Guida a *Mathematica*

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

Come è strutturato *Mathematica*

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

Sintassi

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

```
Sin[x]
```

```
Sin[x]
```

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

```
Out[1]
```

```
{1, 4, 9, 16, 25, 36, 49, 64, 81, 100}
```

Oppure:

```
%%
```

```
Sin[x]
```

La sintassi per le righe di commento è:

```
(*commento*)
```

Operatori aritmetici

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

```
a b
```

```
a b
```

è equivalente a:

```
a * b
```

```
a b
```

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

```
4 Sin[x]
```

```
4 Sin[x]
```

è equivalente a:

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

```
4 Sin[x]
```

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

Costanti built-in

Le principali costanti built-in sono:

```
Infinity
```

```
∞
```

```
Pi
```

```
π
```

```
E
```

```
ⅇ
```

```
I
```

```
ⅈ
```

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

```
N[π]
```

```
3.14159
```

Operatori booleani

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

Funzioni built-in

Passiamo in rassegna alcune funzioni built-in:

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

```
Log[x]
```

```
Log[x]
```

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

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

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

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

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

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

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

```
Sin[x];
```

```
Sin[x]
```

```
Sin[x]
```

Funzioni definite dall'utente

Assegnazione immediata

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

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

```
expr
```

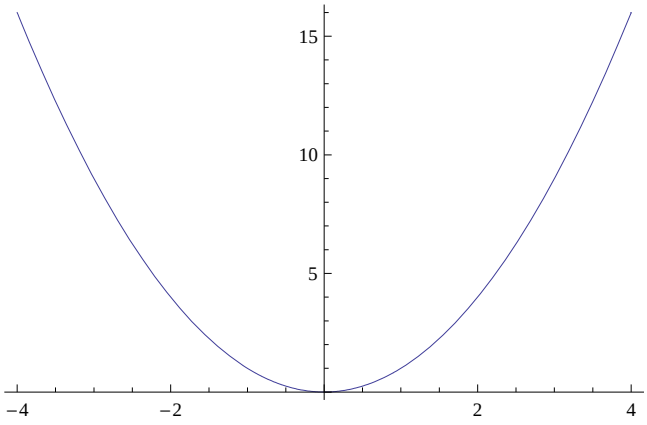
dove <expr> è l'espressione analitica della funzione. Ad esempio:

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

x^2

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

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



Assegnazione ritardata

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

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

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

```
Clear[f]
```

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

```
f[x]
```

x^2

```
Clear[f]
```

Assegnazione condizionata

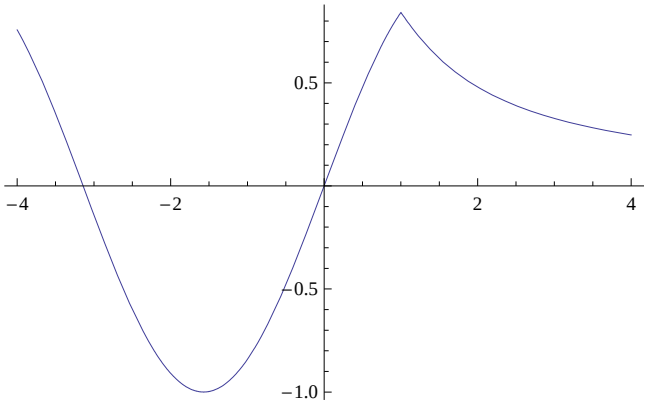
L'*assegnazione condizionata* si utilizza quando dobbiamo definire una funzione che ha espressioni diverse per differenti sottoinsiemi dell'insieme di definizione. Ad esempio, consideriamo la funzione reale di una variabile reale: $f(x) = \sin(x)$, se $x \in (-\infty,1]$; $f(x) = \sin\left(\frac{1}{x}\right)$,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

```
f[x_] := Sin[x] /; x ≤ 1
```

$$f[x\_] := \text{Sin}\left[\frac{1}{x}\right] \;/;\; x > 1$$

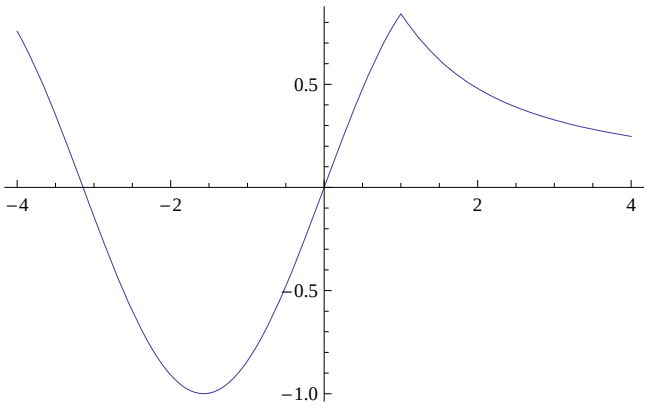
Grafichiamo f

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



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

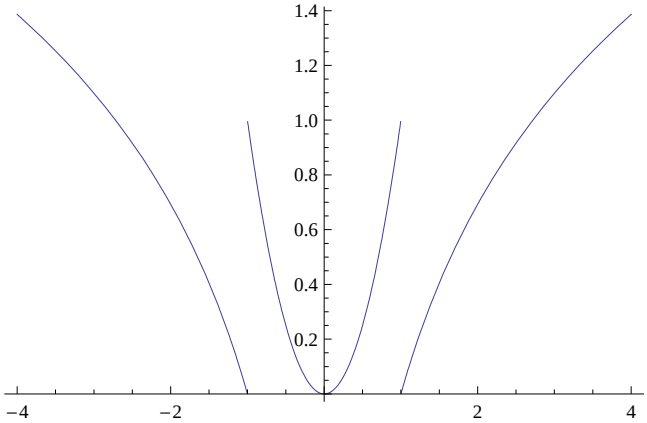
```
Clear[f]

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

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



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

```
Clear[f]

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

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

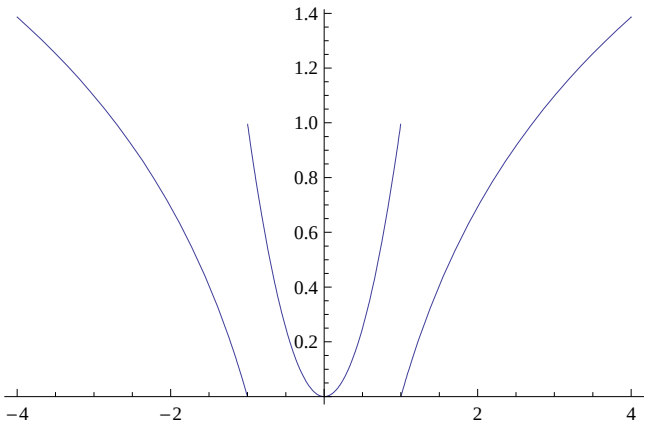



```
Clear[f]
```

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

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

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



```
Clear[f]
```

Definizione di funzioni composte

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

```
f[x_] := x^5 + 6 x^4 - 2 x + Log[x^2 + 1]
ReplaceAll[f[x], x → Sin[t]]
```

$\text{Log}[1 + \text{Sin}[t]^2] - 2\,\text{Sin}[t] + 6\,\text{Sin}[t]^4 + \text{Sin}[t]^5$

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

```
f[x] /. x → Sin[t]
```

$\text{Log}[1 + \text{Sin}[t]^2] - 2\,\text{Sin}[t] + 6\,\text{Sin}[t]^4 + \text{Sin}[t]^5$

```
Clear[f]
```

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

$$f[x\_, y\_] := \sqrt{x^2 - y^2}$$

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

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

$\sqrt{\text{Cos}[t]^2 - \text{Sin}[t]^2}$

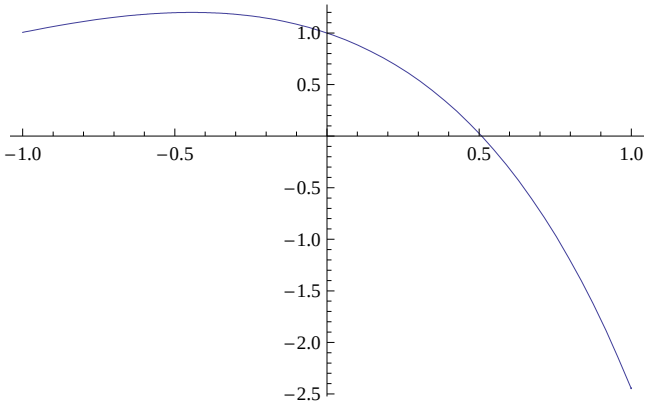
L'istruzione **/.** è estremamente efficace nel caso di plotting di integrali di equazioni differenziali. Vedremo più avanti che la soluzioni di equazioni differenziali (ODE) si effettua con l'istruzione **DSolve**. Supponiamo di avere l'equazione differenziale del secondo ordine: $y'' - 2y' + y = \sin(x)$.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

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

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

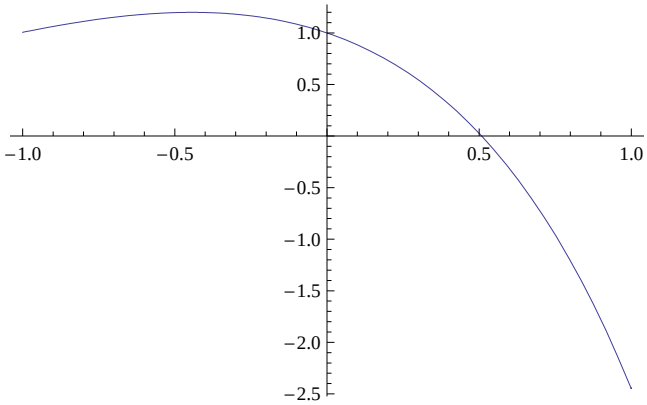
Grafichiamo tale integrale particolare:

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



Utilizzando **ReplaceAll**:

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



L'istruzione Evaluate

Nel calcolo delle derivate è necessario definire la derivata con l'assegnazione immediata. Nel caso contrario, *Mathematica* resituisce un messaggio di errore, poichè cerca di calcolare prima la funzione in un punto *x* per poi calcolare la derivata. Ad esempio:

```
Clear[f]
```

```
f[x_] := x^10
```

La derivata si calcola con l'istruzione **D[f[x],x]**

```
Df[x_] := D[f[x], x]
```

```
Df[2]
```

General::ivar : 2 is not a valid variable. ≫

$\partial_2 1024$

Idem se plottiamo:

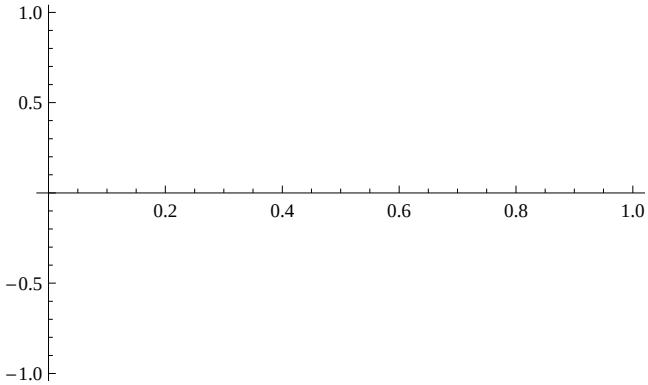
```
Plot[
 Df[x],
 {x, 0, 1}
]
```

General::ivar : 0.000020428571428571428` is not a valid variable. ≫

General::ivar : 0.02042859183673469` is not a valid variable. ≫

General::ivar : 0.04083675510204081` is not a valid variable. ≫

General::stop : Further output of General::ivar will be suppressed during this calculation. ≫



Allora proviamo a forzare il kernel con l'istruzione **Evaluate**:

```
Clear[Df]
```

```
Df[x_] := Evaluate[D[f[x], x]]
```

```
Df[x]
```

$10\,x^9$

Notiamo di passaggio che tutti i comandi che richiedono un solo argomento, possono essere richiamati in notazione postfissa utilizzando **//**. Ad esempio:

```
D[f[x], x] // Evaluate
```

$10\,x^9$

In alternativa, avremmo potuto utilizzare l'assegnazione immediata:

```
Clear[Df]
```

```
Df[x_] = D[f[x], x]
```

$10\,x^9$

Infine, è possibile utilizzare l'istruzione **Derivative** che in forma abbreviata è data dalla classica notazione apicale di Lagrange:

```
f'[x]
```

$10\,x^9$

E quindi le derivate di ordine superiore

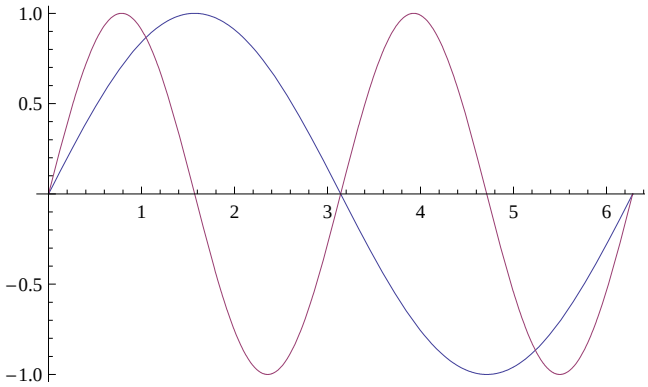
```
f'''[x]
```

$720\,x^7$

Grafici multipli

Anche l'argomento grafici verrà trattato più avanti, vale la pena fare alcune osservazioni, premettendo il comando **Table**, quale generatore di liste.

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



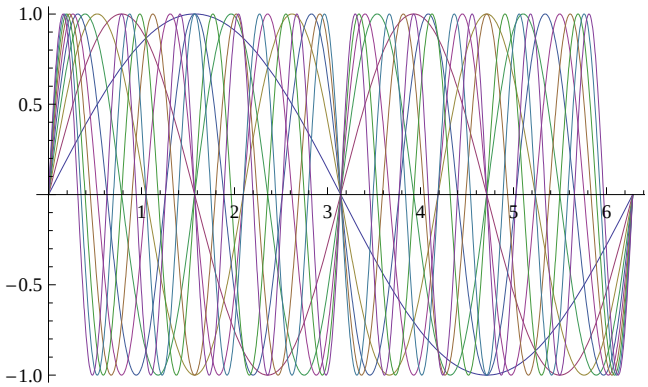
In questo caso abbiamo introdotto le singole funzioni nell'istruzione **Plot**. Diversamente, immaginiamo di avere le funzioni $\sin(k\,x)$, dove $k = 1, 2, \ldots 10$. Scriviamo la lista :

```
funzioni = Table[
  (*termine generale*)
  Sin[k * x],
  (*iteratore. Omettiamo k=1 poichè definito per default*)
  {k, 10}
 ]
```

```
{Sin[x], Sin[2 x], Sin[3 x], Sin[4 x],
 Sin[5 x], Sin[6 x], Sin[7 x], Sin[8 x], Sin[9 x], Sin[10 x]}
```

Se grafichiamo:

```
Plot[
 funzioni,
 {x, 0, 2 π}
]
```



Numeri interi, razionali, reali, complessi

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

```
2.0
```

```
2.
```

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

```
2
```

```
2
```

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

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

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

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

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

```
7. + 2. ⅈ
```

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

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

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

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

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

```
Head[√3.]
```

```
Real
```

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

```
Head[x]
```

```
Symbol
```

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

```
NumberQ[√2.]
```

```
True
```

```
NumberQ[√2]
```

```
False
```

In questo caso, restituisce **False**, poichè *Mathematica* interpreta $\sqrt{2}$ alla stregua di un simbolo e non di un numero. In

maniera simile, agisce **IntegerQ[x]** che controlla se **x** è un intero o meno.

```
IntegerQ[4]
```

```
True
```

```
IntegerQ[-4]
```

```
True
```

```
IntegerQ[4.]
```

```
False
```

Riconoscimento della parità di un intero naturale

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

```
EvenQ[2]
```

```
True
```

```
EvenQ[3]
```

```
False
```

```
OddQ[2]
```

```
False
```

Riconoscimento di numeri primi

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

```
PrimeQ[11 003]
```

```
True
```

```
PrimeQ[11 009]
```

```
False
```

Approssimazioni numeriche

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

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

```
$MachinePrecision
```

```
15.9546
```

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

```
√2.
```

```
1.41421
```

```
SetPrecision[√2., 4]
```

```
1.414
```

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

```
a = 1.49;
```

```
{Floor[a], Ceiling[a], Round[a]}
```

```
{1, 2, 1}
```

```
b = 1.5;
```

```
{Floor[b], Ceiling[b], Round[b]}
```

```
{1, 2, 2}
```

Formato dei numeri

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

```
ScientificForm[21.121]
```

2.1121×10^{1}

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

```
EngineeringForm[81 200.00141]
```

81.2×10^{3}

Approssimazione dei numeri reali con numeri razionali

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

```
Rationalize[√3, 10^-3]
```

$\frac{71}{41}$

Se richiediamo la rappresentazione decimale, dobbiamo applicare la funzione **N[]**; ricordando che ogni funzione richiedente un solo argomento può essere richiamata in notazione postfissa, si ha:

```
Rationalize[√3, 10^-3] // N
```

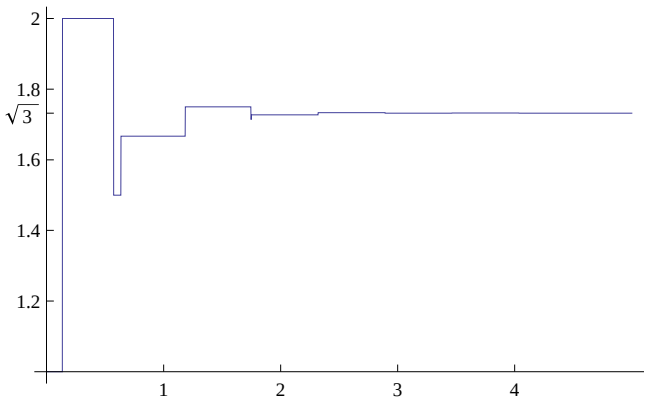
1.73171

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

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

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

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



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

```
x = 5 - 4.999999999999
```

1.00009×10^{-12}

```
Chop[x]
```

0

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

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

1.00009×10^{-12}

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

Liste

Costruzioni di liste

Una lista è un insieme di "oggetti". Ad esempio:

```
lista = {-4, 5, a, x}
```

$\{-4, 5, 1.49, 1.00009 \times 10^{-12}\}$

È possibile eseguire una qualunque operazione sull'oggetto **lista**

```
Log[lista]
```

$\{i\,\pi + \text{Log}[4], \text{Log}[5], 0.398776, -27.6309\}$

y^{lista}

$\left\{\frac{1}{y^4}, y^5, y^{1.49}, y^{1.00009\times 10^{-12}}\right\}$

L'oggetto "lista" ha l'attributo **Listable**, nel senso che una qualunque funzione applicata a una lista, restituisce una nuova lista i cui elementi sono il risultato dell'applicazione della funzione ai singoli elementi di **lista**. Ad esempio:

```
f[x_] := Sin[x + ArcTan[x]] - x^5 + 5 x^4 + x^3 + 1
f[lista]
```

$\{2241 - \text{Sin}[4 + \text{ArcTan}[4]], 126 + \text{Sin}[5 + \text{ArcTan}[5]], 22.2307, 1.\}$

Possiamo generare una lista di liste:

```
lista2 = {lista, {lista, lista}}
```

$\{\{-4, 5, 1.49, 1.00009 \times 10^{-12}\}, \{\{-4, 5, 1.49, 1.00009 \times 10^{-12}\}, \{-4, 5, 1.49, 1.00009 \times 10^{-12}\}\}\}$

Table

Abbiamo visto in precedenza che un potente generatore di liste è **Table**, la cui sintassi (esempio) è:

```
Table[
 k^2,
 {k, 10}
]
```

$\{1, 4, 9, 16, 25, 36, 49, 64, 81, 100\}$

Qui abbiamo denotato con **k** la variabile di iterazione. Ma è chiaro che avremmo potuto utilizzare un qualunque altro simbolo. Ad esempio:

```
Table[
 x^2,
 {x, 10}
]
```

$\{1, 4, 9, 16, 25, 36, 49, 64, 81, 100\}$

anche se è più adatto il simbolo **k**, solitamente utilizzato per indicare una variabile discreta. Negli esempi visti, la generazione della lista parte con **k=1**, che il valore di default. Naturalmente possiamo definirne un altro:

```
Table[
 k^2,
 {k, 0, 10}
]
```

