**? Una possibilita' e' usare il comando **lhs:=rhs/;test** che definisce lhs:=rhs "a condizione" che *test* sia verificato altrimenti non assegna nulla. Nel nostro caso, quindi,

```
Clear[f]
f[x_] := x^2 /; x ≤ 0
f[x_] := x /; x > 0

Plot[f[x], {x, -3, 3}](*disegnamo il grafico di f*)
```

N.B. Se per definire una funzione sono fatte piu' definizioni con condizione, vengono considerate in ordine cronologico fino a quando l'argomento ne verifica una

```
Clear[f]
f[x_] := x^3 /; x > 0
f[x_] := x + 1 /; x >= 0

{f[-1], f[0], f[2]}

Plot[f[x], {x, 0, 3}](*disegnamo il grafico di f*)
```

Vedremo che questo non e' l'unico modo in cui possiamo definire **f**.

Esempio Definiamo la funzione f(x)=log($x^2 - 1$) per x>1 oppure x<-1:

```
Clear[f]
f[x_] := Log[x^2 - 1] /; ((x > 1) || (x < -1))
{f[2], f[-2], f[0]}
```

Anziche' scrivere due definizioni per **f**, ho scritto le due condizioni racchiuse tra parentesi e collegate dall'operatore logico || (**Or**).

La condizione sulla variabile **x** si puo' anche scrivere nell'argomento della funzione. Per esempio la funzioni f dell'ultimo esempio si puo' anche definire

```
g[x_ /; ((x > 1) || (x < -1))] := Log[x^2 - 1]
{g[2], g[-2], g[0]}

Clear[f]
f[x_ /; x > 0] := Sqrt[x]
f[x_ /; x ≤ 0] := 2 Sqrt[-x]
{f[-4], f[4]}
```

La seguente funzione definisce l'equazione della circonferenza di centro $P_0 = (x_0, y_0)$ e raggio r>0

```
eqcirconferenza[{x0_, y0_}, r_ /; r > 0] :=
 Expand[(x - x0)^2 + (y - y0)^2 - r^2] == 0

P = {1, 3}; eqcirconferenza[P, 2]

eqcirconferenza[P, -2]
```

La seguente funzione definisce l'equazione della retta passante per due punti distinti $P_1 = (x_1, y_1)$ e $P_2 = (x_2, y_2)$ del piano

```
retta2Punti[{x1_, y1_}, {x2_, y2_}] :=
 Simplify[(y2 - y1) * (x - x1) - (x2 - x1) * (y - y1)] == 0 /; ((x1 ≠ x2) || (y1 ≠ y2))
```

Notate che la condizione **/;** coinvolge entrambi i punti e quindi non puo' essere inserita nell'argomento.

Con il comando **Roots[eq==0,y]** si trovano le soluzioni dell'equazione eq==0 nell'incognita y.

```
P1 = {2, 3}; P2 = {5, 4}; retta2Punti[P1, P2]
Roots[%, y]
```

La seguente funzione definisce l'equazione della retta passante per il punto (x_0, y_0) e di coefficiente angolare m

```
rettacoefang[{x0_, y0_}, m_] :=
 y - y0 == m (x - x0) (*eq. della retta per (x0,y0) e coefficiente angolare m*)

P = {1, 2}; rettacoefang[P, 3]
Roots[%, y]
```

Fino ad ora non abbiamo posto condizioni sul tipo di argomento **x** nella definizione **f[x_]:=valore**.

```
Clear[f]
f[x_] := x
{f[2], f["stringa"], f[{lista}], f[simbolo], f[1 + 2 I], f[3 / 4]}
```

Posso specificare che l'argomento di una funzione sia un numero reale o complesso o razionale o intero o una lista o una stringa o un simbolo sostituendo **x_** nella definizione con **x_h** dove **h** e' uno dei tipi sopra specificati.

Definiamo la seguente funzione

```
f1[x_] := Log[x] / Log[x + 1]
{f1[y], f1[-4.], f1[4], f1[4.], f1[3 / 4], f1[I], f1["stringa"], f1[{lista}]}
```

Definiamo **f1** solo per valori di **x** reali

```
f2[x_Real] := Log[x] / Log[x + 1]
{f2[y], f2[-4.], f2[4], f2[4.], f2[3 / 4], f2[I], f2["stringa"], f2[{lista}]}
```

oppure solo per valori di **x** reali positivi

```
f3[x_Real /; x > 0] := Log[x] / Log[x + 1]
{f3[y], f3[-4.], f3[4], f3[4.], f3[3 / 4]}
```

o equivalentemente

```
? Positive

f3[x_Real /; Positive[x]] := Log[x] / Log[x + 1]
{f3[y], f3[-4.], f3[4], f3[4.], f3[3 / 4]}
```

La funzione f3 non viene calcolata in **y**, perche' non e' un numero reale ma un simbolo; non viene calcolata in -4<0, pero' non viene nemmeno calcolata in 4, che e' interpretato da *Mathematica* come un intero e non un reale approssimato, e analogamente nel numero razionale 3/4. Posso modificare la definizione in modo da includere anche i numeri interi positivi, usando il simbolo **|** che permette di scegliere tra piu' alternative

```
f4[(x_Real | x_Integer) /; x > 0] := Log[x] / Log[x + 1]
{f4[y], f4[-4.], f4[4], f4[4.], f4[3 / 4]}
```

o equivalentemente usando **x:**

```
f4bis[x : (_Real | _Integer) /; x > 0] := Log[x] / Log[x + 1]
{f4bis[y], f4bis[-4.], f4bis[4], f4bis[4.], f4bis[3 / 4]}
```

oppure includendo la possibilita' che **x** sia anche un razionale positivo

```
f5[((x_Real) | (x_Integer) | (x_Rational)) /; Positive[x]] :=
 Log[x] / Log[x + 1]
{f5[y], f5[-4.], f5[4], f5[4.], f5[3 / 4]}

f5bis[x : (_Real | _Integer | _Rational) /; Positive[x]] := Log[x] / Log[x + 1]
{f5bis[y], f5bis[-4.], f5bis[4], f5bis[4.], f5bis[3 / 4]}

Clear[f, f1, f2, f3, f4, f5]
```

Si possono analogamente usare i comandi **NumberQ[x], IntegerQ[x], Negative[x], NonPositive[x], NonNegative[x], EvenQ[x], OddQ[x]** dopo **/;** che hanno valore *True* se **x** e', rispettivamente, un numero di qualunque tipo (quindi non un simbolo), un intero, un numero negativo, un numero non negativo, un numero non positive, un numero intero pari, un numero intero dispari.

Per esempio, definiamo **g1** sui numeri di ogni tipo (ma non sui simboli o sulle liste)

```
g1[x_ /; NumberQ[x]] := x^2
{g1[a], g1[4], g1[4.], g1[3 / 2], g1[1 + I], g1[{a, b}]}
```

Posso imporre piu' condizioni, separandole con gli operatori logici **And (&&)**, **Or (||)** e **Not (!)** . Definiamo **g2** sui numeri reali di ogni tipo (non sui complessi)

```
g2[x_ /; (NumberQ[x] && Im[x] == 0)] := x^2
{g2[a], g2[4], g2[4.], g2[3 / 2], g2[1 + I]}
```

Definiamo quindi **g3** sui numeri reali positivi

```
g3[x_ /; (NumberQ[x] && Im[x] == 0 && x > 0)] := Log[x] / Log[x + 1]
{g3[y], g3[-4.], g3[4], g3[4.], g3[3 / 4]}
```

Esempio. Definiamo una funzione su liste di lunghezza 2 tale che $f[\{l1,l2\}]=l1^2+l2^2$

```
Clear[f]
f[x_List /; Length[x] == 2] := x[[1]]^2 + x[[2]]^2
```

Esempio. Definiamo una funzione che calcoli il prodotto scalare tra due vettori della stessa lunghezza e avverte quando i vettori non hanno la stessa lunghezza.

```
g[x_List, y_List] := x.y /; Length[x] == Length[y]
g[x_List, y_List] :=
 {"Non e' possibile calcolare il prodotto scalare tra ", x, " e ", y}
```

Esempio. Definiamo f(x) solo sugli interi che vale x! se x e' intero positivo pari; vale (2x)! se e' intero positivo dispari, e vale x altrimenti

```
Clear[f]
f[x_Integer /; (Positive[x] && EvenQ[x])] := x!
f[x_Integer /; (Positive[x] && OddQ[x])] := (2 x)!
f[x_Integer /; NonPositive[x]] := x
{f[4], f[3], f[3.], f[-3]}
```

■ 2.3 Disegnare funzioni di una variabile - Grafici 2D (prima parte)

Il comando per disegnare una funzione f nell'intervallo (a,b) e' **Plot[f[x],{x,a,b}]** (oppure **Plot[{f1[x],f2[x],...},{x,a,b}]** per disegnare due o piu' funzioni contemporaneamente)

```
Plot[Sin[x], {x, 0, 2 Pi}]
```

```
Plot[{Sin[x], Sin[2 x], Sin[3 x]}, {x, 0, 2 Pi}]
```

Notate che l'*output* mostra la parola *Graphics* cioe' *Mathematica* ha creato un *oggetto grafico* e lo ha visualizzato. Infatti

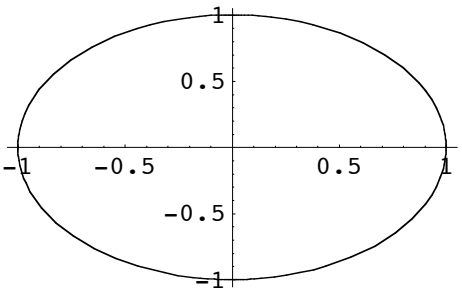
```
Head[Plot[Sin[x], {x, 0, 2 Pi}]]
```

Se in un intervallo la funzione non e' definita, *Mathematica* fa il possibile per disegnarla

```
Plot[Sin[1 / x], {x, -1, 1}]
```

Quando la curva del piano e' rappresentata in forma parametrica allora si deve usare il comando **Parametric-Plot[{f1[t],f2[t]},{t,a,b}]** (o il comando **ParametricPlot[{{f1[t],f2[t]},{g1[t],g2[t]},...},{t,a,b}]** per disegnare piu' funzioni)

```
ParametricPlot[{Sin[t], Cos[t]}, {t, 0, 2 Pi}]
```




```
- Graphics -
```

```
ParametricPlot[{{Sin[t], Cos[t]}, {3 Cos[t], 3 Sin[t]}}, {t, 0, 2 Pi}]
```