{0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100}

Possiamo poi modificare il passo di iterazione:

```
Table[
 k^2,
 {k, 0, 10, 1/10}
]
```

$$\left\{0, \frac{1}{100}, \frac{1}{25}, \frac{9}{100}, \frac{4}{25}, \frac{1}{4}, \frac{9}{25}, \frac{49}{100}, \frac{16}{25}, \frac{81}{100}, 1, \frac{121}{100}, \frac{36}{25}, \frac{169}{100}, \frac{49}{25}, \frac{9}{4}, \frac{64}{25}, \frac{289}{100}, \frac{81}{25}, \frac{361}{100}, 4, \frac{441}{100}, \frac{121}{25}, \frac{529}{100}, \frac{144}{25}, \frac{25}{4}, \frac{169}{25}, \frac{729}{100}, \frac{196}{25}, \frac{841}{100}, 9, \frac{961}{100}, \frac{256}{25}, \frac{1089}{100}, \frac{289}{25}, \frac{49}{4}, \frac{324}{25}, \frac{1369}{100}, \frac{361}{25}, \frac{1521}{100}, 16, \frac{1681}{100}, \frac{441}{25}, \frac{1849}{100}, \frac{484}{25}, \frac{81}{4}, \frac{529}{25}, \frac{2209}{100}, \frac{576}{25}, \frac{2401}{100}, 25, \frac{2601}{100}, \frac{676}{25}, \frac{2809}{100}, \frac{729}{25}, \frac{121}{4}, \frac{784}{25}, \frac{3249}{100}, \frac{841}{25}, \frac{3481}{100}, 36, \frac{3721}{100}, \frac{961}{25}, \frac{3969}{100}, \frac{1024}{25}, \frac{169}{4}, \frac{1089}{25}, \frac{4489}{100}, \frac{1156}{25}, \frac{4761}{100}, 49, \frac{5041}{100}, \frac{1296}{25}, \frac{5329}{100}, \frac{1369}{25}, \frac{225}{4}, \frac{1444}{25}, \frac{5929}{100}, \frac{1521}{25}, \frac{6241}{100}, 64, \frac{6561}{100}, \frac{1681}{25}, \frac{6889}{100}, \frac{1764}{25}, \frac{289}{4}, \frac{1849}{25}, \frac{7569}{100}, \frac{1936}{25}, \frac{7921}{100}, 81, \frac{8281}{100}, \frac{2116}{25}, \frac{8649}{100}, \frac{2209}{25}, \frac{361}{4}, \frac{2304}{25}, \frac{9409}{100}, \frac{2401}{25}, \frac{9801}{100}, 100\right\}$$

■ Range

L'istruzione **Range[k]** genera la lista **{1,2,3...,k}**. Ad esempio:

```
Range[9]
```

{1, 2, 3, 4, 5, 6, 7, 8, 9}

Per **x** reale, **Range[x]** restituisce {1, 2, 3, ...[*x*]}:

```
Range[8.5]
```

{1, 2, 3, 4, 5, 6, 7, 8}

Anche qui è possibile definire un valore di partenza e un passo di iterazione:

```
Range[0, 10, 1/2]
```

$$\left\{0, \frac{1}{2}, 1, \frac{3}{2}, 2, \frac{5}{2}, 3, \frac{7}{2}, 4, \frac{9}{2}, 5, \frac{11}{2}, 6, \frac{13}{2}, 7, \frac{15}{2}, 8, \frac{17}{2}, 9, \frac{19}{2}, 10\right\}$$

```
Range[x, x + 10, 2]
```

$$\{1.00009 \times 10^{-12}, 2., 4., 6., 8., 10.\}$$

Array

L'istruzione **Array[f,n]** genera una lista di *n* elementi :

```
Clear[f]
```

```
Array[f, 4]
```

```
{f[1], f[2], f[3], f[4]}
```

Operazioni sulle liste e manipolazioni di dati

Union

Alcuni comandi operano alla stregua delle operazioni di inclusione e di intersezione della teoria degli insiemi. Per fissare le idee, consideriamo l'esempio seguente. Siano dati gli insiemi (liste):

$$\mathbf{S1} = \left\{x^4, a, e^{\pi}, \sqrt{2}, \mathrm{Sin}[\pi^2]\right\}; \mathbf{S2} = \left\{b, d, a, 10^n, x^4, \mathrm{Sin}[\pi^2], \sqrt{5}\right\};$$

L'unione di tali insiemi (cioè $S_1 \bigcup S_2$) è:

```
Union[S1, S2]
```

$$\left\{\sqrt{2}, \sqrt{5}, 10^n, a, b, d, e^{\pi}, x^4, \mathrm{Sin}[\pi^2]\right\}$$

La funzione **Join** agisce in maniera simile, ma non elimina gli elementi ripetuti:

Join

```
Join[S1, S2]
```

$$\left\{x^4, a, e^{\pi}, \sqrt{2}, \mathrm{Sin}[\pi^2], b, d, a, 10^n, x^4, \mathrm{Sin}[\pi^2], \sqrt{5}\right\}$$

Tuttavia, l'applicazione della funzione **Union** elimina gli elementi ripetuti:

```
Union[Join[S1, S2]]
```

$$\left\{\sqrt{2}, \sqrt{5}, 10^n, a, b, d, e^{\pi}, x^4, \mathrm{Sin}[\pi^2]\right\}$$

Intersection

L'intersezione degli insiemi assegnati è $S_1 \bigcap S_2 = \left\{x^4, \sin(\pi^2)\right\}$. L'operazione di intersezione è implementata dalla funzione **Intersection**

```
Intersection[S1, S2]
```

$$\left\{a, x^4, \mathrm{Sin}[\pi^2]\right\}$$

Range, Complement

Siano dati gli insiemi $X = \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$, $Y = \{1, 4, 7\}$.

```
X = Range[9]
```

```
{1, 2, 3, 4, 5, 6, 7, 8, 9}
```

```
Y = {1, 4, 7};
```

Evidentemente : X - Y = {2, 3, 5, 8, 9}. Questa operazione è svolta da :

```
Complement[X, Y]
```

```
{2, 3, 5, 6, 8, 9}
```

Si osservi che la funzione **Complement** opera anche su insiemi disgiunti. Ad esempio:

```
I1 = Range[3]
```

```
{1, 2, 3}
```

```
I2 = Range[4, 6]
```

```
{4, 5, 6}
```

```
Complement[I1, I2]
```

```
{1, 2, 3}
```

```
Complement[I2, I1]
```

```
{4, 5, 6}
```

In altri termini, la funzione **Complement[lista1,lista2]** restituisce gli elementi contenuti in **lista1** ma che non sono contenuti in **lista2**.

■ First, Last

La funzione **First** restituisce il primo elemento della lista. Ad esempio:

```
lista = {a, b, c, d, e};
```

```
First[lista]
```

```
a
```

Last restituisce l'ultimo elemento:

```
Last[lista]
```

```
e
```

Per l'estrazione dell'elemento *n-esimo*, si utilizza il costrutto **lista[[n]]**. Ad esempio:

```
Clear[lista]
```

$$\text{lista} = \left\{1, 4, 9, 16, 22, -1, 12, -\sqrt{2}\right\};$$

```
lista[[3]]
```

```
9
```

Per eseguire l'operazione inversa:

```
Table[
 lista[[n]], {n, 8}
]
```

$\{1, 4, 9, 16, 22, -1, 12, -\sqrt{2}\}$

Il costrutto **lista[[-n]]**restituisce l'*n-esimo* elemento enumerato dall'ultimo. Ad esempio:

```
lista[[-3]]
```

-1

■ Partition

Per eseguire una partizione su una lista si utilizza **Partition**

```
Clear[lista]
```

```
lista = {1, 8, 27, 64, 125, 216, 343, 512, 729, 1000, 1331, 1728};
```

```
partizione[n_] := Partition[lista, n]
```

```
partizione[1]
```

{{1}, {8}, {27}, {64}, {125}, {216}, {343}, {512}, {729}, {1000}, {1331}, {1728}}

```
partizione[2]
```

{{1, 8}, {27, 64}, {125, 216}, {343, 512}, {729, 1000}, {1331, 1728}}

```
partizione[3]
```

{{1, 8, 27}, {64, 125, 216}, {343, 512, 729}, {1000, 1331, 1728}}

Cioè l'intero *n* è la lunghezza delle singole sottoliste.

```
partizione[6]
```

{{1, 8, 27, 64, 125, 216}, {343, 512, 729, 1000, 1331, 1728}}

```
partizione[7]
```

{{1, 8, 27, 64, 125, 216, 343}}

Una lista di sottoliste:

```
Table[
 partizione[n],
 {n, 5}]
```

```
{{{1}, {8}, {27}, {64}, {125}, {216}, {343}, {512}, {729}, {1000}, {1331}, {1728}},
 {{1, 8}, {27, 64}, {125, 216}, {343, 512}, {729, 1000}, {1331, 1728}},
 {{1, 8, 27}, {64, 125, 216}, {343, 512, 729}, {1000, 1331, 1728}},
 {{1, 8, 27, 64}, {125, 216, 343, 512}, {729, 1000, 1331, 1728}},
 {{1, 8, 27, 64, 125}, {216, 343, 512, 729, 1000}}}
```

Partizionamento con elementi ripetuti:

```
partizione2[n_, d_] := Partition[lista, n, d]
```

```
partizione2[4, 2]
```

```
{{1, 8, 27, 64}, {27, 64, 125, 216},
 {125, 216, 343, 512}, {343, 512, 729, 1000}, {729, 1000, 1331, 1728}}
```

```
partizione2[3, 6]
```

```
{{1, 8, 27}, {343, 512, 729}}
```

Flatten

Per eliminare eventuali livelli di parentesi si utilizza il comando **Flatten**

```
Clear[lista]
```

```
lista = {Sinh[x], {x, x^2, Log[x], Exp[x]}};
```

```
Flatten[lista]
```

$\{\text{Sinh}[x], x, x^2, \text{Log}[x], e^x\}$

O in notazione postfissa:

```
lista // Flatten
```

$\{\text{Sinh}[x], x, x^2, \text{Log}[x], e^x\}$

Il comando **Flatten** dispone di una opzione che controlla il livello di appiattimento.

```
Clear[lista]
```

```
lista = {{{{x^2, y + 1, I, {2}}}}, {2, 4}};
```

```
Flatten[lista, 2]
```

$\{\{x^2, 1 + y, i, \{2\}\}, 2, 4\}$

```
Flatten[lista, 3]
```

$\{x^2, 1 + y, i, \{2\}, 2, 4\}$

```
Flatten[lista, 4]
```

$\{x^2, 1 + y, i, 2, 2, 4\}$

```
Table[
 Flatten[lista, n],
 {n, 1, 4}
]
```

$\{\{\{\{x^2, 1 + y, i, \{2\}\}\}, 2, 4\}, \{\{x^2, 1 + y, i, \{2\}\}, 2, 4\},$
$\{x^2, 1 + y, i, \{2\}, 2, 4\}, \{x^2, 1 + y, i, 2, 2, 4\}\}$

```
%[[4]]
```

$\{x^2, 1 + y, i, 2, 2, 4\}$

Delete

Per eliminare da una lista l'elemento *i-esimo*, si utilizza il comando **Delete**. Ad esempio:

```
Clear[lista]
```

```
lista = {x^Sin[x], 2, y + c, x + 1};
```

```
Delete[lista, 3]
```

$\{x^{\text{Sin}[x]}, 2, 1 + x\}$

Prepend, Append

Per inserire in una lista un nuovo elemento, si utilizza **Prepend**

```
Clear[lista]
```

```
lista = {2, 3, 4, 5};
```

```
Prepend[lista, 1]
```

```
{1, 2, 3, 4, 5}
```

In questo modo la lista rimane immutata:

```
lista
```

```
{2, 3, 4, 5}
```

Se vogliamo modificare anche la lista utilizziamo **PrependTo**

```
PrependTo[lista, 1]
```

```
{1, 2, 3, 4, 5}
```

```
lista
```

```
{1, 2, 3, 4, 5}
```

Volendo inserire un elemento in coda ad una lista, si utilizza **Append**

```
Clear[lista]
```

```
lista = Range[4]
```

```
{1, 2, 3, 4}
```

```
Append[lista, 5]
```

```
{1, 2, 3, 4, 5}
```

Osserviamo tuttavia, che la lista è immutata:

```
lista
```

```
{1, 2, 3, 4}
```

Volendo modificare anche la lista iniziale, si utilizza **AppendTo**

```
AppendTo[lista, 5]
```

```
{1, 2, 3, 4, 5}
```

```
lista
```

```
{1, 2, 3, 4, 5}
```

```
PrependTo[lista, 1]
```

```
{1, 1, 2, 3, 4, 5}
```

■ ReplacePart

Se vogliamo sostituire l'elemento *k-esimo* con un elemento, utilizziamo la funzione **ReplacePart**

```
Clear[lista]
lista = {x + y, x + z, x + 2 * y, x + 2 * z};
ReplacePart[lista, x + 3 * z, 2]
{x + y, x + 3 z, x + 2 y, x + 2 z}
ReplacePart[lista, x + 3 * z, -2]
{x + y, x + z, x + 3 z, x + 2 z}
```

■ Position, MemberQ, Sort, Reverse

L'istruzione **Position** restituisce la posizione di un elemento all'interno di una lista assegnata.

```
Clear[lista]
lista = {a, a, z, x, b, {d, 2, {e, {f, g}}}};
Position[lista, d]
{{6, 1}}
Position[lista, x]
{{4}}
Position[lista, e]
{{6, 3, 1}}
Position[lista, f]
{{6, 3, 2, 1}}
Position[lista, g]
{{6, 3, 2, 2}}
```

MemberQ[lista,y] controlla se **y** appartiene alla lista, restituendo **True** in caso affermativo, **False** nel caso contrario.

```
MemberQ[lista, a]
True
MemberQ[lista, d]
False
```

Nel caso di **d** il kernel restituisce **False**, in quanto tale elemento si trova in un livello annidato.

Se una lista è costituita da parole o lettere, il comando **Sort** riordina la lista in ordine alfabetico, mentre **Reverse** inverte l'ordine.

```
Clear[lista]
lista = {x, d, g, p, q, w, e, h, d, f, j, x};
lista // Sort
{d, d, e, f, g, h, j, p, q, w, x, x}
```

```
(*eliminiamo gli elementi ripetuti*)
lista2 = lista // Union
```

```
{d, e, f, g, h, j, p, q, w, x}
```

```
(*invertiamo l'ordine*)
```

```
lista2 // Reverse
```

```
{x, w, q, p, j, h, g, f, e, d}
```

■ Map

Map è probabilmente il comando più potente in grado di agire sulle liste. Esaminiamo la sua sintassi. Per essere più precisi, tale comando non agisce solo sulle liste, cioè sugli insiemi discreti, ma anche su quelli continui, del tipo intervallo (limitato o illimitato) di numeri reali. Quindi consideriamo i due casi distinti: 1) X è un insieme continuo, 2) X è un insieme discreto. Nel primo caso sia $f : X \to Y$, una funzione da $X \subseteq (-\infty, +\infty)$. Il costrutto **Map[f,X]** restituisce l'insieme $f(X)$, cioè l'immagine di X attraverso f. Come è noto, tale sottoinsieme di Y si chiama *codominio* della funzione.

```
(*un intervallo [a,b] si indica - in codice Mathematica - con Interval[{a,b}]*)
```

```
Map[ArcSin, Interval[{-1, 1}]]
```

$$\text{Interval}\left[\left\{-\frac{\pi}{2}, \frac{\pi}{2}\right\}\right]$$

Una sintassi abbreviata è:

```
ArcSin@Interval[{-1, 1}]
```

$$\text{Interval}\left[\left\{-\frac{\pi}{2}, \frac{\pi}{2}\right\}\right]$$

Cioè anzichè scrivere **Map[f,X]** scriviamo **f@X**, oppure **f/@X**

```
ArcSin /@ Interval[{-1, 1}]
```

$$\text{Interval}\left[\left\{-\frac{\pi}{2}, \frac{\pi}{2}\right\}\right]$$

Passiamo al caso discreto:

```
Clear[f]
```

```
X = {a, b, c, d, e};
```

```
Map[f, X]
```

```
{f[a], f[b], f[c], f[d], f[e]}
```

```
f /@ X
```

```
{f[a], f[b], f[c], f[d], f[e]}
```

Osserviamo che **Map[f,X]** opera solo sul primo livello ignorando eventuali livelli annidati. Esempio:

```
Clear[f, X]
```

```
X = {{1, 2}, {4, {0, 10}}}
```

```
{{1, 2}, {4, {0, 10}}}
```

```
f /@ X
```

```
{f[{1, 2}], f[{4, {0, 10}}]}
```

Il costrutto **Map[f,lista,r]** applica f fino al livello r se la lista è costituita da $d > r$ livelli.

```
Map[f, X, 2]
```

```
{f[{f[1], f[2]}], f[{f[4], f[{0, 10}]}]}
```

Il comando **MapAll[f,X]** applica f a tutti i livelli. La sintassi equivalente è **f//@X**

```
f //@ X
```

```
f[{f[{f[1], f[2]}], f[{f[4], f[{f[0], f[10]}]}]}]
```

■ L'attributo Listable

Le funzioni built-in hanno l'attributo **Listable**: ignorano l'eventuale presenza di parentesti graffe (ad ogni livello) nei loro argomenti. Esempio:

```
Sin[{a, {4, pi}, {2, 1}}]
```

```
{Sin[a], {Sin[4], Sin[pi]}, {Sin[2], Sin[1]}}
```

Per default le funzioni definite dall'utente non hanno l'attributo **Listable**. Esempio:

```
f[{a, {4, pi}, {2, 1}}]
```

```
f[{a, {4, pi}, {2, 1}}]
```

Per assegnare l'attributo **Listable** si utilizza il comando **SetAttributes[f, Listable]**

```
SetAttributes[f, Listable]
```

```
f[{a, {4, pi}, {2, 1}}]
```

```
{f[a], {f[4], f[pi]}, {f[2], f[1]}}
```

Per quanto detto, le funzioni built-in hanno l'attributo **Listable**. Ad esempio, l'operazione di derivazione di una funzione reale di una o più variabili, è implementata dalla funzione built-in **D[f,x]**, dove **f** è la funzione e **x** la variabile rispetto a cui vogliamo determinare la derivata. Consideriamo un esempio istruttivo di operazione di derivazione, anche se l'applicazione grafica richiede la conoscenza del comando **ParametricPlot**. Si tratta di tracciare la traiettoria di un oscillatore armonico bidimensionale. Come è noto, per particolari valori delle frequenze dei moti componenti, si ottengono le note figure di Lissajous.

```
Clear[f, x, y]
```

```
x[t_] := Cos[3 t]; y[t_] := Cos[2 t]
```

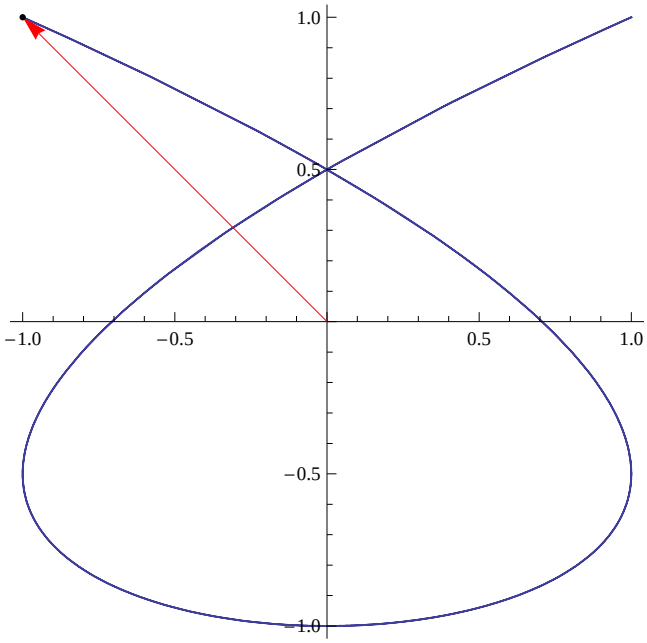
```
posizione[t_] := {x[t], y[t]}
```

```
velocita[t_] = D[posizione[t], t]
```

```
{-3 Sin[3 t], -2 Sin[2 t]}
```

```
grafico[τ_] := ParametricPlot[
  {x[t], y[t]},
  {t, 0, 2 π},
  PlotStyle → Thickness[0.003],
  Epilog → {
    {
     Red, Arrow[{{0, 0}, {x[τ], y[τ]}}]
    },
    Point[{x[τ], y[τ]}]
   }
 ]
```

```
grafico[π]
```