Ci sembrera' strano che non otteniamo una circonferenza perche' *Mathematica* ha usato sugli assi due diverse unita' di misura. L'*opzione grafica* **AspectRatio->r** fissa il rapporto tra altezza H e larghezza W del rettangolo che contiene il disegno uguale a r=H/W e quindi, di conseguenza,fissa la scala sugli assi. Se **r=1** il grafico e' contenuto in un quadrato. Per *default* **r=1/GoldenRatio**

```
? GoldenRatio
```

Per avere un grafico con la stessa scala sugli assi bisogna specificare **AspectRatio->Automatic**

```
ParametricPlot[{Sin[t], Cos[t]}, {t, 0, 2 Pi}, AspectRatio → 1]
```

```
ParametricPlot[{Sin[t], Cos[t]}, {t, 0, 2 Pi}, AspectRatio → Automatic]
```

```
ParametricPlot[{{Sin[t], Cos[t]}, {3 Cos[t], 3 Sin[t]}},
 {t, 0, 2 Pi}, AspectRatio → Automatic]
```

2.3.1 Opzioni grafiche

Quando *Mathematica* disegna un grafico deve fare diverse scelte: la scala sugli assi, come disegnare gli assi, il colore dello sfondo, se disegnare una cornice, se disegnare una griglia... e' possibile scegliere il modo in cui visualizzare il grafico specificando una o piu' *opzioni grafiche (graphic options)* nel comando **Plot** o **ParametricPlot** con l'istruzione **Opzione-> valore** (come abbiamo fatto con **AspectRatio**). Le opzioni grafiche riguardano proprieta' globali dell'oggetto grafico. Ogni opzione, se non menzionata, resta fissata a un certo valore di *default*.

Vediamo alcune delle *opzioni grafiche* maggiormente usate:

• **PlotLabel->"text"** scrive il *label* text sul disegno. Il testo da scrivere va messo sempre tra virgolette, per distinguerlo dai comandi di grafica.

```
ParametricPlot[{Cos[19 t], Sin[20 t]}, {t, 0, 2 Pi},
 AspectRatio → Automatic, PlotLabel → "Curva di Lissajous"]
```

```
ParametricPlot[{6 t/(1 + t^3), 6 t^2/(1 + t^3)},
 {t, -1, 100}, PlotLabel -> "Folium di Cartesio"]
```

• **PlotRange->{c,d}** disegna la parte di grafico con il valore di y tra c e d cioe' disegna il grafico di y=f[x] nel rettangolo [a,b]x[c,d]. Per *default* **PlotRange->Automatic**, cioe' *Mathematica* sceglie cosa disegnare. Se si vogliono disegnare tutti i punti del grafico si deve usare **PlotRange->All** (per essere sicuri di visualizzare ogni parte del grafico)

```
Plot[Exp[-x^2], {x, -10, 10}, PlotLabel -> "Curva Gaussiana"]
```

```
Plot[Exp[-x^2], {x, -10, 10},
 PlotLabel -> "Curva Gaussiana", PlotRange → All]
```

```
Plot[Sin[x^2], {x, 0, 3}, PlotRange → {0, 1.2}]
```

```
Plot[2 - Sin[x^2], {x, 0, 3}, PlotRange → {0, 3}]
```

• **Frame->True** disegna una cornice intorno al grafico. Per *default* **Frame->False**.

```
Plot[Sin[x^2], {x, 0, 3 Pi}, Frame → True]
```

• **Gridlines->Automatic** disegna automaticamente una griglia. Per *default* **Gridlines->None** ma si puo' anche specificare la disposizione delle linee verticali e orizzontali con **GridLines->{{x1,x2,...},{y1,y2,...}}.**

```
Plot[Sin[x^2], {x, 0, 3 Pi}, GridLines → {{Pi / 2}, {-0.5, 0.5}}]
```

```
Plot[Sin[x^2], {x, 0, 3 Pi}, GridLines → Automatic]
```

• **Axes->False** non disegna gli assi coordinati. Per *default* **Axes->True.** Il comando **Axes->{False,True}** non disegna l'asse x ma disegna l'asse y**.**

• **AxesOrigin->{x0,y0}** specifica che gli assi si incontrino nel punto di coordinate **{x0,y0}**. Per *default* **AxesOrigin->Automatic** cioe' il sistema decide il punto d'intersezione.

• **AxesLabel ->{"xlabel","ylabel"}** mette i nomi agli assi x e y. Per *default* **AxesLabel->None.**

```
Plot[Cos[x^2], {x, 0, 3 Pi}, AxesLabel → {"x", "Cos[x^2]"}]
```

• **Ticks->None** non scrive le tacche sugli assi coordinati. Per *default* **Ticks->Automatic** cioe' *Mathematica* decide automaticamente. Con il comando **Ticks->{{x1,x2,...},{y1,y2,...}}** si possono decidere le posizioni da segnare su entrambi gli assi.

```
ParametricPlot[{3 t - Sin[t], 3 - 3 Cos[t]},
 {t, 0, 2 Pi}, AspectRatio → Automatic,
 Ticks → {{0, Pi, 2 Pi, 3 Pi, 4 Pi, 5 Pi, 6 Pi}, {2, 4, 6, 8}},
 PlotLabel -> "Cicloide"]
```

• **DisplayFunction→Identity** crea l'oggetto grafico ma non lo visualizza. Per *default* **DisplayFunction-> $DisplayFunction**.

• **Background->GrayLevel[i]** oppure **Background->RGBColor[r,g,b]** specifica il colore dello sfondo del grafico. Per esempio bianco=GrayLevel[1], nero=GrayLevel[0], rosso=RGBColor[1,0,0], verde=RGBColor[0,1,0], blu=-RGBColor[0,0,1], giallo=RGBColor[1,1,0], viola=RGBColor[1,0,1], bianco=RGBColor[1,1,1], nero=RGBColor[0,0,0].

```
Plot[Cos[x^2] Sin[x], {x, 0, 3 Pi}, Background -> GrayLevel[0.3]]
```

L'elenco dei colori disponibili si trova nel *package* **Graphics'Colors' .** Per la lista completa dei colori rimando all'*Help browser*.

```
Needs["Graphics`Colors`"]
Orange
Pink
Plot[Cos[x^2] Sin[x], {x, 0, 3 Pi}, Background -> Pink]
```

E' interessante notare che *Mathematica* si ricorda di ogni grafico che produce, che quindi puo' rivisualizzare, eventualmente cambiando leopzioni grafiche usate precedentemente.Il comando **Show[plot]** rivisualizza il grafico **plot**, mentre il comando **Show[plot,Options->valore]** rivisualizza il grafico cambiando le opzioni grafiche.

```
grafico = Plot[Sin[x^2], {x, 0, 3 Pi}]
```

Il comando **InputForm[grafico]** mostra le opzioni usate per disegnare **grafico**.

```
InputForm[grafico]
```

Cambio alcune opzioni grafiche

```
Show[grafico, PlotLabel -> "sin(x²)", Background → RGBColor[0.6, 0, 0.6]]
```

Ridisegno il grafico in modo che la larghezza sia 3 volte l'altezza

```
Show[grafico, AspectRatio → 1 / 3]
```

Ridisegno il grafico visualizzando sull'asse x solo i multipli di π

```
Show[grafico, Ticks → {{0, Pi, 2 Pi, 3 Pi}, {}}]
```

Show[{plot1,plot2,...}] disegna due o piu' grafici contemporaneamente. Nel seguente esempio *Mathematica* crea i grafici plot1 e plot2 ma non li visualizza. Successivamente li visualizza insieme con il comando **Show**

```
plot1 = Plot[Sin[x], {x, -Pi / 2, Pi / 2}, DisplayFunction → Identity]
plot2 = Plot[ArcSin[x], {x, -1, 1}, DisplayFunction → Identity]
Show[{plot1, plot2}, DisplayFunction → $DisplayFunction]
```

Possiamo usare **Show** per disegnare grafici di funzioni in intervalli distinti .

```
plot3 = Plot[x^2, {x, 0, 1}, DisplayFunction → Identity]
plot4 = Plot[2 - x^2, {x, 1, 2}, DisplayFunction → Identity]
Show[{plot3, plot4}, DisplayFunction → $DisplayFunction]
```

Show[GraphicsArray[{g1,...,gn}]] permette di visualizzare diversi grafici in un' unica riga. Per esempio

```
g1 = Plot[Tan[x], {x, 0, 2 Pi}, PlotRange -> {-2 Pi, 2 Pi}]
g2 = Plot[Sin[x + x^2], {x, -2 Pi, 2 Pi}]

Show[GraphicsArray[{g1, g2}]]
Show[GraphicsArray[{{g1}, {g2}}]]
```

Piu' in generale

```
Show[GraphicsArray[{{g1, g2}, {plot1, plot2}}]]
```

2.3.2 Direttive grafiche

Con il comando **PlotStyle->{style1,...,style2}** e' possibile specificare le *direttive grafiche (graphics directives)* che riguardano il modo in cui disegnamo il grafico come per esempio, lo spessore, il colore e il tipo di linea che disegna il grafico. Per far questo usiamo, per esempio, le seguenti *direttive grafiche*

• **Thickness[r]** con 0<r<1 che disegna le linee di spessore r volte la larghezza dell'intero grafico;

• **Dashing[{r1,...,rn}]** rappresenta le linee come una sequenza di segmenti di lunghezza r1,..,rn. Questa sequenza e' ripetuta le volte necessarie per disegnare il grafico;

• **AbsoluteThickness[w]** che disegna le linee di spessore *w*, con *w* misurata con un'unita' di misura assoluta (per *default* w=1 che corrispone a 1/72 inch);

• **AbsoluteDashing[{w1,...,wn}]** rappresenta le linee come una sequenza di segmenti di lunghezza w1,..,wn. Questa sequenza e' ripetuta le volte necessarie per disegnare il grafico;

• **GrayLevel[i]** disegna le linee con una tonalita' di grigio tra bianco (1) e nero (0);