Vettori e Matrici

Abbiamo già visto che *Mathematica* rappresenta vettori e matrici attraverso le liste. Precisamente, un vettore è una lista di scalari, mentre una matrice è una lista di vettori. Tutto ciò che non rientra in una lista, per *Mathematica* è uno scalare. Ne consegue che per poter eseguire operazioni sulle matrici, occorre familiarizzare con le principali istruzione che operano sulle liste. Come abbiamo avuto modo di vedere, un potente generatore di liste è **Table**, per cui se ad esempio, vogliamo dichiarare una matrice quadrata di ordine 4, scriviamo:

```
A = Table[
   (*elemento di matrice*)
   a[i, k],
   (*indice di riga*)
   {i, 4},
   (*indice di colonna*)
   {k, 4}
  ] // MatrixForm
```

$$\begin{pmatrix} a[1,1] & a[1,2] & a[1,3] & a[1,4] \\ a[2,1] & a[2,2] & a[2,3] & a[2,4] \\ a[3,1] & a[3,2] & a[3,3] & a[3,4] \\ a[4,1] & a[4,2] & a[4,3] & a[4,4] \end{pmatrix}$$

Il codice precedente può essere utilizzato a patto di conoscere l'espressione di a_{ik} in funzione degli indici i, k. Diversamente, la matrice va dichiarata elemento per elemento. Ad esempio supponiamo di avere la seguente matrice sul campo complesso:

```
B = {
  (*prima riga*)
  {-e, √2 , π},
  (*seconda riga*)
  {i, -1, 0},
  {0, 0, 2 + 3 i}
 }
```

$$\left\{\left\{-e, \sqrt{2}, \pi\right\}, \{i, -1, 0\}, \{0, 0, 2+3i\}\right\}$$

Visualizziamola nel classico formato:

```
B // MatrixForm
```

$$\begin{pmatrix} -e & \sqrt{2} & \pi \\ i & -1 & 0 \\ 0 & 0 & 2+3i \end{pmatrix}$$

```
Clear[A, B, a]
```

In alternativa a **Table**, c'è l'istruzione **Array**:

```
A = Array[
  a,
  {3, 3}
 ]
```

```
{{a[1, 1], a[1, 2], a[1, 3]}, {a[2, 1], a[2, 2], a[2, 3]}, {a[3, 1], a[3, 2], a[3, 3]}}
```

L'istruzione **Map** restituisce l'immagine di una applicazione f:

```
lista = {x, y, z, t};
```

```
Map[
 (*applicazione/funzione*)
 f,
 (*insieme di definizione*)
 lista
]
```

```
{f[x], f[y], f[z], f[t]}
```

```
Map[
 Sin,
 {+∞}
]
```

```
{Interval[{-1, 1}]}
```

Riprendiamo la matrice

```
B = {
   (*prima riga*)
   {-e, √2 , π},
   (*seconda riga*)
   {i, -1, 0},
   {0, 0, 2 + 3 i}
  };
```

Il generico elemento di matrice è:

```
b[i_, k_] := B[[i, k]]
```

```
b[1, 1]
```

```
-e
```

```
b[2, 1]
```

```
i
```

```
b[3, 1]
```

```
0
```

etc.

La riga *i-esima*:

```
r[i_] := B[[i]]
```

```
r[1]
```

```
{-e, √2 , π}
```

```
r[2]
```

```
{i, -1, 0}
```

```
r[3]
```

```
{0, 0, 2 + 3 i}
```

Per la colonna *k-esima* utilizziamo l'istruzione **`Transpose`** che esegue l'operazione di trasposizione di una matrice assegnata, i.e. calcola la matrice trasposta B^T:

```
Clear[c]
```

```
c[k_] := (*trasposta di B*)Transpose[B](*estrae la riga k-esima di Bᵀ*)[[k]]
```

```
c[1]
```

```
{-e, i, 0}
```

```
c[2]
```

```
{√2 , -1, 0}
```

```
Clear[c]
```

In alternativa, utilizziamo **`Map`** e la nozione di funzione pura

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

```
c[k_] := Map[#[[k]] &, B]
```

```
c[2]
```

$\{\sqrt{2}, -1, 0\}$

```
Clear[b, c, f]
```

Operazioni elementari su vettori e matrici

Moltiplicazione di uno scalare λ per un vettore (a, b, c):

```
λ {a, b, c}
```

$\{a\,\lambda, b\,\lambda, c\,\lambda\}$

Per quanto detto in precedenza, *Mathematica* tratta i vettori come liste, per cui in una data operazione se un vettore non è dichiarato come lista, *Mathematica* lo tratta come uno scalare. Ad esempio, se v è un vettore di $\mathbb{R}^3$ e lo sommiamo al vettore precedente:

```
vettoresomma = v + {a, b, c}
```

$\{a + v, b + v, c + v\}$

che è la somma dello scalare v e del vettore (a, b, c). Ad esempio se $v = \{d, e, f\}$, proviamo a sostituire nell'espressione precedente

```
vettoresomma1 = vettoresomma /. v → {d, e, f}
```

$\{\{a + d, a + e, a + f\}, \{b + d, b + e, b + f\}, \{c + d, c + e, c + f\}\}$

cioè una matrice:

```
vettoresomma1 // MatrixForm
```

$$\begin{pmatrix} a+d & a+e & a+f \\ b+d & b+e & b+f \\ c+d & c+e & c+f \end{pmatrix}$$

Il risultato corretto si ottiene dichiarando v

```
v = {d, e, f};
```

```
vettoresomma2 = v + {a, b, c}
```

$\{a + d, b + e, c + f\}$

```
Clear[v]
```

In uno spazio vettoriale dotato di prodotto scalare, il prodotto scalare canonico viene eseguito con il punto: **.** Ad esempio

```
v = {v1, v2, v3}; w = {w1, w2, w3};
```

```
v.w
```

$v1\,w1 + v2\,w2 + v3\,w3$

```
Clear[A, B, a, b, c, d]
```

```
A = {
  (*prima riga*)
  {x, y}
 }
```

```
{{x, y}}
```

```
B = {
  (*prima riga*)
  {a, b},
  (*seconda riga*)
  {c, d}
 }
```

```
{{a, b}, {c, d}}
```

```
A.B
```

```
{{a x + c y, b x + d y}}
```

```
Clear[A, B]
```

```
A = Array[
   a,
   {5, 5}
  ];
```

```
B = Array[
   b,
   {5, 2}
  ];
```

```
A.B
```

```
{{a[1, 1] b[1, 1] + a[1, 2] b[2, 1] + a[1, 3] b[3, 1] + a[1, 4] b[4, 1] + a[1, 5] b[5, 1],
  a[1, 1] b[1, 2] + a[1, 2] b[2, 2] + a[1, 3] b[3, 2] + a[1, 4] b[4, 2] + a[1, 5] b[5, 2]},
 {a[2, 1] b[1, 1] + a[2, 2] b[2, 1] + a[2, 3] b[3, 1] + a[2, 4] b[4, 1] + a[2, 5] b[5, 1],
  a[2, 1] b[1, 2] + a[2, 2] b[2, 2] + a[2, 3] b[3, 2] + a[2, 4] b[4, 2] + a[2, 5] b[5, 2]},
 {a[3, 1] b[1, 1] + a[3, 2] b[2, 1] + a[3, 3] b[3, 1] + a[3, 4] b[4, 1] + a[3, 5] b[5, 1],
  a[3, 1] b[1, 2] + a[3, 2] b[2, 2] + a[3, 3] b[3, 2] + a[3, 4] b[4, 2] + a[3, 5] b[5, 2]},
 {a[4, 1] b[1, 1] + a[4, 2] b[2, 1] + a[4, 3] b[3, 1] + a[4, 4] b[4, 1] + a[4, 5] b[5, 1],
  a[4, 1] b[1, 2] + a[4, 2] b[2, 2] + a[4, 3] b[3, 2] + a[4, 4] b[4, 2] + a[4, 5] b[5, 2]},
 {a[5, 1] b[1, 1] + a[5, 2] b[2, 1] + a[5, 3] b[3, 1] + a[5, 4] b[4, 1] + a[5, 5] b[5, 1],
  a[5, 1] b[1, 2] + a[5, 2] b[2, 2] + a[5, 3] b[3, 2] + a[5, 4] b[4, 2] + a[5, 5] b[5, 2]}}
```

```
Clear[A, B, a, b]
```

```
A[m_, n_] := Array[
  a,
  {m, n}
 ]
```

```
B[n_, p_] := Array[
  b,
  {n, p}
 ]
```

```
f[m_, n_, p_] := A[m, n].B[n, p]
```

```
f[3, 1, 3] // MatrixForm
```

$$\begin{pmatrix} a[1,1]\,b[1,1] & a[1,1]\,b[1,2] & a[1,1]\,b[1,3] \\ a[2,1]\,b[1,1] & a[2,1]\,b[1,2] & a[2,1]\,b[1,3] \\ a[3,1]\,b[1,1] & a[3,1]\,b[1,2] & a[3,1]\,b[1,3] \end{pmatrix}$$

Sistemi di equazioni lineari

LinearSolve

Mathematica risolve sistemi di equazioni lineari attraverso l'istruzione **LinearSolve**, la cui sintassi richiede la scrittura matriciale del sistema, cioè del tipo AX = *B*, dove $A = (a_{ik})$ è la matrice dei coefficienti, mentre *X* e *B* sono rispettivamente il vettore colonna delle incognite e il vettore colonna dei termini noti. Ad esempio, nel caso di un sistema di 3 equazioni nelle 3 incognite *x*, *y*, *z*, si ha:

```
A = {{0, 1, 2}, {2, -√3, 1}, {1, -4, 0}}; B = {2, 1, -1};
```

```
X = LinearSolve[
  (*matrice dei coefficienti*)
  A,
  (*vettore dei termini noti*)
  B
 ]
```

$$\left\{-1-\frac{16}{-15+2\sqrt{3}},\ -\frac{4}{-15+2\sqrt{3}},\ \frac{-13+2\sqrt{3}}{-15+2\sqrt{3}}\right\}$$

```
x = X[[1]]
```

$$-1-\frac{16}{-15+2\sqrt{3}}$$

```
y = X[[2]]
```

$$-\frac{4}{-15+2\sqrt{3}}$$

```
z = X[[3]]
```

$$\frac{-13+2\sqrt{3}}{-15+2\sqrt{3}}$$

Verifica:

```
A.X == B
```

```
True
```

Proviamo con un sistema omogeneo:

```
Clear[X]
```

```
X = LinearSolve[
  A,
  {0, 0, 0}
 ]
```

```
{0, 0, 0}
```

Attenzione: nel caso omogeneo, l'istruzione **LinearSolve** restituisce solo la soluzione banale, ignorando le eventuali soluzioni non nulle (le cosiddette *soluzioni proprie* o *autosoluzioni*). Nel caso omogeneo bisogna utilizzare l'istruzione **NullSpace** che restituisce una base dello spazio nullo della matrice dei coefficienti:

```
A // NullSpace
```

```
{}
```

cioè il sottospazio improprio, per cui il sistema assegnato ammette la sola soluzione banale. Ciò è confermato dalla determinazione del rango di A (che in questo caso è uguale al numero delle incognite)

```
A // MatrixRank
```

```
3
```

```
Clear[A, B, X]
```

Consideriamo quest'altro esempio:

```
A = {{2, 7, -3}, {2, -3, 2}, {2, 3, -1}};
```

Riesce:

```
A // MatrixRank
```

```
2
```

onde il sistema $AX = 0$ ammette ∞^1 autosoluzioni. Proviamo a determinarle con il comando **LinearSolve**:

```
LinearSolve[
 A,
 {0, 0, 0}
]
```

```
{0, 0, 0}
```

che come preannunciato, restituisce la sola soluzione banale. Utilizzando **NullSpace**

```
A // NullSpace
```

```
{{-1, 2, 4}}
```

per cui la soluzione generale è $\lambda(-1, 2, 4)$, essendo λ un numero reale arbitrario. Possiamo comunque forzare *Mathematica* attraverso il comando **Solve**

```
X = {x, y, z};
```

```
Solve[
 A.X == {0, 0, 0},
 X
]
```

General::ivar : $\frac{-13+2\sqrt{3}}{-15+2\sqrt{3}}$ is not a valid variable. ≫

General::ivar : $\frac{-13+2\sqrt{3}}{-15+2\sqrt{3}}$ is not a valid variable. ≫

$$\text{Solve}\left[\text{False}, \left\{-1-\frac{16}{-15+2\sqrt{3}}, -\frac{4}{-15+2\sqrt{3}}, \frac{-13+2\sqrt{3}}{-15+2\sqrt{3}}\right\}\right]$$

o con **Reduce**

```
Reduce[
 A.X == {0, 0, 0},
 X
]
```

Reduce::ivar : $-1-\frac{16}{-15+2\sqrt{3}}$ is not a valid variable. ≫

$$\text{Reduce}\left[\text{False}, \left\{-1-\frac{16}{-15+2\sqrt{3}}, -\frac{4}{-15+2\sqrt{3}}, \frac{-13+2\sqrt{3}}{-15+2\sqrt{3}}\right\}\right]$$

che però restituisce solo una delle infinite soluzioni non nulle.

```
Clear[A, B, X]
```

LinearSolveFunction

Se nell'istruzione **LinearSolve** non specifichiamo il vettore dei termini noti, *Mathematica* restituisce una **LinearSolveFunction**:

```
A = {{2, 5}, {-1, √2}};
linearsolve = LinearSolve[A]
```

```
LinearSolveFunction[{2, 2}, <>]
```

Ad esempio, se il vettore dei termini noti è

```
B = {2, -3};
linearsolve[B] // Simplify
```

$$\left\{1+\frac{5}{\frac{5}{2}+\sqrt{2}}, -\frac{4}{5+2\sqrt{2}}\right\}$$

Verifica:

```
A.% == B
```

```
True
```

```
Clear[A, B]
```

Sistemi incompatibili

Per un sistema "quadrato" (i.e. la matrice dei coefficienti è una matrice quadrata) con rango minore del numero delle incognite, **LinearSolveFunction** da un messaggio di errore

```
A = {{1, 1}, {1, 1}};
linearsolve = LinearSolve[A]
```

LinearSolve::sing1 : The matrix {{1, 1}, {1, 1}} is singular so a factorization will not be saved.

```
LinearSolveFunction[{2, 2}, <>]
```

In ogni caso restituisce il risultato corretto:

```
linearsolve[{2, 2}]
```

```
{2, 0}
```

Per un vettore dei termini noti del tipo $(b, -b)$ il sistema è incompatibile:

```
linearsolve[{b, -b}]
```

```
LinearSolve::nosol : Linear equation encountered that has no solution. ≫
LinearSolveFunction[{2, 2}, <>][{b, -b}]
```

Funzioni definite dall'utente (parte seconda)

Abbiamo visto che le funzioni definite dall'utente si realizzano attraverso un pattern _. Nel caso specifico di un'assegnazione ritardata, scriviamo **f[x_]:=<expr>**. Per quanto detto, un pattern rappresenta "qualunque cosa". Più rigorosamente, un pattern rappresenta una classe di espressioni. È naturale chiedersi se sia possibile vincolare il pattern ad un formato specifico. La risposta è affermativa e la sintassi è **x_Formato**. Ad esempio l'input:

```
f[x_Integer] := x * Log[x + 1]
```

accetta solo valori interi della variabile indipendente. Infatti:

```
f[2] // N
```

```
2.19722
```

```
f[√2]
```

```
f[√2]
```

```
f[a]
```

```
f[a]
```

```
Clear[f]
```

Consideriamo il codice:

```
f[x_Symbol] := x * Log[x + 1]
```

```
f[2]
```

```
f[2] // N
```

```
f[2.]
```

```
f[a]
```

```
a Log[1 + a]
```

Abbiamo quindi la situazione opposta alla precedente: il costrutto **x_Symbol** accetta solo simboli.

```
Clear[f, lista]
```

Un pattern può essere vincolato ad una lista. Ad esempio:

```
lista = {Exp[-a * x^2], x, x^2};
```

```
f[x_List] := ArcTanh[x]
```

```
f[2] // N
```

```
f[2.]
```

```
f[lista]
```

$\{\text{ArcTanh}[e^{-a x^2}], \text{ArcTanh}[x], \text{ArcTanh}[x^2]\}$

```
Clear[f]
```

Se *f* è una funzione definita dall'utente (o built-in), è possibile applicare *n* volte *f* a se stessa in un punto *x*, applicando la funzione built-in **Nest[f,x,n]**. Esempio:

```
Nest[f, x, 4]
```

```
f[f[f[f[x]]]]
```

NestList[f,x,n] genera invece la lista di applicazioni.

```
NestList[f, x, 4]
```

```
{x, f[x], f[f[x]], f[f[f[x]]], f[f[f[f[x]]]]}
```

In altri termini, mentre **Nest[f,x,n]** restituisce una funzione, **NestList[f,x,n]**restituisce una lista di funzioni. Esempio:

```
Clear[f]
```

```
f[x_] := Sin[ArcTan[x] + 4]
```

```
(*utilizziamo il comando Symplify per semplificare l'output*)
```

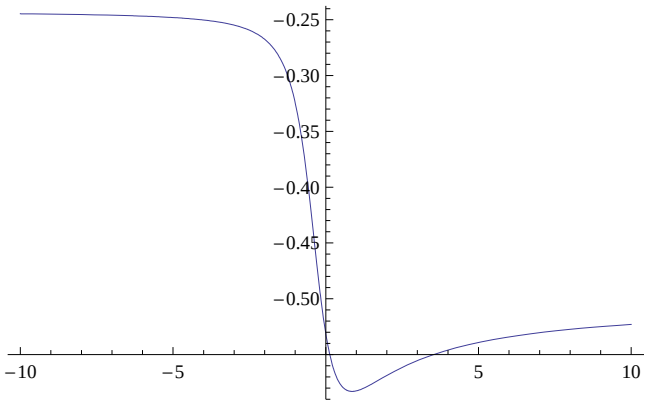
```
fn[x_, n_] := Nest[f, x, n] // Simplify
```

```
fn[x, 4]
```

```
Sin[4 + ArcTan[Sin[4 + ArcTan[Sin[4 + ArcTan[Sin[4 + ArcTan[x]]]]]]]]
```

```
(*tracciamo un grafico di fn[x,5]*)
```

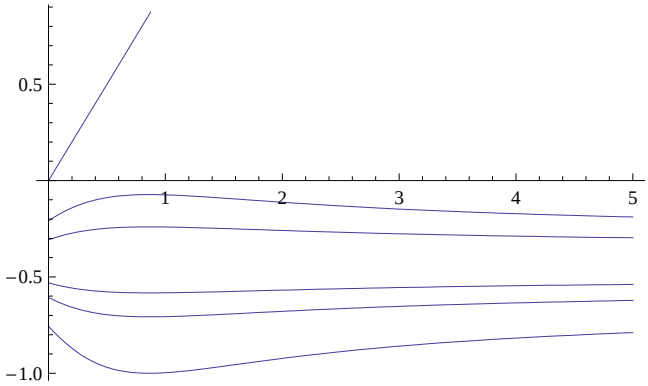
```
Plot[
 fn[x, 5],
 {x, -10, 10}
]
```




```
NestList[f, x, 5]
```

```
{x, Sin[4 + ArcTan[x]], Sin[4 + ArcTan[Sin[4 + ArcTan[x]]]],
 Sin[4 + ArcTan[Sin[4 + ArcTan[Sin[4 + ArcTan[x]]]]]],
 Sin[4 + ArcTan[Sin[4 + ArcTan[Sin[4 + ArcTan[Sin[4 + ArcTan[x]]]]]]]],
 Sin[4 + ArcTan[Sin[4 + ArcTan[Sin[4 + ArcTan[Sin[4 + ArcTan[Sin[4 + ArcTan[x]]]]]]]]]]}
```

```
Plot[
 Table[fn[x, n], {n, 0, 5}],
 {x, 0, 5}
]
```




```
Clear[f, fn]
```