•**RGBColor[r,g,b]** disegna le linee colorate con componenti red, green e blu specificate.

```
Plot[Sin[x], {x, 0, 2 Pi}, PlotStyle → {Thickness[.010], RGBColor[0, 1, 0]}]

Plot[Cos[x], {x, 0, 2 Pi}, PlotStyle → {GrayLevel[.5], Dashing[{.03, .01}]}]

Plot[{Sin[x], Cos[x], Sinh[x]}, {x, 0, Pi},
 PlotStyle → {{RGBColor[1, 0, 1], Thickness[.025]}, {RGBColor[1, 1, 0],
    Thickness[.015]}, {RGBColor[0, 1, 1], Thickness[.005]}}]
```

Le *direttive grafiche* - cioe' **PlotStyle->Options** - non si possono cambiare in **Show** perche' il grafico e' stato gia' disegnato!

```
grafico = Plot[Exp[-x^2], {x, -1, 1}]
```

Pero' *Mathematica* "protesta" se gli chiedo di disegnare il grafico in rosso, cioe' modificando le *direttive grafiche*

```
Show[grafico, PlotStyle → RGBColor[1, 0, 0]]
```

Esempio. Si consideri il grafico

```
Plot[{Cosh[x], Sinh[x]}, {x, -1, 1}]
```

Ridisegnare il grafico colorando le curve una rossa e una verde, con il grafico contenuto in un quadrato con una cornice.

```
Plot[{Cosh[x], Sinh[x]}, {x, -1, 1}, AspectRatio → 1, Frame → True,
 PlotStyle -> {{RGBColor[1, 0, 0]}, {RGBColor[0, 1, 0]}}]
```

Mathematica e' anche in grado di disegnare il grafico di funzioni definite in forma implicita con il comando **ImplicitPlot[eq==0,{x,a,b}]**. La variabile **x** e' associata all'asse orizzontale. Dobbiamo inizialmente "istruire" *Mathematica* caricando il *package*

```
Needs["Graphics`ImplicitPlot`"]
```

Il comando

```
Names["Graphics`ImplicitPlot`*"]
```

mi ricorda le nuove funzioni caricate dal *package*.

```
ImplicitPlot[x^2 + 4 y^2 == 1, {x, -2, 2}, AspectRatio → Automatic]
```

Esempio. Scrivere in tre modi diversi una funzione che disegna la circonferenza di centro (x_0, y_0) e raggio R>0

1) considero la rappresentazione parametrica della circonferenza

```
f1[{x0_, y0_}, R_ /; R > 0] := ParametricPlot[
  {x0 + R * Cos[t], y0 + R * Sin[t]}, {t, 0, 2 * Pi}, AspectRatio → Automatic]

f1[{1, 1}, 2]
```

2) disegno il grafico della curva in forma implicita

```
f2[{x0_, y0_}, R_ /; R > 0] :=
 ImplicitPlot[(x - x0)^2 + (y - y0)^2 == R^2, {x, x0 - R, x0 + R}]

f2[{1, 1}, 2]
```

3) disegno i grafici delle due curve $y=y0\pm\sqrt{R^2-(x-x0)^2}$ curva in forma implicita

```
f3[{x0_, y0_}, R_ /; R > 0] :=
 Plot[{y0 - Sqrt[R^2 - (x - x0)^2], y0 + Sqrt[R^2 - (x - x0)^2]},
  {x, x0 - R, x0 + R}, AspectRatio → 1]

f3[{1, 1}, 2]
```

Nel disegnare grafici bidimensionali puo' essere utile conoscere il *package* **Graphics`FilledPlot`** che evidenzia la regione di piano compresa tra una funzione e l'asse x o tra due o piu' funzioni.

```
Needs["Graphics`FilledPlot`"]
Names["Graphics`FilledPlot`*"]

FilledPlot[Cos[x], {x, 0, Pi}]

FilledPlot[{Cos[x], Sin[x]^2, Sin[x]}, {x, 0, Pi}]
```

Con l'opzione **Fills→Options** e' possibile specificare il colore della regione da disegnare.

```
FilledPlot[Cos[x], {x, 0, Pi}, Fills → RGBColor[1, 0, 0]]

FilledPlot[{Cos[x], Sin[x]^2}, {x, 0, Pi}, Fills → RGBColor[0, 0, 1]]

FilledPlot[{Cos[x], Sin[x]^2, Sin[x]},
 {x, 0, Pi}, Fills → {RGBColor[1, 0, 0], GrayLevel[.7]}]
```

Torneremo piu' avanti a parlare di grafici.

Esercizi

1) Disegnare la retta passante per in punti (1,2) e (3,5). Specificare due opzioni grafiche a scelta.

2) Disegnare la funzione f(x)=cos(x)+cos(3 x) nell'intervallo [-4 π, 4 π].

a) Modificare **AspectRatio** affinche' l'altezza del grafico sia due volte la larghezza.

b) Con il comando **Ticks** far comparire sull'asse delle x solo i multipli pari di Pi.

c) Con il comando **PlotLabel** dare un nome al disegno.

3) Sia f(x)=x^2. Disegnare f(x) e f(x+1) in (-3,3). Tracciare il grafico di f(x) piu' spesso di quello di f(x+1).

4) Definire la funzione f(x)=xsin(1/x), f(0)=0 e disegnarla in (-1/2,1/2), con una linea tratteggiata.

5) Disegnare in un array le tre parabole $y = ax^2 + bx + c$ con $(a, b, c) = (1, 3, 3),\ (2, 1, 0)\ e\ (3, 0, 1)$, specificando per ciascuna una direttiva grafica.

■ 2.4 Successioni

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

Definire una successione e' esattamente l'analogo che definire una funzione

```
a[i_] := 1 / i^2
```

Possiamo specificare che **i** sia intero o intero non negativo.

```
b[i_Integer] := 1 / i^2
c[i_Integer /; NonNegative[i]] := 1 / i^2
```

Oppure possiamo definire la successione $a_n = n^2$, n naturale, usando la funzione **IntegerQ[n]**

```
a[n_ /; (IntegerQ[n] && n > 0)] := n^2
```

Esempio 1. Vediamo come operano in maniera differente i comandi **FullSimplify** e **Simplify** sulla successione $\frac{k^k}{k!}$

```
Clear[a]
a[k_] := k^k / k!

Simplify[a[k + 1] / a[k]]
FullSimplify[a[k + 1] / a[k]]
```

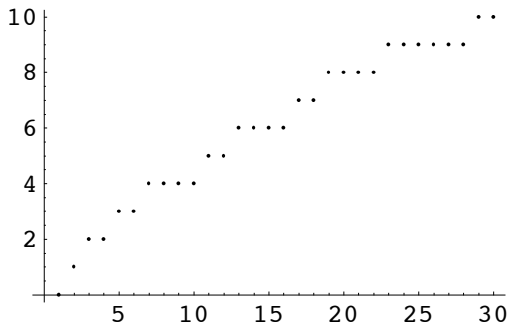
• Se vogliamo generare i primi termini di una successione usiamo il comando **Table[a[i],{i,1,n}]** che crea una lista costituita da n elementi della successione (oppure **Table[a[i],{i,1,n,s]** incrementa l'indice di s)

```
Clear[a]
Table[a[i], {i, 1, 10}]
Table[a[i], {i, 1, 10, 2}]
Table[a_i, {i, 1, 10}]
```

Se vogliamo rappresentare graficamente i termini di una successione usiamo il comando **ListPlot[{a_1 ,a_2 ,...,a_n}]** : questo comando serve per rappresentare la successione di punti del piano di coordinate (1,a_1),{2,a_2),...,(n,a_n).

Nella tabella che segue usiamo la funzione **PrimePi[x]** che fornisce il numero primo minore o uguale di x

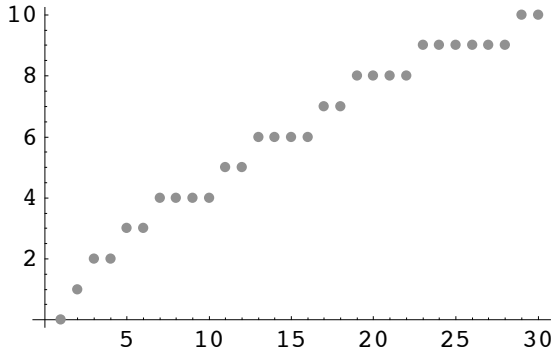
```
ll = Table[PrimePi[n], {n, 1, 30}]
ListPlot[ll]
```




```
- Graphics -
```

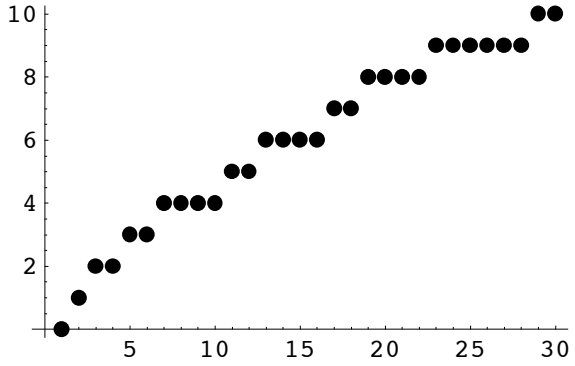
L'opzione **PlotStyle** puo' anche essere usata in **ListPlot**. Se voglio rappresentare i punti di dimensione diversa posso usare il comando **PlotStyle→PointSize[d]**, con d compreso tra 0 e 1, che permette di disegnare i punti di dimensione "d volte" la larghezza dell'intero grafico. Per *default* d=*0.008*. Si puo' anche usare **PlotStyle→AbsolutePointSize[d]** che rappresenta i punti di diametro d misurati in una unita' di misura assoluta (per *default* d=1/72 inch).

```
g1 = ListPlot[ll, PlotStyle → {GrayLevel[.5], PointSize[0.02]}]
```




```
- Graphics -
```

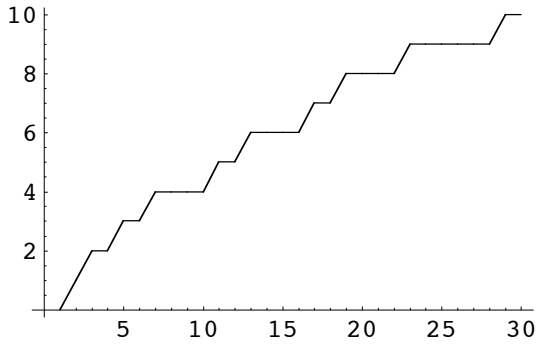
```
ListPlot[ll, PlotStyle → AbsolutePointSize[6]]
```




```
- Graphics -
```