Un altro esempio

$$f[x\_] := \frac{1}{1 + \mathrm{Sin}[x]}$$

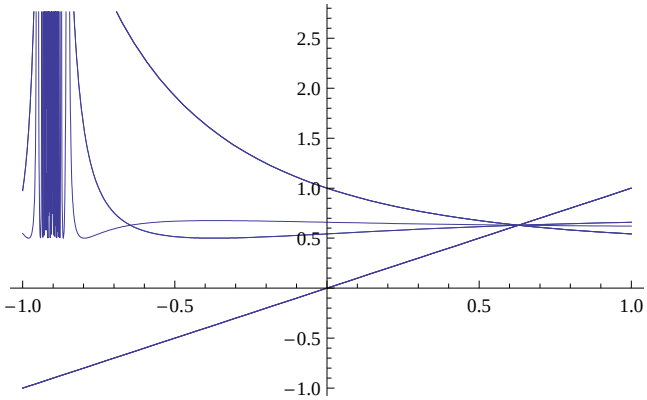
```
Nest[f, x, 5]
```

$$\frac{1}{1 + \mathrm{Sin}\left[\frac{1}{1+\mathrm{Sin}\left[\frac{1}{1+\mathrm{Sin}\left[\frac{1}{1+\mathrm{Sin}[x]}\right]}\right]}\right]}$$

```
NestList[f, x, 5]
```

$$\left\{x, \frac{1}{1 + \mathrm{Sin}[x]}, \frac{1}{1 + \mathrm{Sin}\left[\frac{1}{1+\mathrm{Sin}[x]}\right]}, \frac{1}{1 + \mathrm{Sin}\left[\frac{1}{1+\mathrm{Sin}\left[\frac{1}{1+\mathrm{Sin}[x]}\right]}\right]}, \frac{1}{1 + \mathrm{Sin}\left[\frac{1}{1+\mathrm{Sin}\left[\frac{1}{1+\mathrm{Sin}\left[\frac{1}{1+\mathrm{Sin}[x]}\right]}\right]}\right]}, \frac{1}{1 + \mathrm{Sin}\left[\frac{1}{1+\mathrm{Sin}\left[\frac{1}{1+\mathrm{Sin}\left[\frac{1}{1+\mathrm{Sin}\left[\frac{1}{1+\mathrm{Sin}[x]}\right]}\right]}\right]}\right]}\right\}$$

```
Plot[
 Table[
  NestList[f, x, n],
  {n, 0, 3}
 ],
 {x, -1, 1}
]
```




```
Clear[f]
```

I comandi **Nest** e **NestList** hanno una notevole applicazione nella ricerca delle soluzioni di un'equazione del tipo $f(x) = x$ che come è noto, si risolve per ricorsione. Più precisamente, Find si sceglie un punto iniziale x_0 per poi iterare f su x_0, studiando la convergenza della successione ottenuta. Sul testo di Stan Wagon (*Guida a Mathematica*, ed. McGrawHill) è riportato il caso dell' equazione $\cos(x) = x$. Assumiamo come punto iniziale $x_0 = 1.5$, quindi iteriamo $f(x) = \cos(x)$ su x_0 per n volte :

```
f[x_] := Cos[x]
```

```
x0 = 1.5;
```

```
iterazione[n_] := NestList[
  f,
  x0,
  n
 ]
```

Per $n = 30$:

```
iterazione[30]
```

```
{1.5, 0.0707372, 0.997499, 0.542405, 0.85647, 0.655109, 0.792982, 0.701724,
 0.76373, 0.722261, 0.750313, 0.731476, 0.74419, 0.735637, 0.741403, 0.737522,
 0.740137, 0.738376, 0.739563, 0.738763, 0.739302, 0.738939, 0.739183,
 0.739019, 0.73913, 0.739055, 0.739105, 0.739071, 0.739094, 0.739079, 0.739089}
```

Le iterazioni sembrano convergere a 0.739089:

```
f[0.739089]
```

```
0.739083
```

```
Clear[iterazione]
```

Per esaminare termini successivi Wagon suggerisce di annidare **Nest** in **NestList**, poichè **Nest** restituisce $l'\,n-$ esima iterata :

```
iterazione[n_, m_] := NestList[
  f,
  Nest[
   f,
   x0,
   n
  ],
  m
 ]
```

Ad esempio, se fissiamo $n = 10^3$ *e* $m =$ 10^2

```
iterazione[10^3, 10^2]
```

```
{0.739085, 0.739085, 0.739085, 0.739085, 0.739085, 0.739085, 0.739085,
 0.739085, 0.739085, 0.739085, 0.739085, 0.739085, 0.739085, 0.739085,
 0.739085, 0.739085, 0.739085, 0.739085, 0.739085, 0.739085, 0.739085,
 0.739085, 0.739085, 0.739085, 0.739085, 0.739085, 0.739085, 0.739085, 0.739085,
 0.739085, 0.739085, 0.739085, 0.739085, 0.739085, 0.739085, 0.739085, 0.739085,
 0.739085, 0.739085, 0.739085, 0.739085, 0.739085, 0.739085, 0.739085, 0.739085,
 0.739085, 0.739085, 0.739085, 0.739085, 0.739085, 0.739085, 0.739085, 0.739085,
 0.739085, 0.739085, 0.739085, 0.739085, 0.739085, 0.739085, 0.739085, 0.739085,
 0.739085, 0.739085, 0.739085, 0.739085, 0.739085, 0.739085, 0.739085, 0.739085,
 0.739085, 0.739085, 0.739085, 0.739085, 0.739085, 0.739085, 0.739085, 0.739085,
 0.739085, 0.739085, 0.739085, 0.739085, 0.739085, 0.739085, 0.739085, 0.739085,
 0.739085, 0.739085, 0.739085, 0.739085, 0.739085, 0.739085, 0.739085, 0.739085,
 0.739085, 0.739085, 0.739085, 0.739085, 0.739085, 0.739085, 0.739085, 0.739085}
```

```
Clear[iterazione]
```

Svincoliamoci dal punto inziale x_0 utilizzando il comando `Random` quale generatore di numeri reali. Precisamente `Random[]` restituisce un numero reale casuale appartenente all'intervallo [0,1]:

```
Random[]
```

```
0.20225
```

Per cambiare l'intervallo di appartenenza, ad esempio [-1,10]:

```
Random[
 (*tipo di numero (Real, Integer, Complex)*)
 Real,
 (*intervallo di appartenenza*)
 {-1, 10}
]
```

```
8.39963
```

Ciò detto, assumiamo x_0 numero reale casuale appartenente all'intervallo [-*a*,*a*] per un assegnato $a > 1$

```
x0[a_] := Random[
  (*tipo di numero (Real, Integer, Complex)*)
  Real,
  (*intervallo di appartenenza*)
  {-a, a}
 ]
```

A questo punto osserviamo che il comando `Table` (visto in precedenza) può essere utilizzato senza iteratore. Ad esempio:

```
Table[
 x,
 {10}
]
```

```
{x, x, x, x, x, x, x, x, x, x}
```

Quindi scriviamo:

```
iterazione[a_] := Table[
  Nest[
   f,
   x0[a],
   a],
  {20}
 ]
```

Per $a = 10^2$:

```
iterazione[10^2]
```

```
{0.739085, 0.739085, 0.739085, 0.739085, 0.739085, 0.739085,
 0.739085, 0.739085, 0.739085, 0.739085, 0.739085, 0.739085, 0.739085,
 0.739085, 0.739085, 0.739085, 0.739085, 0.739085, 0.739085, 0.739085}
```

Ne consegue che la convergenza a 0.739085 è indipendente dal punto iniziale x_0. Inoltre, esistenza ed unicità della radice dell'equazione $f(x) = x$ è espressa dal noto Teorema di Brouwer (*o* del punto fisso) che afferma :

" Se f è una funzione derivabile in $[a, b]$ tale che $|f'(x)| < \mu$, allora esiste ed è unica la radice dell' equazione $f(x) = x$.

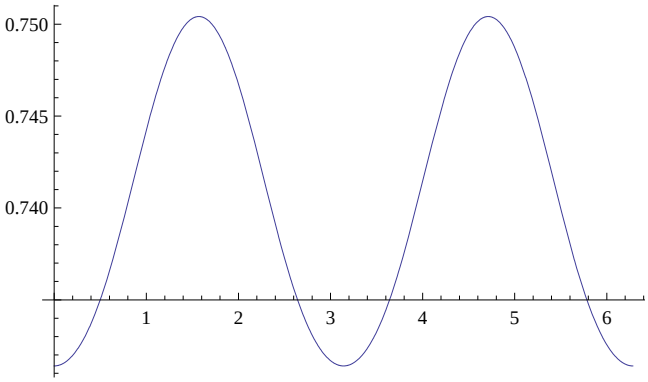
Dal momento che la convergenza del processo di iterazione è indipendente dal punto iniziale, segue che il risultato dell'iterazione di ordine n $f((f(f(\ldots f(x)))))$ per $n \to +\infty$ è la funzione costante $f(x) = x_0$. Infatti :

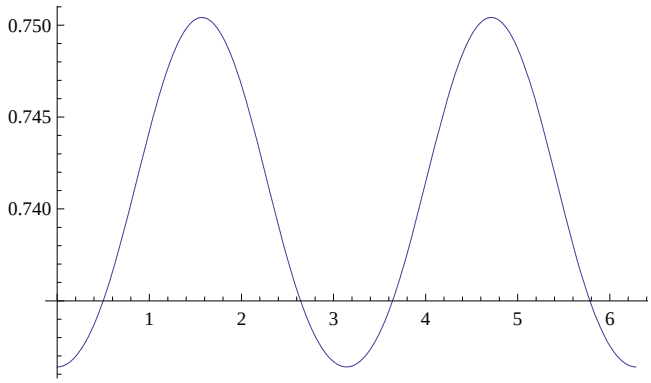
```
Clear[n]
```

```
fn[x_, n_] := Nest[f, x, n]
```

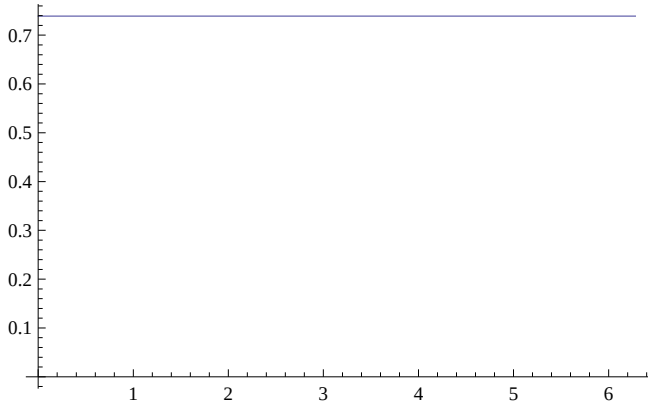
```
plot[n_] := Plot[
  fn[x, n],
  {x, 0, 2 π},
  PlotRange → All
 ]
```

```
plot[10]
```






```
plot[214]
```




```
Clear[F, f]
```

Un punto da tener presente è che in *Mathematica* le funzioni non sono tali solo nel senso dell'Analisi matematica, cioè enti del tipo $f : x \in X \to f(x) \in R$ oppure $g : (x_1, x_2, \ldots, x_n) \in A \to R$, bensì oggetti il cui argomento può essere a sua volta una funzione. Ad esempio, la seguente istruzione:

```
F[f_] := Integrate[f, x]
```

restituisce una primitiva della funzione reale di una variabile reale f, che come vediamo è lasciata inespressa. Si noti l'utilizzo dell'assegnazione ritardata; in tal modo l'integrale viene calcolato quando richiamiamo F per un assegnato argomento f. Ad esempio:

```
F[x^10]
```

$$\frac{x^{11}}{11}$$

Notiamo che nell'istruzione **Integrate[f,x]** bisogna introdurre solo il simbolo **f** e non **f[x]**. Rammentiamo che in Analisi matematica il simbolo corretto è il primo, giacché rappresenta la legge di corrispondenza tra l'insieme X (insieme di definizione della funzione) e un altro sottoinsieme di R, mentre **f[x]** denota il valore assunto dalla funzione in un generico punto $x \in X$. In ogni caso, proviamo:

```
Clear[F, f]

F[f_] := Integrate[f[x], x]

F[Sin]
```

```
-Cos[x]
```

```
F[x^4]
```

$$\int x^4[x]\,dx$$

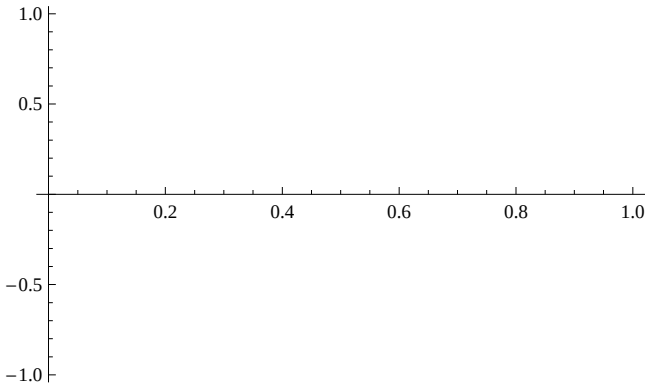
Da questi esempi, vediamo che la seconda assegnazione fornisce il risultato corretto solo per le funzioni built-in, vale a dire **Sin,Cos,Tan**, etc. Se abbiamo una generica $f(x)$ *Mathematica* cerca di calcolare l'integrale di $f(x)(x)$ cosa ovviamente impossibile, in quanto priva di senso.

Anche alcune istruzioni grafiche come ad esempio **Plot[]**, sono funzioni che agiscono su funzioni:

```
plot[f_] := Plot[
  (*espressione della funzione*)
  f,
  (*intervallo*)
  {x, 0, 1}
 ]
```

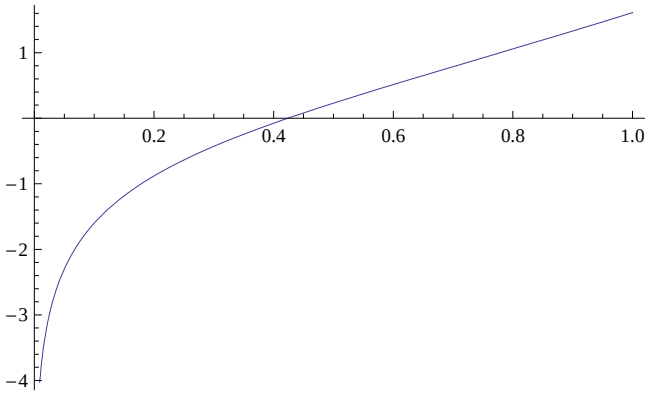
comunica al kernel di graficare la restrizione di una funzione f all'intervallo chiuso [0,1]. È essenziale l'assegnazione ritardata, poiché

```
plot1[f_] = Plot[
  f,
  {x, 0, 1}
 ]
```



Cioè *Mathematica* non traccia nulla, se non gli assi coordinati. E questo è ovvio, in quanto non abbiamo specificato l'espressione analitica della funzione.

```
plot[x^2 + Log[x + Sin[x]]]
```



Possiamo poi definire funzioni il cui argomento è una lista anziché un numero. In questo caso scriviamo **f[lista_List]**

```
f[lista_List] := lista^2
```

```
f[{1, 2, 4}]
```

```
{1, 4, 16}
```

Le liste possono essere anche annidate:

```
f[{1, {2, 4}, 8, {{{2, 10}}}}]
```

```
{1, {4, 16}, 64, {{{4, 100}}}}
```

Se tentiamo di calcolare:

```
f[2]
```

```
f[2]
```

Mathematica non calcola nulla, giacché abbiamo specificato che l'argomento della funzione (cioè la variabile indipendente) deve essere una lista e non un numero.

```
Clear[f]
```

Grafica bidimensionale

La nozione di primitiva grafica

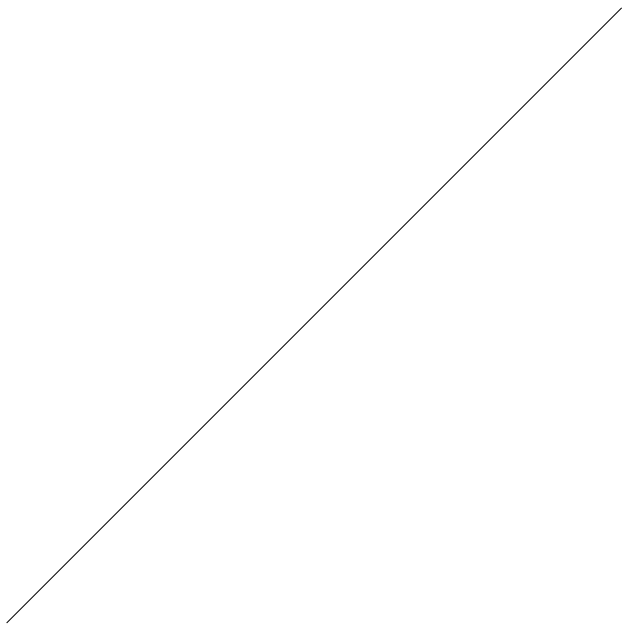
Un segmento che unisce due punti del piano cartesiano, per *Mathematica* è una **primitiva grafica**, come lo è anche un generico punto $P(x, y)$ del piano cartesiano. Ad esempio, il segmento che unisce i punti $A(2, 1)$ e $B(0, -1)$ è:

```
segmento = Line[{{2, 1}, {0, -1}}]
```

```
Line[{{2, 1}, {0, -1}}]
```

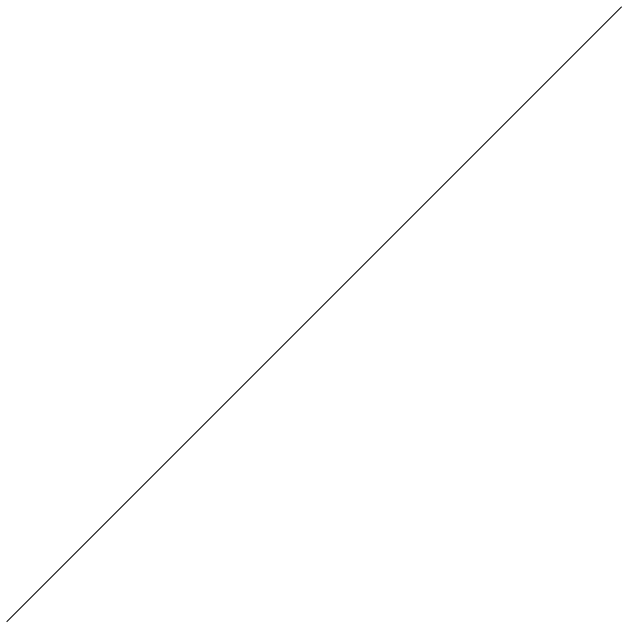
Alcune osservazioni: 1) notiamo che l'argomento di **Line** è una matrice quadrata di ordine 2, le cui righe sono le coordinate cartesiane degli estremi del segmento. 2) L'output non è un oggetto grafico, anzi *Mathematica* restituisce l'input. Per visualizzare il segmento dobbiamo utilizzare l'istruzione **Graphics:**

```
Graphics[segmento]
```



È chiaro che per velocizzare il codice, si scrive direttamente (indendando il codice, in modo da renderlo più leggibile):

```
Graphics[
 (*segmento di estremi (2,1), (0,-1)*)
 Line[{{2, 1}, {0, -1}}]
]
```



L'istruzione **Graphics** ha molte opzioni. Per visualizzarle utilizziamo l'help in linea:

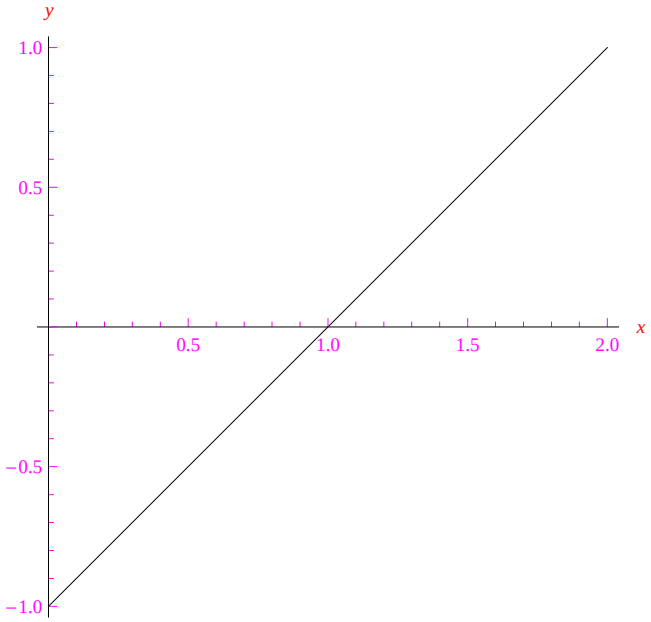
```
? Graphics
```

Graphics[*primitives*, *options*] represents a two-dimensional graphical image. ≫

Ad esempio, possiamo includere gli assi coordinati, dopo averli "abbellitti" con le seguenti opzioni

```
SetOptions[
  {
   Graphics,
   Plot
  },
  TicksStyle -> Directive[Hue[5 / 6], 9]
 ];
```

```
Graphics[
 (*segmento di estremi (2,1), (0,-1)*)
 Line[{{2, 1}, {0, -1}}],
 (*opzioni*)
 Axes → True,
 AxesLabel →
  {
   Style["x", Small, Red],
   Style["y", Small, Red]
  }
]
```

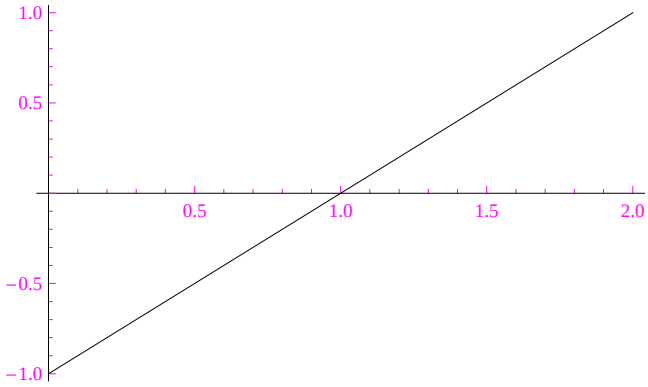


Si noti che allo stesso risultato si giunge con l'istruzione **Plot** includente **Epilog**, a patto che l'argomento di **Plot** sia la funzione **Null**

```
? Epilog
```

Epilog is an option for graphics functions which gives a list of
graphics primitives to be rendered after the main part of the graphics is rendered. ≫

```
Plot[
 Null,
 {x, 0, 2},
 Epilog →
  {
   Line[{{2, 1}, {0, -1}}]
  }
]
```



Ritornando a **Graphics** possiamo inserire gli estremi del segmento, dopo averli dichiarati.

```
A = Point[{2, 1}]
```

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

Questo output non ci deve sorprendere, poiché il comando **Point** và processato da **Graphics**

```
Clear[A]
```

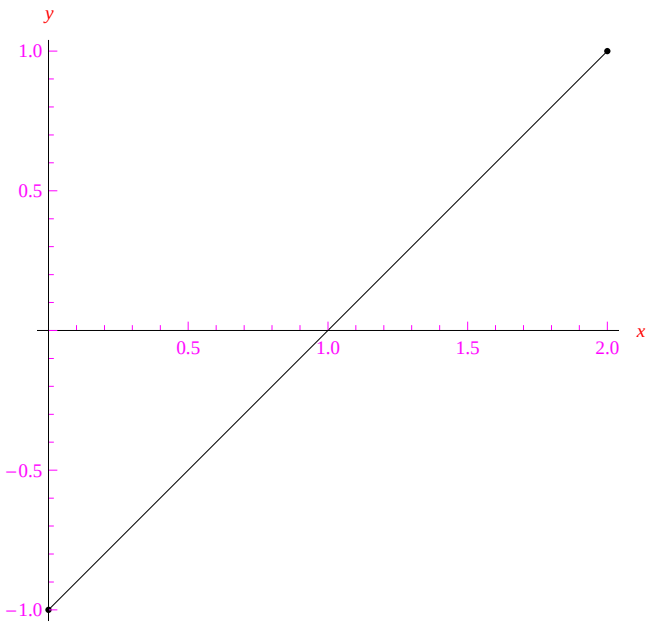
```
A = Graphics[Point[{2, 1}]]
```

Allo stesso modo:

```
B = Graphics[Point[{0, -1}]]
```


Anche in questo caso per risparmiare sul codice, scriviamo in un unica istruzione, ricordando che la lista di primitive deve essere racchiusa tra le parentesi graffe.

```
Graphics[
 {
  Line[{{2, 1}, {0, -1}}],
  Point[{2, 1}],
  Point[{0, -1}]
 },
 (*opzioni*)
 Axes → True,
 AxesLabel →
  {
   Style["x", Small, Red],
   Style["y", Small, Red]
  }
]
```

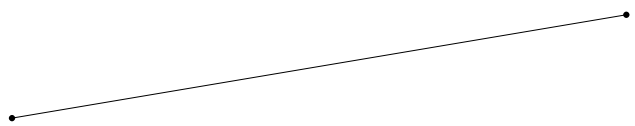


Un altro comando estremamente potente è **Show** (e relative opzioni):

```
? Show
```

Show[*graphics*, *options*] shows graphics with the specified options added.
Show[g_1, g_2, …] shows several graphics combined. ≫

```
Show[
 {
  Graphics[Line[{{2, 1}, {0, -1}}]],
  Graphics[Point[{2, 1}]],
  Graphics[Point[{0, -1}]]
 },
 PlotRange → All,
 AspectRatio → 0.2

]
```



Vediamo altre direttive grafiche. Rettangolo di vertice inferiore sinistro (x_1, y_1) e vertice superiore destro (x_2, y_2), con colore di default nero:

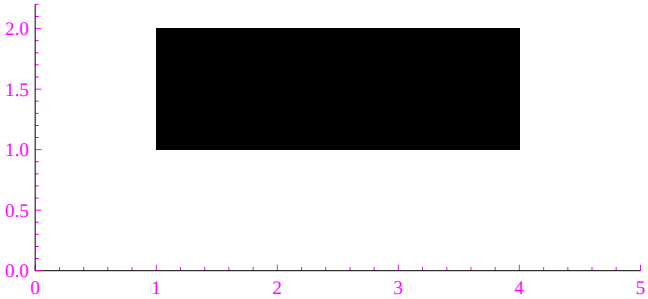
```
Rectangle[{x1, y1}, {x2, y2}]
```

```
Rectangle[{x1, y1}, {x2, y2}]
```

Al solito, dobbiamo processare il comando con l'istruzione **Graphics**

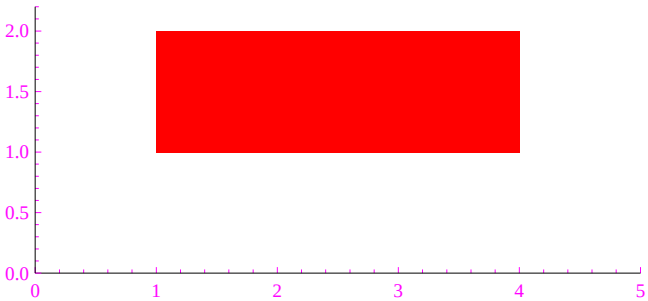
```
Graphics[
 Rectangle[{1, 1}, {4, 2}]
]
```

```
Graphics[
 Rectangle[{1, 1}, {4, 2}],
 PlotRange → {{0, 5}, {0, 2.2}},
 Axes → True
]
```



Per colorare lo sfondo delle primitive del tipo **Rectangle**, si possono specificare i colori (**Red**,**Yellow**,**Green**,**Magenta**, etc.) nelle opzioni di **Graphics**

```
Graphics[
 {
  Red,
  Rectangle[{1, 1}, {4, 2}]
 },
 PlotRange → {{0, 5}, {0, 2.2}},
 Axes → True
]
```



I predetti colori possono essere invocati dall'istruzione **RGBColor**

```
? RGBColor
```

RGBColor[*red*, *green*, *blue*] is a graphics directive which
specifies that objects which follow are to be displayed, if possible, in the color given.
RGBColor[*r*, *g*, *b*, *a*] specifies opacity *a*.
≫

oppure da **Hue**

```
? Hue
```

Hue[*h*] is a graphics directive which specifies that objects
which follow are to be displayed, if possible, in a color corresponding to hue *h*.
Hue[*h*, *s*, *b*] specifies colors in terms of hue, saturation and brightness.
Hue[*h*, *s*, *b*, *a*] specifies opacity *a*. ≫

Circonferenza di centro (x_0, y_0) e raggio R

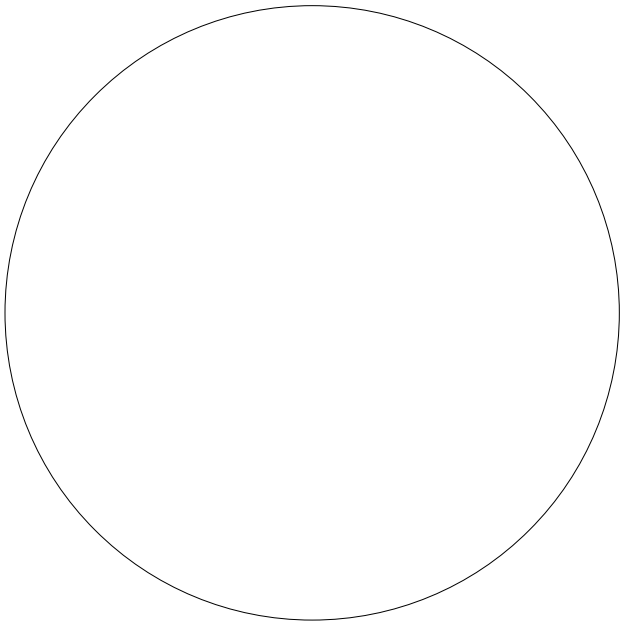
```
Circle[{x0, y0}, R]
```

```
Circle[{x0, y0}, R]
```

Possiamo definire una funzione i cui argomenti sono le coordinate del centro e il raggio

```
crf[x0_, y0_, R_] := Circle[{x0, y0}, R] // Graphics
```

```
crf[1, 1, 1]
```

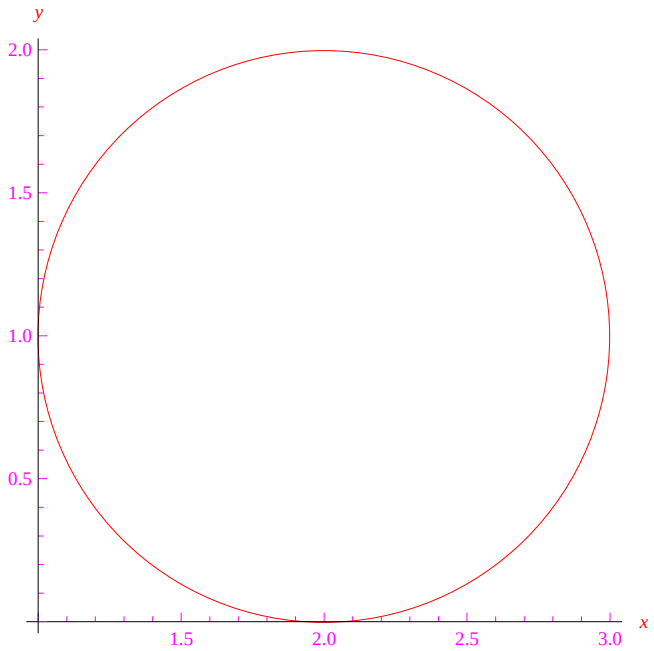


Miglioriamone l'aspetto

```
Clear[crf]
```

```
crf[x0_, y0_, R_] := Graphics[
  {
   Red,
   Circle[{x0, y0}, R]
  },
  Axes → True,
  AxesLabel →
   {
    Style["x", Small, Red],
    Style["y", Small, Red]
   }
 ]
```

```
crf[2, 1, 1]
```



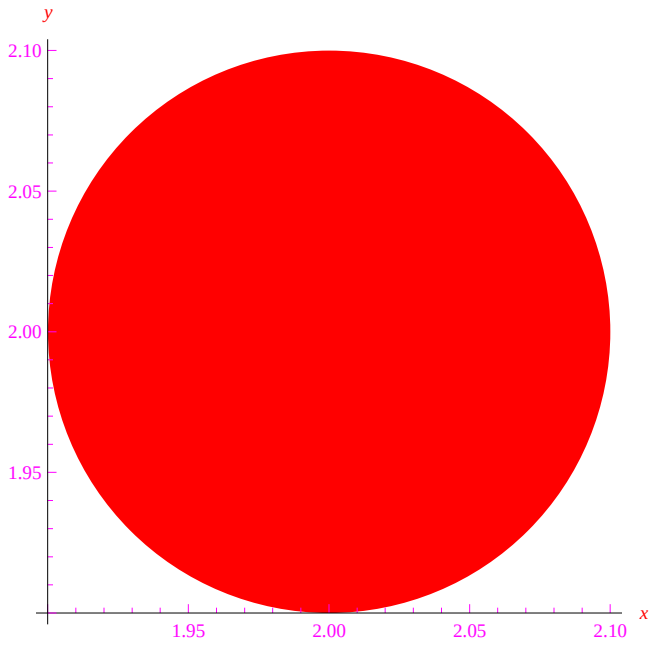
Per il cerchio di centro (x_0, y_0) e raggio R

```
Disk[{x0, y0}, R]
```

```
Disk[{x0, y0}, R]
```

Quindi

```
Graphics[
 {
  Red,
  Disk[{2, 2}, 0.1]
 },
 Axes → True,
 AxesLabel →
  {
   Style["x", Small, Red],
   Style["y", Small, Red]
  }
]
```

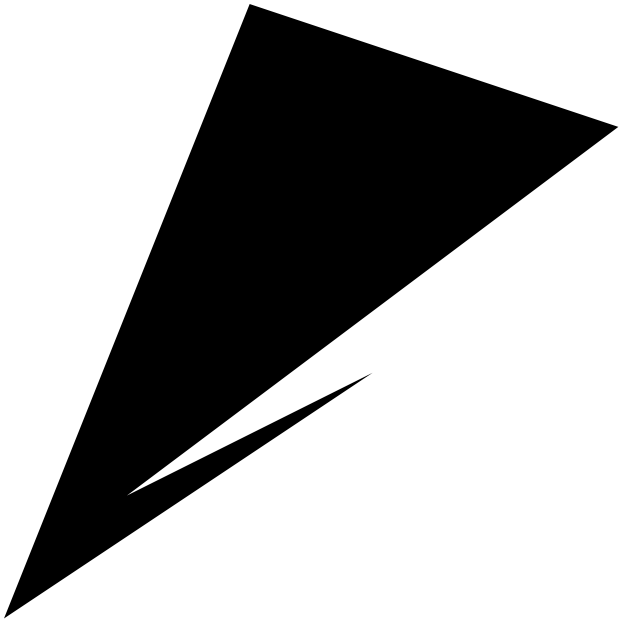


<u>Poligono di vertici</u> V_1, V_2, ..., V_N

```
Polygon[{x1, x2}, ...]
```

Esempio:

```
Polygon[{{0, 1}, {2, 2}, {-1, 0}, {1, 5}, {4, 4}}] // Graphics
```



Un esempio più complicato che richiede l'utilizzo di **RandomReal** e di **EdgeForm**

```
? RandomReal
```

RandomReal[] gives a pseudorandom real number in the range 0 to 1.
RandomReal[{x_{min}, x_{max}}] gives a pseudorandom real number in the range x_{min} to x_{max}.
RandomReal[x_{max}] gives a pseudorandom real number in the range 0 to x_{max}.
RandomReal[*range*, *n*] gives a list of *n* pseudorandom reals.
RandomReal[*range*, {n_1, n_2, …}] gives an $n_1 \times n_2 \times \ldots$ array of pseudorandom reals.
RandomReal[*dist*, …] samples from the symbolic continuous distribution *dist*. ≫

```
? EdgeForm
```

EdgeForm[*g*] is a graphics directive which specifies that edges of polygons and other filled graphics objects are to be drawn using the graphics directive or list of directives *g*. ≫

Generiamo una lista di coppie ordinate di numeri reali appartenenti all'intervallo [0, 1]

```
RandomReal[1, {3, 2}]
```

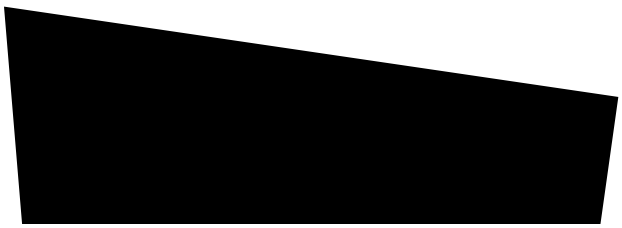
```
{{0.162392, 0.167663}, {0.473403, 0.700317}, {0.994582, 0.95529}}
```

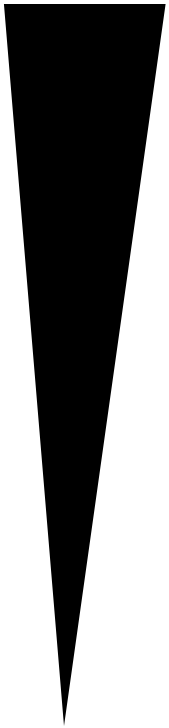
Le predette coppie ordinate possono essere le coordinate cartesiane dei vertici di un triangolo

```
triangolo = Polygon[RandomReal[1, {3, 2}]]
```

```
Polygon[{{0.0227777, 0.971765}, {0.216608, 0.943232}, {0.096318, 0.0884835}}]
```

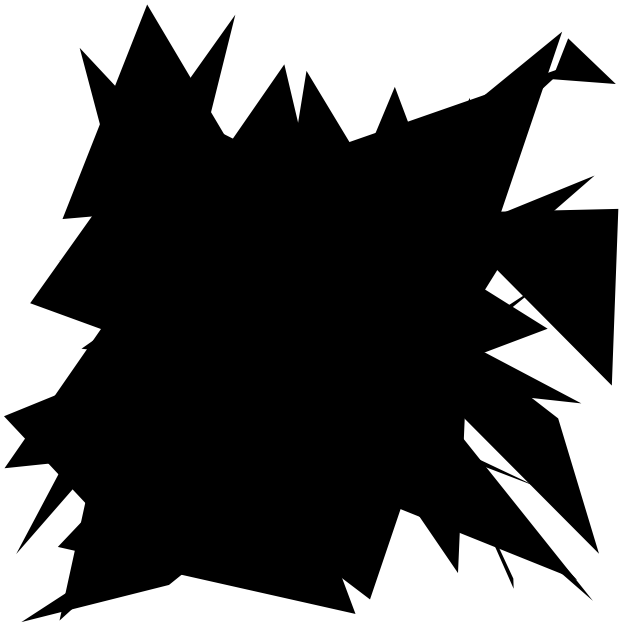
```
triangolo // Graphics
```





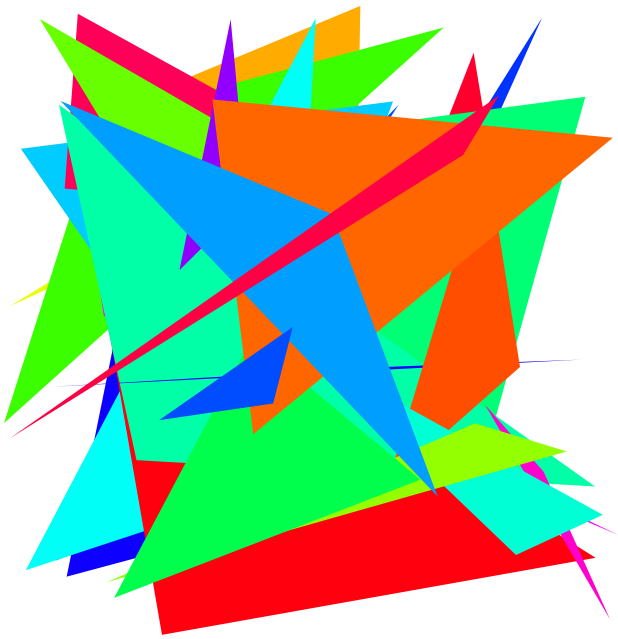
Costruiamo una lista di 30 triangoli random:

```
listatriangoli = Table[
   (*argomento principale*)
   Polygon[RandomReal[1, {3, 2}]],
   (*iterazione*)
   {30}
  ] // Graphics
```