Se voglio unire i punti della lista usero' il comando **PlotJoined→True**

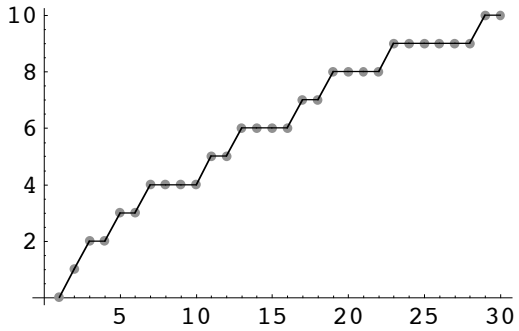
```
g2 = ListPlot[l1, PlotJoined → True]
```




```
- Graphics -
```

Possiamo anche sovrapporre i due grafici precedenti, sostanzialmente ottenendo l'effetto di aggiungere punti alla spezzata. Per far questo uso il comando **Show[g1,g2,...]** che mostra piu' grafici contemporaneamente

```
Show[g1, g2]
```



Esempio. Ora generiamo la successione il cui elemento n e' l'ennesimo numero primo

```
l1 = Table[Prime[n], {n, 100}];
ListPlot[l1]
```

E' possibile usare il comando **ListPlot** anche su una lista di coppie che vengono interpretate come {ascissa,ordinata}

```
l2 = Table[{Cos[t], Sin[t]}, {t, 0, 2 Pi, Pi / 20}];
```

In questo modo otteniamo 40 punti che si collocano sulla circonferenza di centro l'origine e raggio 1. Rappresentiamoli graficamente

```
ListPlot[l2, PlotStyle → AbsolutePointSize[4], AspectRatio → Automatic]
```

Il *package* Graphics`MultipleListPlot` consente di rappresentare due o piu' liste contemporaneamente.

```
<< Graphics`MultipleListPlot`

l1 = Table[{1 / n, n * N[Sin[n]]}, {n, 1, 10}];
l2 = N[Table[{t, Log[t]}, {t, 1, 5}]];

MultipleListPlot[l1, l2]
```

• E' possibile definire *successioni per ricorrenza*. Per esempio

```
Clear[a]
a[0] = 1; a[n_] := n^2 a[n - 1] /; n ≥ 1
Table[a[n], {n, 0, 10}]
```

Esempio 2. Un classico esempio di succesione definita per ricorrenza e' *la successione dei numeri di Fibonacci*, definita dalla regola f(0)=0,f(1)= 1, f(n)=f(n-2)+f(n-1) se n>1.

```
Clear[f]
f[0] = 0; f[1] = 1; f[n_Integer /; (n > 1)] := f[n - 2] + f[n - 1]

Table[f[i], {i, 0, 10}]

Timing[f[30]]
```

Mathematica ci mette molto tempo (dipende ovviamente dal computer sul quale state lavorando) a calcolare f[30] perche' deve calcolare f[i] i<30 molte volte. Si puo' accellerare il procedimento, memorizzando i numeri calcolati (cioe' guadagno in velocita' ma perdo in memoria!). Per fare questo combino i comandi **Set(=)** e **SetDelayed(:=)** nel seguente modo

```
ff[0] = 0; ff[1] = 1;
ff[n_Integer /; (n > 1)] := ff[n] = ff[n - 1] + ff[n - 2]
```

cosi' se chiedo a *Mathematica* di calcolare **ff[n]**, *Mathematica* memorizza di volta in volta i valori ff[i] ,i<n, che quindi vengono calcolati una sola volta.

Esercizio 1. Provate a disegnare i primi 15 numeri della successione di Fibonacci, scegliendo la dimensione dei punti e successivamente unendo tra loro i punti (in realta' *Mathematica* conosce la successione dei numeri di Fibonacci)

```
?Fibonacci
```

Esercizio 2. Definite per ricorrenza le seguenti due successioni i cui primi termini sono dati, rispettivamente, da {1,2,4,8,16,32,64,128...} e {2,3,6,18,108,1944,209952,...}. Scrivete una funzione di *Mathematica* per calcolare il generico valore ennesimo.

Esempio 3. Costruiamo la successione

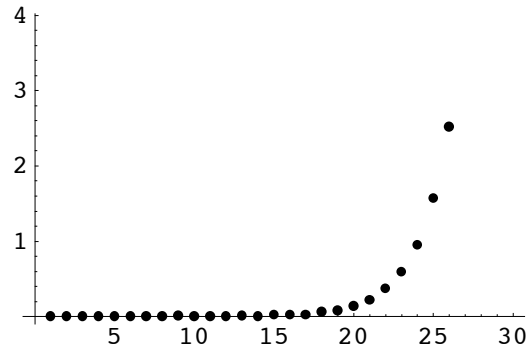
```
a[j_] := First[Timing[f[j]]]
```

che rappresenta il tempo impiegato per calcolare i numeri di Fibonacci nella posizione j, secondo la prima definizione. Costruiamo la lista costituita dai primi 30 termini della successione

```
tempi=Table[a[j],{j,1,30}]
```

Il disegno dei dati cosi ottenuti mostra la crescita esponenziale del tempo impiegato per il calcolo

```
ListPlot[tempi/Second,PlotStyle->PointSize[.02]]
```




```
- Graphics -
```

Confrontate con i tempi impiegati usando la seconda definizione cioe' **ff**.