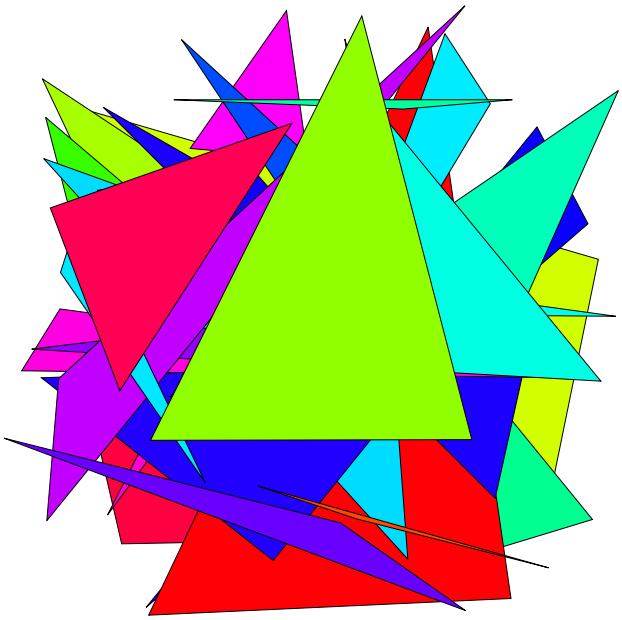
Coloriamo i vari triangoli in modalità random:

```
listatriangoli1 = Table[
   (*argomento principale*)
   {
    Hue[RandomReal[]],
    Polygon[RandomReal[1, {3, 2}]]
   },
   (*iterazione*)
   {30}
  ] // Graphics
```



Aggiungiamo un contorno black

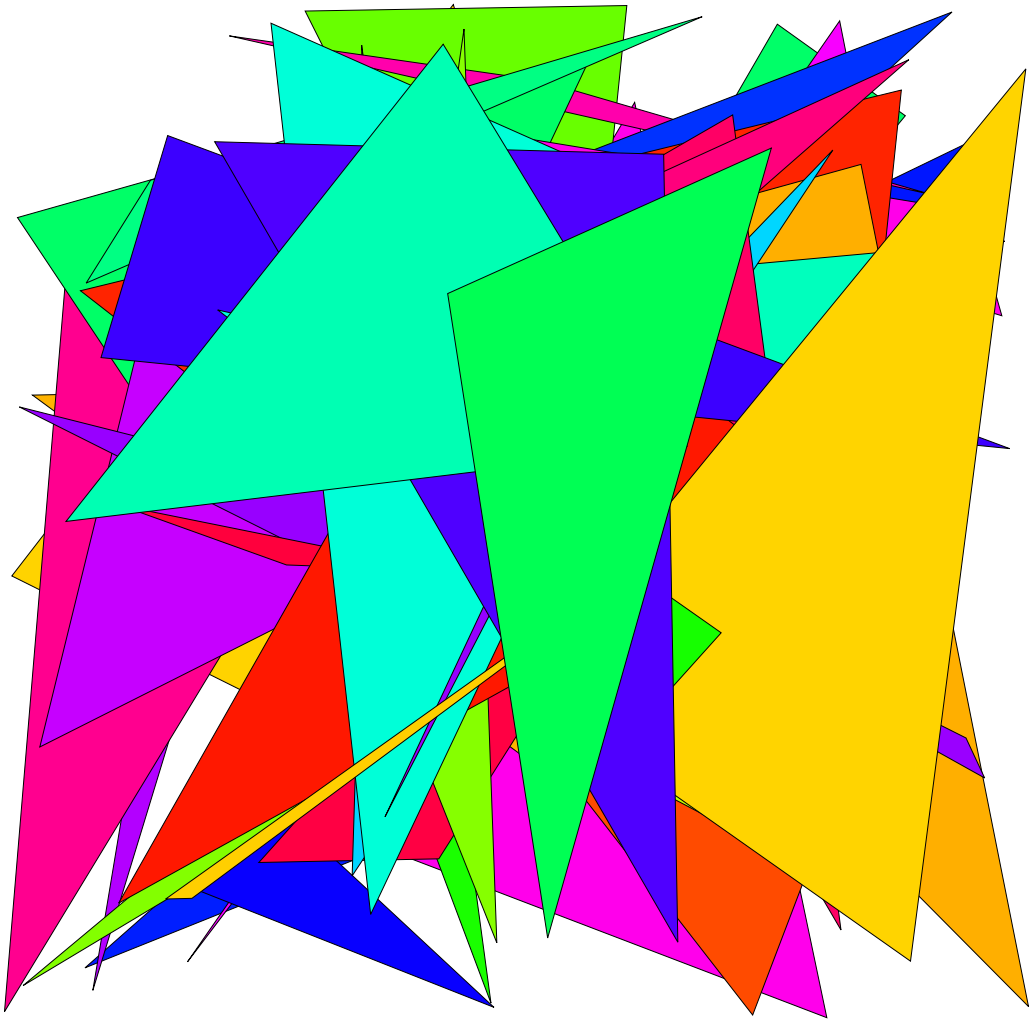
```
listatriangoli2 = Table[
   (*argomento principale*)
   {
    EdgeForm[Black],
    Hue[RandomReal[]],
    Polygon[RandomReal[1, {3, 2}]]
   },
   (*iterazione*)
   {30}
  ] // Graphics
```



Generiamo un'altra lista per poi esportarla come gif animata.

```
listatriangoli3[a_] := Graphics[
  Table[
   (*argomento principale*)
   {
    EdgeForm[Black],
    Hue[RandomReal[]],
    Polygon[RandomReal[1, {3, 2}]]
   },
   (*iterazione*)
   {n, a}
  ],
  ImageSize → {500, 500}
 ]
```

```
listatriangoli3[50]
```

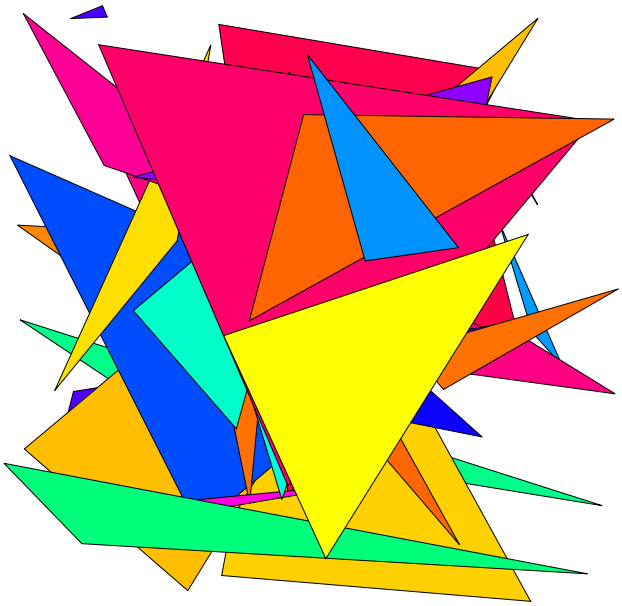


```
lista = Table[
   listatriangoli3[a],
   {a, 1, 50}
  ];
```

```
Export["triangolirandom.gif", lista]
```

triangolirandom.gif

```
Graphics[
 Table[
  {
   EdgeForm[{Black}],
   Hue[RandomReal[]],
   Polygon[RandomReal[1, {3, 2}]]
  },
  {30}
 ]
]
```



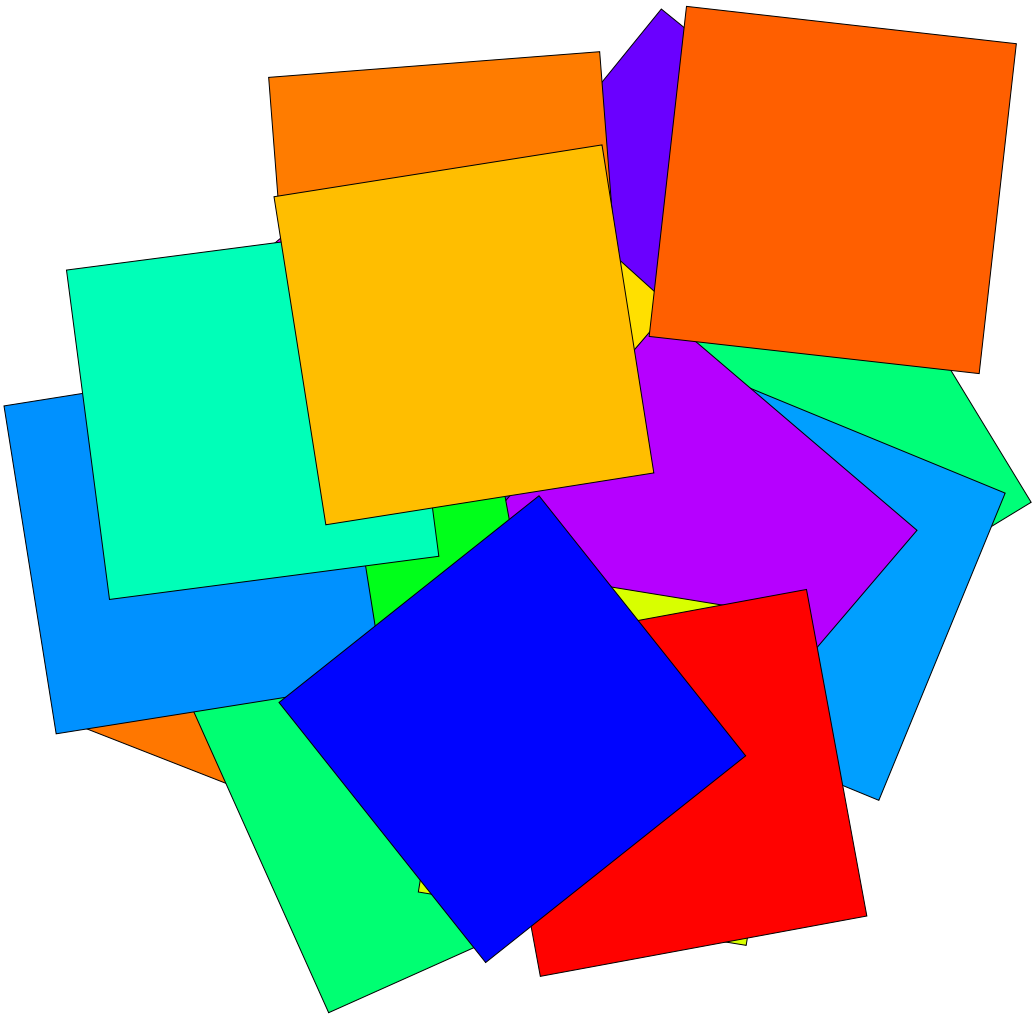
In maniera simile è possibile generare una lista di rettangoli/quadrati random, utilizzando il comando **Rotate**

```
? Rotate
```

Rotate[g, θ] represents 2D graphics primitives g
rotated counterclockwise by θ radians about the center of their bounding box.
Rotate[g, θ, {x, y}] rotates 2D graphics primitives about the point {x, y}.
Rotate[g, θ, w] rotates 3D graphics primitives by θ radians around the 3D vector w anchored at the origin.
Rotate[g, θ, w, p] rotates around the 3D vector w anchored at p.
Rotate[g, {u, v}] rotates around the origin transforming the 3D vector u to v.
Rotate[g, θ, {u, v}] rotates by angle θ in the plane spanned by 3D vectors u and v. ≫

```
listarettangoli[a_] := Graphics[
  {
   EdgeForm[Black],
   Table[
    {
     Hue[RandomReal[]],
     Rotate[Rectangle[RandomReal[2, 2]],
      RandomReal[2 Pi]]
    },
    {n, a}
   ]
  },
  ImageSize → {500, 500}
 ]
```

```
listarettangoli[20]
```



Generiamo una animazione grafica in formato gif:

```
lista1 = Table[
   listarettangoli[a],
   {a, 1, 50}
  ];
```

```
Export["listarettangoli.gif", lista1];
```

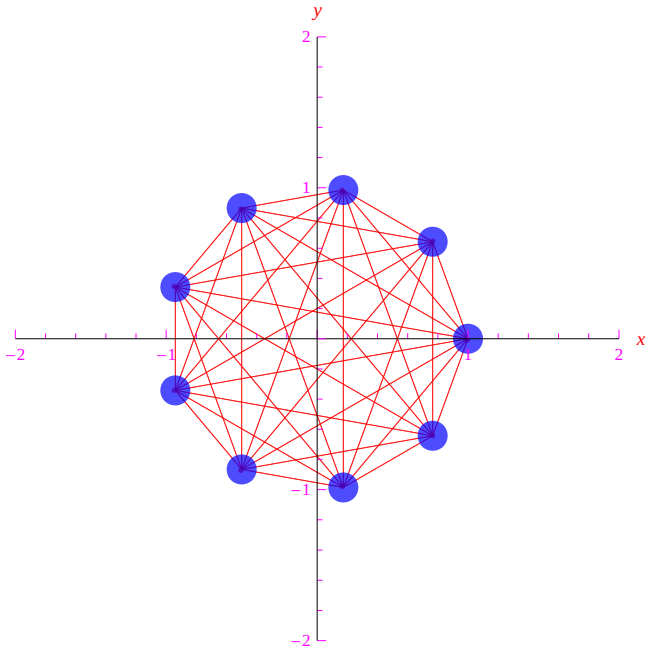
Concludiamo questa parte, modificando uno script preso dalla guida in linea per la generazione di un grafo. *Mathematica* dispone di un comando opportuno per generare tali enti topologici, ma qui è utilizzata la semplice direttiva bidimensionale **Graphics** e l'istruzione **Tuples**

```
? Tuples
```

Tuples[*list*, *n*] generates a list of all possible *n*-tuples of elements from *list*.
Tuples[{$list_1$, $list_2$, …}] generates a list of all possible tuples whose i^{th} element is from $list_i$. ≫

```
SetOptions[
  {
   Graphics
  },
  TicksStyle -> Directive[
    Hue[5 / 6],
    8
   ]
 ];

pts[a_] :=
  Table[
   {
    Cos[8 n π / 9],
    Sin[8 n π / 9]
   },
   {n, 0, a}
  ];

graf[a_] := Show[
  Graphics[
   {
    Opacity[0.7],
    Red,
    Line[Tuples[pts[a], 2]],
    Blue,
    PointSize[0.05],
    Point[pts[a]]
   },
   PlotRange → {{-2, 2}, {-2, 2}},
   Axes → True,
   AxesLabel → {
     Style["x", Small, Red],
     Style["y", Small, Red]
    }
  ]
 ]
```

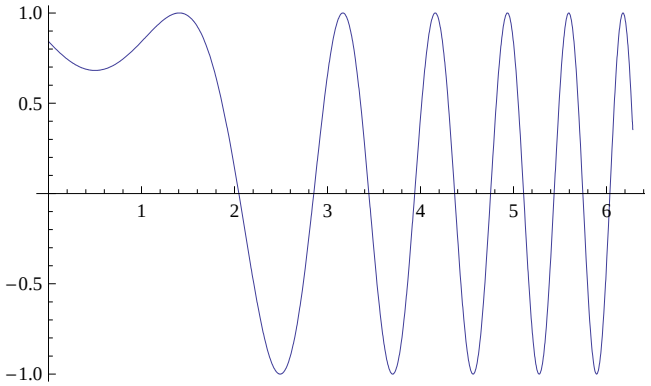
```
graf[8]
```



■ L'istruzione Plot

Ci proponiamo di graficare una funzione reale f della variabile reale x. A tale scopo, utilizziamo l'istruzione **Plot** che implementa un algoritmo adattivo consistente nell'utilizzo di un numero sufficiente di punti del grafico $y = f(x)$, in modo da disegnare una curva sufficientemente "liscia" (a patto che f sia sufficientemente regolare). Supponiamo che

```
f[x_] := Sin[x^2 - x + 1]

grafico = Plot[
  (*espressione della funzione*)
  f[x],
  (*intervallo della restrizione di f*)
  {x, 0, 2 π}
 ]
```



Il grafico è piuttosto scarno, cerchiamo quindi di migliorarne l'aspetto. È preferibile innanzitutto assegnare definitivamente le opzioni grafiche attraverso l'istruzione

```
? SetOptions
```

SetOptions[*s*, *name*$_1$ –> *value*$_1$, *name*$_2$ –> *value*$_2$, …] sets the specified default options for a symbol *s*.
SetOptions[*stream*, …] or SetOptions["*name*", …] sets options associated with a particular stream.
SetOptions[*object*, …] sets options associated with an external object such as a NotebookObject. ≫

```
SetOptions[
  Plot,
  TicksStyle → Directive[Hue[5 / 6], 8]
 ];
```

Analizziamo questo codice, rimandando all'help in linea.

```
? TicksStyle
```

TicksStyle is an option for graphics functions which specifies how ticks should be rendered. ≫

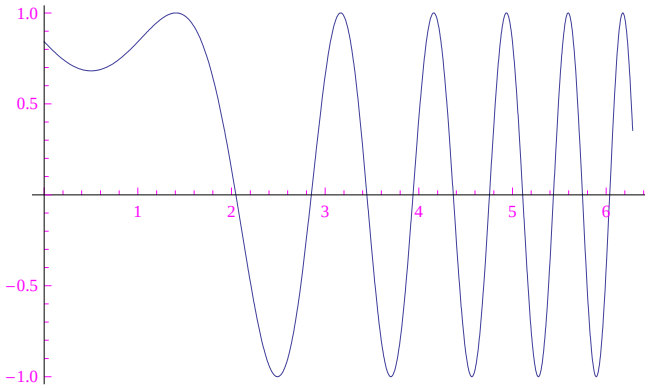
```
? Directive
```

Directive[g_1, g_2, …] represents a single graphics directive composed of the directives g_1, g_2, …. ≫

Rigrafichiamo

```
Clear[grafico]
```

```
grafico = Plot[
  (*espressione della funzione*)
  f[x],
  (*intervallo della restrizione di f*)
  {x, 0, 2 π}
 ]
```



Possiamo specificare i valori dell'ascissa e dell'ordinata $y = f(x)$, per poi specificare gli assi coordinati con l'istruzione **AxesLabel** che incorpora

```
?Style
```

Style[*expr*, *options*] displays with *expr* formatted using the specified option settings.
Style[*expr*, "*style*"] uses the option settings for the specified style in the current notebook.
Style[*expr*, *color*] displays using the specified color.
Style[*expr*, Bold] displays with fonts made bold.
Style[*expr*, Italic] displays with fonts made italic.
Style[*expr*, Underlined] displays with fonts underlined.
Style[*expr*, Larger] displays with fonts made larger.
Style[*expr*, Smaller] displays with fonts made smaller.
Style[*expr*, *n*] displays with font size *n*.
Style[*expr*, Tiny], Style[*expr*, Small], etc. display with fonts that are tiny, small, etc. ≫

e **Text**

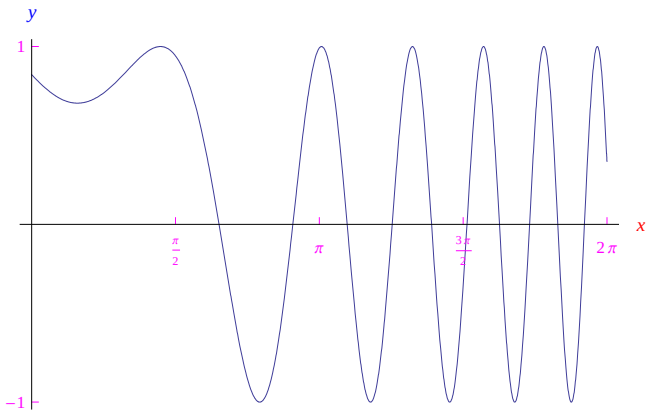
```
?Text
```

Text[*expr*] displays with *expr* in plain text format.
Text[*expr*, *coords*] is a graphics primitive
that displays the textual form of *expr* centered at the point specified by *coords*. ≫

Quindi

```
(*resetto la variabile grafico*)

Clear[grafico]

(*traccio il grafico*)
```

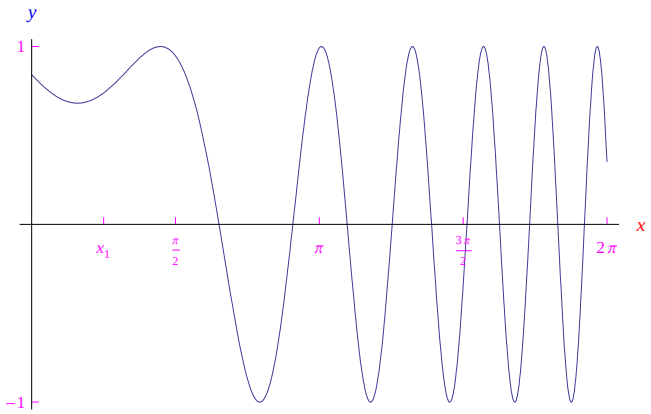
```
grafico = Plot[
  (*espressione della funzione*)
  f[x],
  (*intervallo della restrizione di f*)
  {x, 0, 2 π},
  (*denominazione degli assi coordinati*)
  AxesLabel →
   {
    (*formattazione asse x*)
    Style["x", Small, Red],
    Style["y", Small, Blue]
   },
  (*valori dell'ascissa e dell'ordinata*)
  Ticks ->
   {
    (*asse x*)
    {
     π/2, π, 3 π/2, 2 π
    },
    (*asse y*)
    {
     -1, 1
    }
   }
 ]
```



Volendo "labellare" un'ascissa, ad esempio vogliamo riportare sull'asse x il valore $x_1 = \frac{\pi}{4}$

```
Clear[grafico]
```

```
grafico = Plot[
  (*espressione della funzione*)
  f[x],
  (*intervallo della restrizione di f*)
  {x, 0, 2 π},
  (*denominazione degli assi coordinati*)
  AxesLabel →
   {
    (*formattazione asse x*)
    Style["x", Small, Red],
    Style["y", Small, Blue]
   },
  (*valori dell'ascissa e dell'ordinata*)
  Ticks ->
   {
    (*asse x*)
    {
     π/2, {π/4, "x1"}, π, 3 π/2, 2 π
    },
    (*asse y*)
    {
     -1, 1
    }
   }
 ]
```



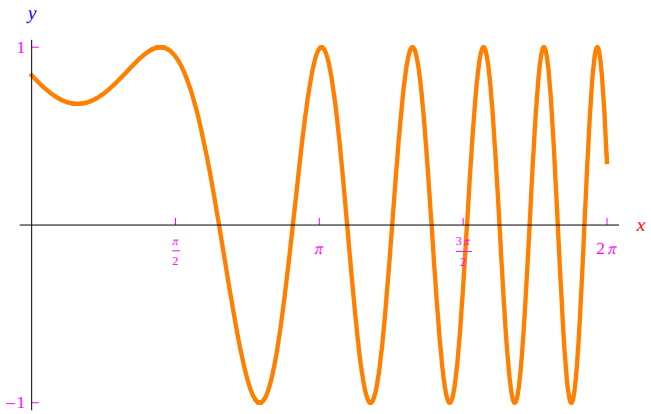
Possiamo agire sulla curva, colorandola diversamente o ispessendola o magari disegnandola a tratteggio, utilizzando **PlotStyle**:

```
?PlotStyle
```

PlotStyle is an option for plotting and related functions that specifies styles in which objects are to be drawn. ≫

```
Clear[grafico]
```

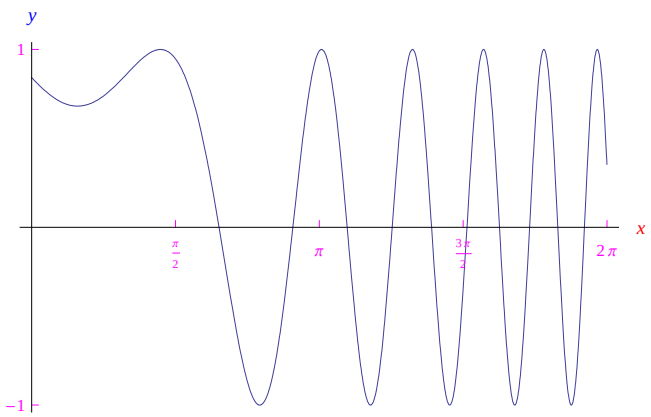
```
grafico = Plot[
  (*espressione della funzione*)
  f[x],
  (*intervallo della restrizione di f*)
  {x, 0, 2 π},
  (*modifichiamo il colore della curva, ispessendola*)
  PlotStyle → Directive[Orange, Thick],
  (*denominazione degli assi coordinati*)
  AxesLabel →
   {
    (*formattazione asse x*)
    Style["x", Small, Red],
    Style["y", Small, Blue]
   },
  (*valori dell'ascissa e dell'ordinata*)
  Ticks ->
   {
    (*asse x*)
    {
     π/2, π, 3 π/2, 2 π
    },
    (*asse y*)
    {
     -1, 1
    }
   }
 ]
```



Thicks è una lista, per cui può essere definita a parte:

```
assex = {π/2, π, 3 π/2, 2 π}; assey = {-1, 1};
Clear[grafico]
```

```
grafico = Plot[
  (*espressione della funzione*)
  f[x],
  (*intervallo della restrizione di f*)
  {x, 0, 2 π},
  (*denominazione degli assi coordinati*)
  AxesLabel →
   {
    (*formattazione asse x*)
    Style["x", Small, Red],
    Style["y", Small, Blue]
   },
  (*valori dell'ascissa e dell'ordinata*)
  Ticks →
   {
    assex,
    assey
   }
 ]
```



Se ci sono molti valori, conviene utilizzare uno dei generatori di liste visti nella sezione "Liste".

```
Clear[assex, assey]
```

```
assex[xmin_, xmax_, step_] := Range[Round[xmin], Round[xmax], step]
```

```
assey[ymin_, ymax_, step_] := Range[Round[ymin], Round[ymax], step]
```

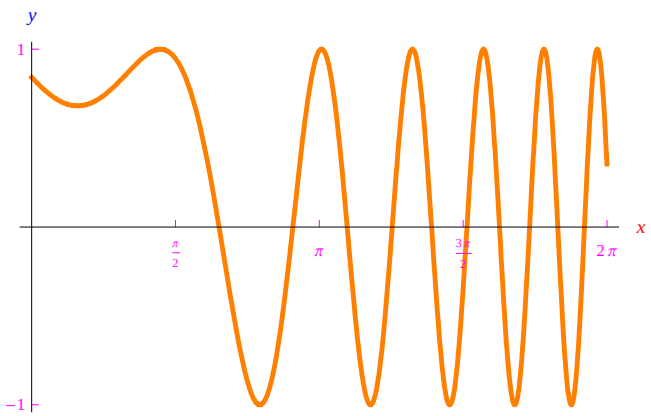
```
assex[0, 10 π, 3]
```

```
{0, 3, 6, 9, 12, 15, 18, 21, 24, 27, 30}
```

Possiamo agire sullo spessore con **Thickness**

```
Clear[grafico]
```

```
grafico = Plot[
  (*espressione della funzione*)
  f[x],
  (*intervallo della restrizione di f*)
  {x, 0, 2 π},
  (*modifichiamo il colore della curva, ispessendola*)
  PlotStyle → Directive[Orange, Thickness[0.008]],
  (*denominazione degli assi coordinati*)
  AxesLabel →
   {
    (*formattazione asse x*)
    Style["x", Small, Red],
    Style["y", Small, Blue]
   },
  (*valori dell'ascissa e dell'ordinata*)
  Ticks ->
   {
    (*asse x*)
    {
     π/2, π, 3 π/2, 2 π
    },
    (*asse y*)
    {
     -1, 1
    }
   }
 ]
```



La formattazione degli assi è controllata dall'istruzione **AxesStyle**

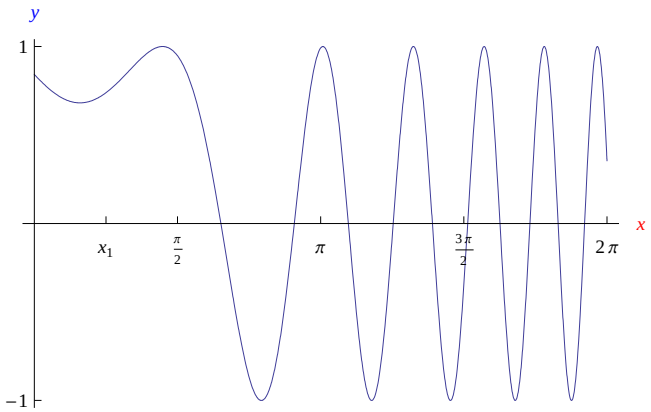
```
? AxesStyle
```

AxesStyle is an option for graphics functions which specifies how axes should be rendered. ≫

Si badi che tale istruzione potrebbe andare in conflitto con i valori settati in **SetOptions**, per cui chiudiamo il Kernel e ridefiniamo

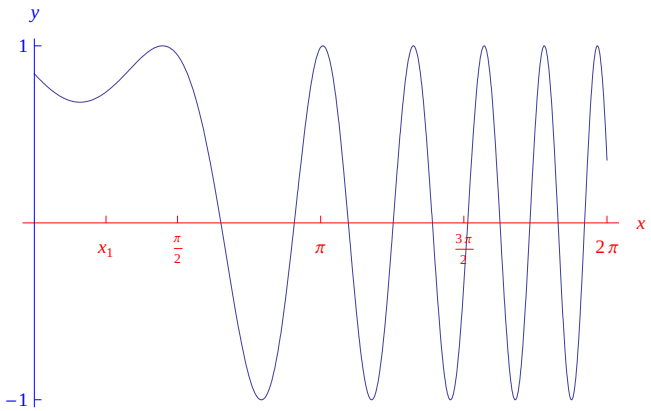
```
f[x_] := Sin[x^2 - x + 1]
```

```
grafico = Plot[
  (*espressione della funzione*)
  f[x],
  (*intervallo della restrizione di f*)
  {x, 0, 2 π},
  (*denominazione degli assi coordinati*)
  AxesLabel →
   {
    (*formattazione asse x*)
    Style["x", Small, Red],
    Style["y", Small, Blue]
   },
  (*valori dell'ascissa e dell'ordinata*)
  Ticks ->
   {
    (*asse x*)
    {
     π/2, {π/4, "x1"}, π, 3 π/2, 2 π
    },
    (*asse y*)
    {
     -1, 1
    }
   }
 ]
```



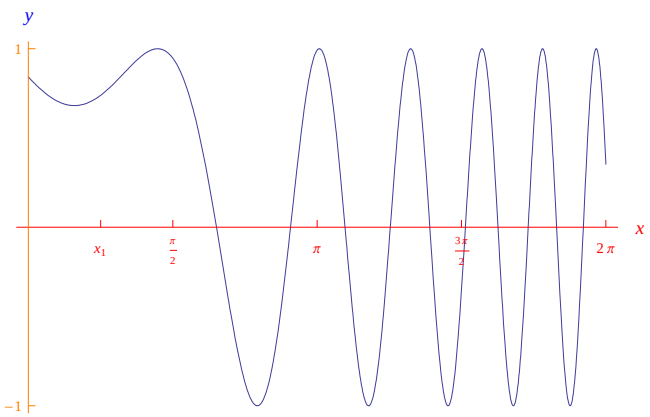
Utilizzando **AxesStyle**

```
grafico = Plot[
  (*espressione della funzione*)
  f[x],
  (*intervallo della restrizione di f*)
  {x, 0, 2 π},
  (*denominazione degli assi coordinati*)
  AxesLabel →
   {
    (*formattazione asse x*)
    Style["x", Small, Red],
    Style["y", Small, Blue]
   },
  (*valori dell'ascissa e dell'ordinata*)
  Ticks ->
   {
    (*asse x*)
    {
     π/2, {π/4, "x1"}, π, 3 π/2, 2 π
    },
    (*asse y*)
    {
     -1, 1
    }
   },
  AxesStyle → {
    Red,
    Blue
   }
 ]
```



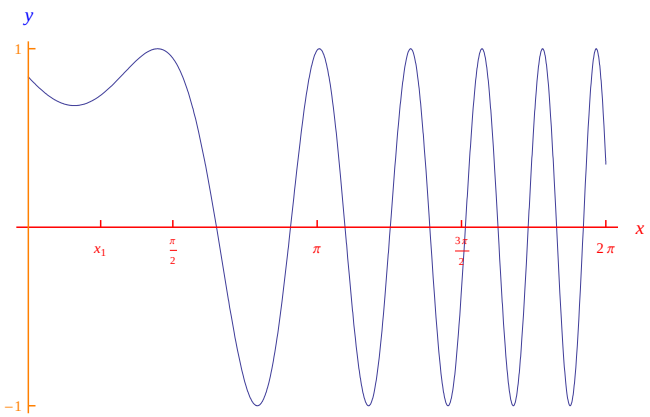
volendo modificare la dimensione dei font:

```
grafico = Plot[
  (*espressione della funzione*)
  f[x],
  (*intervallo della restrizione di f*)
  {x, 0, 2 π},
  (*denominazione degli assi coordinati*)
  AxesLabel →
   {
    (*formattazione asse x*)
    Style["x", Small, Red],
    Style["y", Small, Blue]
   },
  (*valori dell'ascissa e dell'ordinata*)
  Ticks ->
   {
    (*asse x*)
    {
     π/2, {π/4, "x1"}, π, 3 π/2, 2 π
    },
    (*asse y*)
    {
     -1, 1
    }
   },
  AxesStyle →
   {
    Directive[Red, 7],
    Directive[Orange, 7]
   }
 ]
```



o lo spessore

```
grafico = Plot[
  (*espressione della funzione*)
  f[x],
  (*intervallo della restrizione di f*)
  {x, 0, 2 π},
  (*denominazione degli assi coordinati*)
  AxesLabel →
   {
    (*formattazione asse x*)
    Style["x", Small, Red],
    Style["y", Small, Blue]
   },
  (*valori dell'ascissa e dell'ordinata*)
  Ticks ->
   {
    (*asse x*)
    {
     π/2, {π/4, "x1"}, π, 3 π/2, 2 π
    },
    (*asse y*)
    {
     -1, 1
    }
   },
  AxesStyle →
   {
    Directive[Red, 7, Thickness[0.00255]],
    Directive[Orange, 7, Thickness[0.00255]]
   }
 ]
```

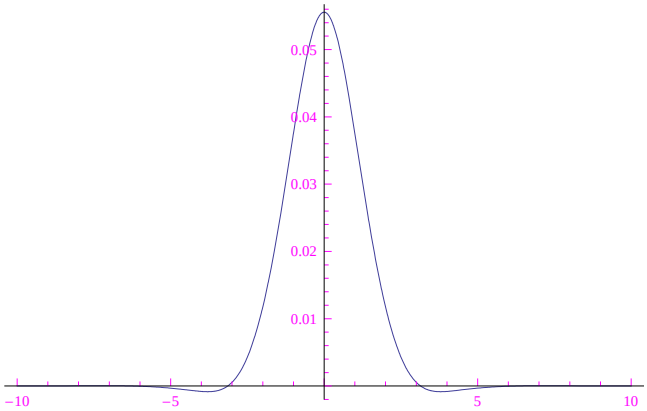


Un'opzione importante è **PlotRange** che per default è settata su **Automatic**

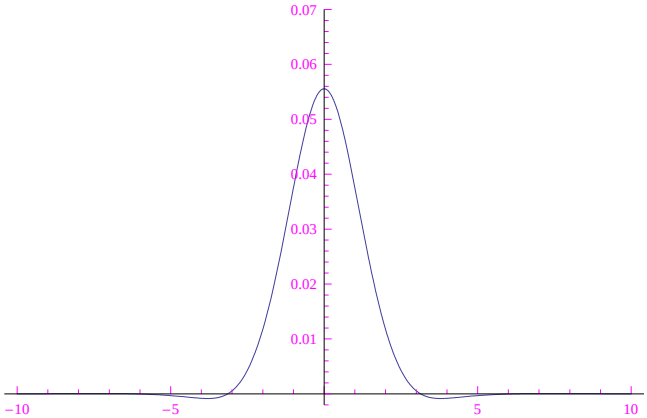
```
? PlotRange
```

PlotRange is an option for graphics functions that specifies what range of coordinates to include in a plot. ≫

```
Plot[
 (x Sin[Sin[x]] - Sin[x]^2)/x^6,
 {x, -10, 10}
]
```




```
Plot[
 (x Sin[Sin[x]] - Sin[x]^2)/x^6,
 {x, -10, 10},
 PlotRange → {-0.002, 0.07}
]
```



Per le altre opzioni, in particolare per l'utilizzo di scale logaritmiche, si rimanda all'help in linea.

```
? Plot
```

Plot[f, {x, x_{min}, x_{max}}] generates a plot of f as a function of x from x_{min} to x_{max}.
Plot[{f_1, f_2, …}, {x, x_{min}, x_{max}}] plots several functions f_i. ≫

■ Studio di funzione

In questo numero studiamo il grafico di una funzione, in modo da conoscere i vari comandi utilizzati nel Calcolo differenziale.

```
Clear[f]
```

```
f[x_] := Log[x + 1] / (x - 1)
```

- **Insieme di definizione**. La funzione è definita in $X = (-1, 1) \bigcup (1, +\infty)$.

- **Comportamento agli estremi**

```
(*calcolo del limite per x→-1⁺*)

Limit[
 (*funzione*)
 f[x],
 (*punto di accumulazione*)
 x → -1,
 (*direzione di avvicinamento*)
 Direction → -1
]
```

∞

```
(*comportamento in un intorno di x=1*)

Limit[
 f[x],
 x → 1,
 (*limite sinistro*)
 Direction → 1
]
```

$-\infty$

```
Limit[
 f[x],
 x → 1,
 (*limite destro*)
 Direction → -1
]
```

∞

```
(*comportamento all'infinito*)

Limit[
 f[x],
 x → +∞
]
```

0

- **Derivata prima**

```
Df[x_] = f'[x] // Simplify
```

$$\frac{-1 + x - (1 + x)\,\mathrm{Log}[1 + x]}{(-1 + x)^2\,(1 + x)}$$

Se proviamo a utilizzare il comando **Solve**

```
? 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*. ≫

```
Solve[
 (*equazione*)
 f'[x] == 0,
 (*incognita*)
 x
]
```

Solve::dinv :
The expression $(1+x)^x$ involves unknowns in more than one argument, so inverse functions cannot be used. ≫

$$\text{Solve}\left[\frac{1}{(-1+x)\,(1+x)} - \frac{\text{Log}[1+x]}{(-1+x)^2} == 0,\ x\right]$$

Forziamo la soluzione con il comando **Reduce**

```
? Reduce
```

Reduce[*expr*, *vars*] reduces the statement *expr* by solving equations or inequalities for *vars* and eliminating quantifiers.
Reduce[*expr*, *vars*, *dom*] does the reduction over
the domain *dom*. Common choices of *dom* are Reals, Integers and Complexes. ≫

```
Reduce[(*equazione*)
 f'[x] == 0,
 (*incognita*)
 x
]
```

$$x == -1 + e^{1+\text{ProductLog}\left[-\frac{2}{e}\right]}\ ||\ x == -1 + e^{1+\text{ProductLog}\left[-1,-\frac{2}{e}\right]}$$

che esprime la soluzione attraverso la funzione speciale **ProductLog**. Proviamo numericamente:

```
Reduce[f'[x] == 0, x] // N
```

$$x == -0.321655 + 1.44794\,i\ ||\ x == -0.321655 - 1.44794\,i$$

sembra che ci siano solo 2 soluzioni complesse coniugate. Specifichiamo il dominio:

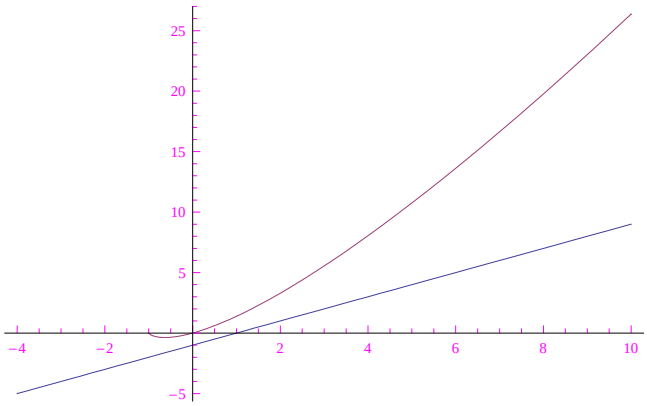
```
Reduce[
  f'[x] == 0,
  x,
  Reals
 ] // N
```