Esempio 4. Disegnamo i primi 20 punti della successione $a_n = (-1)^{n/2}\ \frac{n}{n+1}$ n pari; $a_n = \frac{n}{n+1}$ n dispari .

```
Clear[a]
a[n_ /; EvenQ[n]] := (-1)^(n / 2) n / (n + 1)
a[n_ /; OddQ[n]] := n / (n + 1)
ListPlot[Table[a[n], {n, 1, 20}]]
```

■ 2.5 Somme e Serie

Per sommare i termini di una successione e' disponibile il comando **Sum[f[i],{i,imin,imax}]** (se non si specifica il limite inferiore *Mathematica* lo assume uguale ad 1) mentre il comando **Sum[f[i],{i,imin,imax,s}]** calcola la somma incrementando l'indice i di s.

```
a[i_] := 1 / i!; Sum[a[i], {i, 1, 10}]
Sum[x^i / i, {i, 6}]
Sum[x^i / i, {i, 1, 6, 2}]
```

In qualche caso il comando **Sum** riesce a calcolare la somma dei termini della successione senza conoscere a propri il numero di termini, per esempio

```
Sum[q^i, {i, 0, n}](*somma geometrica*)
Sum[k, {k, 1, n}](*somma aritmetica*)
```

oppure

```
Sum[k^3, {k, 1, n}]
Sum[k^4, {k, 1, n}]
Sum[k^a, {k, 1, n}]
```

Mathematica e' anche in grado, talvolta, di calcolare la somma di una serie. Per esempio

```
Sum[1 / a^n, {n, 1, Infinity}](*serie geometrica*)
Sum[1 / k^2, {k, 1, Infinity}](*serie armonica*)
Sum[1 / k^a, {k, 1, Infinity}](*serie armonica generalizzata*)
Sum[x^k / k!, {k, 0, Infinity}](*serie esponenziale*)
```

ma se la serie e' complicata *Mathematica* non riesce a calcolarla esattamente

```
Sum[1 / (i! + (2 i)!), {i, 1, Infinity}]
```

Si puo' comunque trovare un'approssimazione numerica del risultato

```
N[%]
```

oppure usare direttamente il comando **NSum[f,{i,imin,imax}].**

Mathematica si accorge che la serie armonica diverge

```
Sum[1 / k, {k, 1, Infinity}]
```

Esempio 4. Dimostrate, usando *Mathematica*, che la serie il cui termine generico e' $a_n = \frac{(n+1)^2}{5^n + n}$ converge.

```
a[n_] := (n + 1)^2 / (5^n + n)
```

E' verificata la C.N. per la convergenza della serie cioe'

```
Limit[a[n], n → Infinity](*calcolo il limite della successione*)
```

Applico il criterio del rapporto

```
l = Limit[a[n + 1] / a[n], n → Infinity]
```

dato che l<1 la serie converge. Quale sara', approssimativamente, la somma della serie?

```
NSum[a[n], {n, 1, Infinity}]
```

Esempio 5. Dimostrate, usando *Mathematica*, che la serie il cui termine generico e' $a_n = \pi^n / (n^n \log(n+1))$ converge.

```
a[n_] := (Pi)^n / ((n^n) * Log[n + 1])
```

E' verificata la C.N. per la convergenza della serie cioe'

```
Limit[a[n], n → Infinity]
```

Applico il criterio della radice

```
l = Limit[Sqrt[a[n]], n → Infinity]
```

dato che l<1 la serie converge. Quale sara', approssimativamente, la somma della serie?

```
NSum[a[n], {n, 1, Infinity}]
```

Puo' essere utile sapere che il comando **Series[f,{x,x0,n}]** trova lo sviluppo in serie di potenze di *f* intorno al punto x0, fino al termine (x-x0)^n

```
Series[f[x], {x, x0, 4}]
```

Nell'espressione del resto compare, al posto del simbolo o piccolo il simbolo O grande. Quindi O[x]^5 altro non e' che o[x]^4.

```
Series[Exp[x], {x, 0, 4}]
f[x_] := 1 / (1 + x^2); Series[f[x], {x, 0, 5}]
```

Se si vuole solo il polinomio di Taylor, senza il resto, si usa il comando **Normal,** che tronca lo sviluppo

```
polinomio = Normal[Series[f[x], {x, 0, 5}]]

Plot[{polinomio, f[x]}, {x, -1, 1},
 PlotStyle → {{Thickness[0.01]}, {RGBColor[1, 0, 0]}}]
```

Esercizio. Disegnate la funzione x+sin(x) nell'intervallo $(-2\pi, 2\pi)$ insieme al suo polinomio di Taylor di ordine 5 centrato nell'origine.

Esercizio. Sia $f(x)=\sum_{n=0}^{\infty} (-10)^n \, x^{2n+2} / (2^{2n+2} \, n! \, (n+3)!)$. Disegnare f(x) in [-1,1].

Esercizio. Determinare il polinomio di Taylor di $f(x) = \sqrt{\cos(x)}$ di centro 0 e ordine 4. Disegnate

$f(x)$ e il polinomio nell'intervallo $(-\pi/2,\pi/2)$.