```
False
```

Ne consegue che la derivata è priva di zeri nel campo reale. Per un ulteriore conferma procediamo per via grafica plottando le funzioni $x-1$ e $(x+1)\ln(x+1)$

```
SetOptions[
  Plot,
  TicksStyle → Directive[Hue[5 / 6], 7]
 ];
```

```
Plot[
 {
  x - 1,
  (x + 1) Log[x + 1]
 },
 {x, -4, 10}
]
```



da cui vediamo che le curve non si intersecano. Per lo studio del segno, è immediato verificare dal grafico precedente che $f'(x) < 0$.

Calcoliamo la derivata seconda:

```
D2f[x_] = f''[x] // Simplify
```

$$\frac{1 + 2x - 3x^2 + 2(1+x)^2 \operatorname{Log}[1+x]}{(-1+x)^3 (1+x)^2}$$

Determiniamo gli zeri di $f''(x)$

```
Solve[D2f[x] == 0, x]
```

Solve::dinv :

The expression $(1+x)^{x^2}$ involves unknowns in more than one argument, so inverse functions cannot be used. ≫

$$\operatorname{Solve}\left[\frac{1 + 2x - 3x^2 + 2(1+x)^2 \operatorname{Log}[1+x]}{(-1+x)^3 (1+x)^2} == 0, x\right]$$

Qui abbiamo un problema simile a quello riscontrato nella ricerca degli zeri della derivata prima. Proviamo con **Reduce**

```
Reduce[D2f[x] == 0, x]
```

```
$Aborted
```

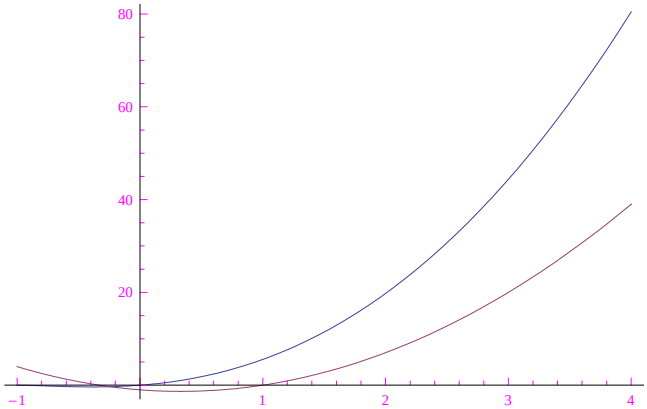
Dal momento che **Reduce** ha in questo caso un costo computazionale notevole, proviamo con **FindRoot**

```
? FindRoot
```

FindRoot[f, {x, x_0}] searches for a numerical root of f, starting from the point $x = x_0$.
FindRoot[*lhs* == *rhs*, {x, x_0}] searches for a numerical solution to the equation *lhs* == *rhs*.
FindRoot[{f_1, f_2, …}, {{x, x_0}, {y, y_0}, …}] searches for a simultaneous numerical root of all the f_i.
FindRoot[{eqn_1, eqn_2, …}, {{x, x_0}, {y, y_0}, …}] searches for a numerical solution to the simultaneous equations eqn_i. ≫

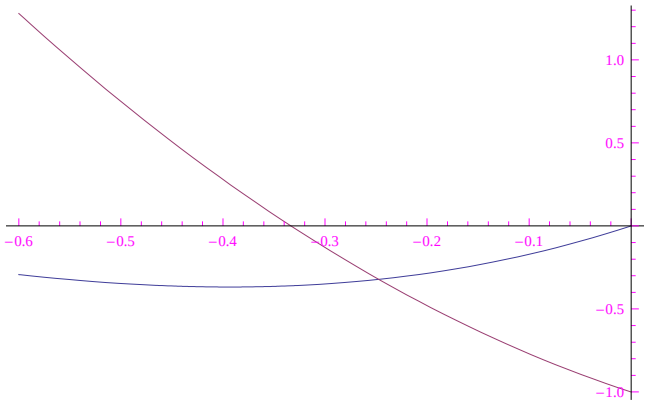
Dobbiamo quindi assegnare un punto di partenza. Allora è preferibile dapprima graficare le funzioni $y = 2(x+1)^2 \ln(x+1)$ e $y = 3x^2 - 2x - 1$

```
Plot[
 {
  (*funzioni*)
  2 (x + 1)^2 Log[x + 1], 3 x^2 - 2 x - 1
 },
 {x, -1, 4}
]
```



da cui vediamo una intersezione e quindi, una radice, nell'intervallo $(-1, 0)$. Restringiamo l'intervallo di plotting

```
Plot[
 {
  (*funzioni*)
  2 (x + 1)^2 Log[x + 1], 3 x^2 - 2 x - 1
 },
 {x, -0.6, 0}
]
```



Uno starting point è quindi $\xi_0 = -0.25$

```
sol = FindRoot[D2f[x] == 0, {x, -0.25}]
```

```
{x → -0.247306}
```

Verifichiamo

```
D2f[x /. sol]
```

1.00984×10^{-16}

Ne consegue che uno zero della derivata seconda è:

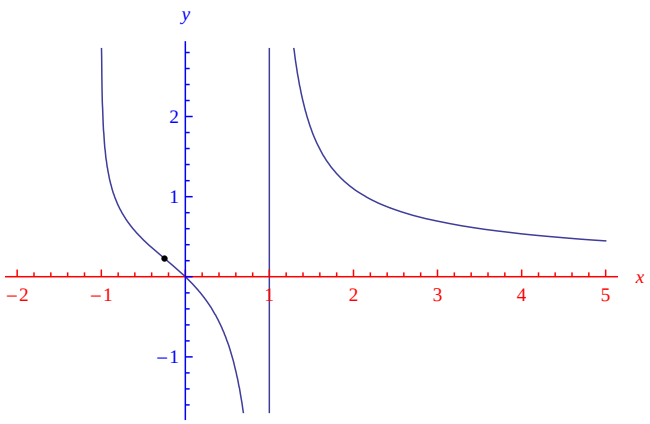
```
x0 = x /. sol
```

```
-0.247306
```

```
f[x0]
```

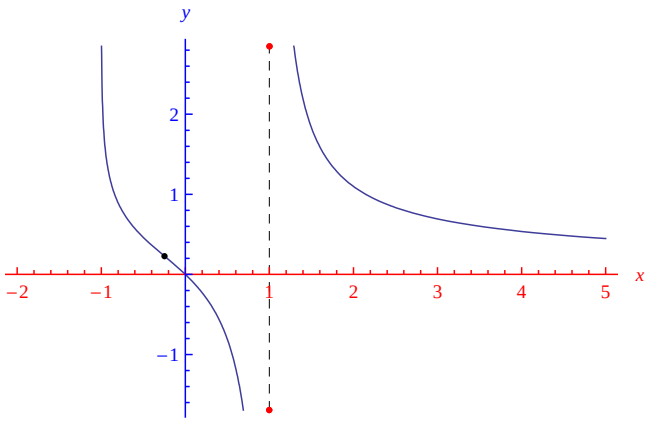
```
0.227768
```

```
graficofunzione0 = Plot[
  f[x],
  {x, -2, 5},
  PlotStyle → {
    Thickness[0.00255]
   },
  AxesLabel →
   {
    Style["x", Small, Red],
    Style["y", Small, Blue]
   },
  AxesStyle →
   {
    Directive[Red, Thickness[0.00255]],
    Directive[Blue, Thickness[0.00255]]
   },
  Epilog →
   {
    Red,
    Black,
    Point[{x0, f[x0]}]
   }
 ]
```



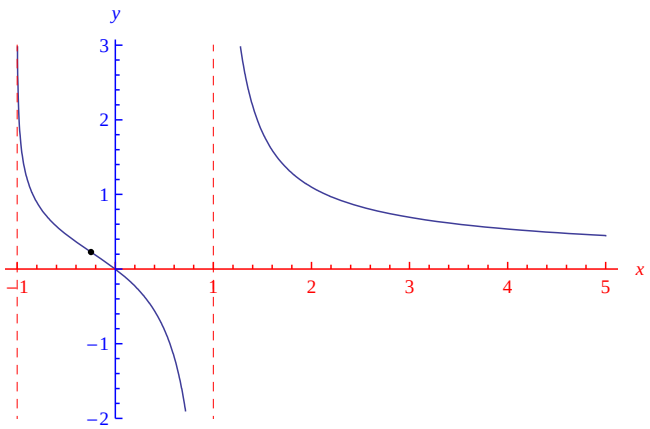
I preferibile utilizzare l'ozione **Exclusions**, altrimenti *Mathematica* traccia l'asintoto come parte del grafico. Utilizzando, poi, **ExclusionsStyle** possiamo ridisegnare l'asintoto con una formattazione adeguata (ad esempio, in tratteggio):

```
graficofunzione01 = Plot[
  f[x],
  {x, -2, 5},
  Exclusions → x == 1,
  ExclusionsStyle → {
    Dashed, Red
   },
  PlotStyle → {
    Thickness[0.00255]
   },
  AxesLabel →
   {
    Style["x", Small, Red],
    Style["y", Small, Blue]
   },
  AxesStyle →
   {
    Directive[Red, Thickness[0.00255]],
    Directive[Blue, Thickness[0.00255]]
   },
  Epilog →
   {
    Red,
    Black,
    Point[{x0, f[x0]}]
   }
 ]
```



Un aspetto migliore può essere ottenuto utilizzando **Epilog** che ingloba direttive grafiche bidimensionali:

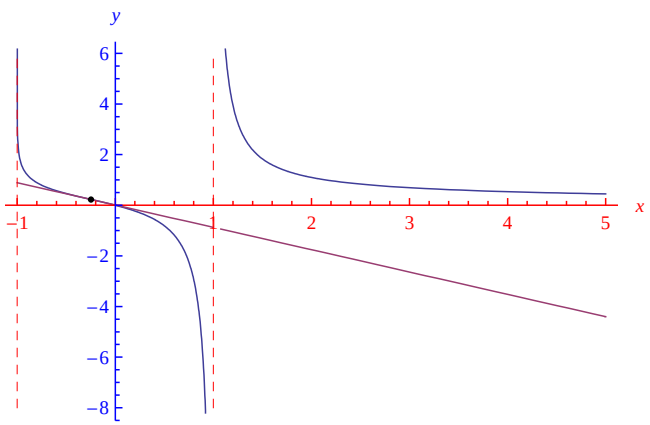
```
graficofunzione02 = Plot[
  f[x],
  {x, -1, 5},
  Exclusions → x == 1,
  PlotStyle → {
    Thickness[0.00255]
   },
  AxesLabel →
   {
    Style["x", Small, Red],
    Style["y", Small, Blue]
   },
  AxesStyle →
   {
    Directive[Red, Thickness[0.00255]],
    Directive[Blue, Thickness[0.00255]]
   },
  Epilog →
   {
    Red,
    Dashed,
    (*asintoto x-1=0*)
    Line[{{1, -3}, {1, 3}}],
    (*asintoto 2*)
    Line[{{-1, -3}, {-1, 3}}],
    Black,
    Point[{x0, f[x0]}]
   }
 ]
```



Equazione della tangente al grafico nel punto di flesso:

```
g[x_] := f[x0] + Df[x0] * (x - x0)
```

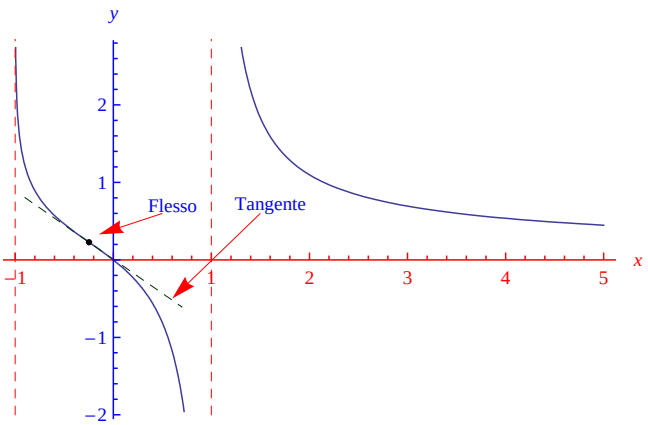
```
graficofunzione = Plot[
  {
   (*funzione*)
   f[x],
   (*tangente*)
   g[x]
  },
  {x, -1, 5},
  Exclusions → x == 1,
  PlotStyle → {
    Thickness[0.00255]
   },
  AxesLabel →
   {
    Style["x", Small, Red],
    Style["y", Small, Blue]
   },
  AxesStyle →
   {
    Directive[Red, Thickness[0.00255]],
    Directive[Blue, Thickness[0.00255]]
   },
  Epilog →
   {
    Red,
    Dashed,
    (*asintoto x-1=0*)
    Line[{{1, -8}, {1, 6}}],
    (*asintoto 2*)
    Line[{{-1, -8}, {-1, 6}}],
    Black,
    Point[{x0, f[x0]}]
   }
 ]
```



Per migliorare l'aspetto tronchiamo la tangente:

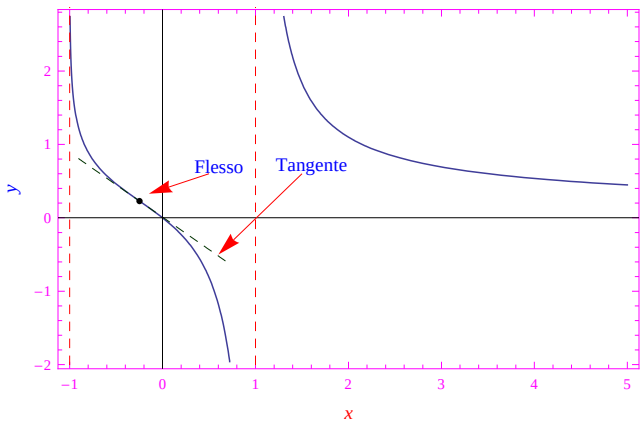
```
g1[x_] := Which[
  x < -0.9 || x > 0.7, Null,
  x > -0.9 && x < 0.7, g[x]
 ]
```

```
graficofunzione1 = Plot[
  {
   (*funzione*)
   f[x],
   (*tangente*)
   g1[x]
  },
  {x, -1, 5},
  Exclusions → x == 1,
  PlotStyle → {
    Directive[Thickness[0.00255]],
    Directive[Dashed, RGBColor[0, 0.2, 0], Thickness[0.00155]]
   },
  AxesLabel →
   {
    Style["x", Small, Red],
    Style["y", Small, Blue]
   },
  AxesStyle →
   {
    Directive[Red, Thickness[0.00255]],
    Directive[Blue, Thickness[0.00255]]
   },
  Epilog →
   {
    Red,
    Arrow[{{0.5, 0.6}, {x0 + 0.1, f[x0] + 0.1}}],
    Arrow[{{1.5, 0.6}, {0.6, -0.5}}],
    Text[Style["Flesso", Small, Blue], {0.6, 0.7}],
    Text[Style["Tangente", Small, Blue], {1.6, 0.7}],
    Dashed,
    Line[{{1, -2}, {1, 3}}],
    Line[{{-1, -2}, {-1, 3}}],
    Black,
    Point[{x0, f[x0]}]
   }
 ]
```



In alternativa, possiamo utilizzare **Frame**, resettando prima **SetOptions**

```
SetOptions[
  Plot,
  TicksStyle → Directive[Hue[5 / 6], 7],
  FrameStyle -> Directive[Hue[5 / 6], 7]
 ];

graficofunzione2 = Plot[
  {
   (*funzione*)
   f[x],
   (*tangente*)
   g1[x]
  },
  {x, -1, 5},
  Exclusions → x == 1,
  PlotStyle → {
    Directive[Thickness[0.00255]],
    Directive[Dashed, RGBColor[0, 0.2, 0], Thickness[0.00155]]
   },
  Frame → True,
  FrameLabel →
   {
    Style["x", Small, Red],
    Style["y", Small, Blue]
   },
  Epilog →
   {
    Red,
    Arrow[{{0.5, 0.6}, {x0 + 0.1, f[x0] + 0.1}}],
    Arrow[{{1.5, 0.6}, {0.6, -0.5}}],
    Text[Style["Flesso", Small, Blue], {0.6, 0.7}],
    Text[Style["Tangente", Small, Blue], {1.6, 0.7}],
    Dashed,
    Line[{{1, -2}, {1, 3}}],
    Line[{{-1, -2}, {-1, 3}}],
    Black,
    Point[{x0, f[x0]}]
   }
 ]
```




```
Export["graficofunzione2.eps", graficofunzione2];
```

Vediamo quest'altro esempio di studio di funzione:

```
Clear[f, graficofunzione, Df, D2f, x0]
```

```
f[x_] := ArcSin[(x^2 - 1)/(x^2 + 1)]
```

```
Df[x_] = f'[x] // Simplify
```

$$\frac{2\sqrt{\frac{x^2}{(1+x^2)^2}}}{x}$$

```
Solve[Df[x] == 0, x]
```

Solve::verif :
Potential solution {x → 0} (possibly discarded by verifier) should be checked by hand. May require use of limits. ≫

{}

```
Reduce[Df[x] == 0, x]
```

False

Calcolando a mano la derivata prima:

```
Clear[Df]
```

```
Df[x_] := Which[
  x > 0, 2/(x^2 + 1),
  x < 0, -2/(x^2 + 1)
 ]
```

da cui vediamo che non è definita in $x = 0$

```
Limit[
 Df[x],
 x → 0,
 Direction → +1
]
```

-2

```
Limit[
 Df[x],
 x → 0,
 Direction → -1
]
```

2

per cui $P_0\left(0, \frac{\pi}{2}\right)$ è un punto angoloso del grafico della funzione. L'equazione della tangente a sinistra in P_0 è

```
tan1[x_] := 2 * x - π/2
```

L'equazione della tangente a destra in P_0 è

```
tan2[x_] := 2 * x - π/2
```

La derivata seconda è

```
D2f[x_] = f''[x] // Simplify
```

$$-\frac{4\sqrt{\frac{x^2}{(1+x^2)^2}}}{1+x^2}$$

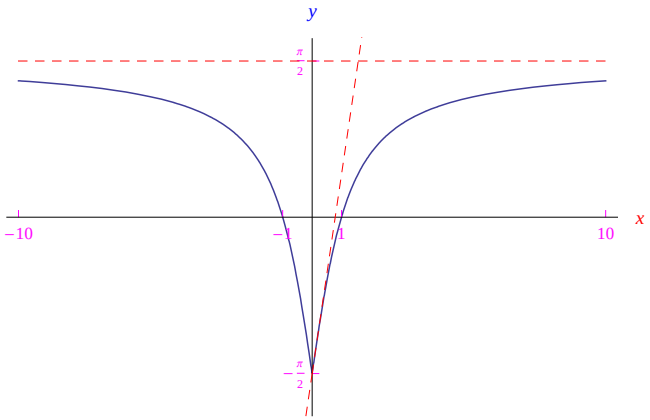
Anche qui conviene procedere manualmente:

```
Clear[D2f]

D2f[x_] := Which[
  x ≥ 0, -4 x/(x^2 + 1)^2,
  x < 0, 4 x/(x^2 + 1)^2
 ]
```

che è sempre < 0 per $x \neq 0$, per cui il grafico volge la concavità verso il basso.

```
SetOptions[
  Plot,
  TicksStyle -> Directive[Hue[5 / 6], 8],
  FrameStyle -> Directive[Hue[5 / 6], 8]
 ];
```

```
graficofunzione = Plot[
  f[x],
  {x, -10, 10},
  PlotRange → {-2, 1.8},
  PlotStyle → Directive[Thickness[0.00255]],
  AxesLabel →
   {
    Style["x", Small, Red],
    Style["y", Small, Blue]
   },
  Ticks →
   {
    {-1, 1, 10, -10},
    {
     π / 2, -π / 2
    }
   },

  Epilog →
   {
    Red,
    Dashed,
    Line[{{0, -π / 2}, {-2, tan1[-2]}}],
    Line[{{0, -π / 2}, {2, tan2[2]}}],
    Line[{{-10, π/2}, {10, π/2}}]
   }
 ]
```



Un altro esempio:

```
Clear[f, Df, graficofunzione]
```

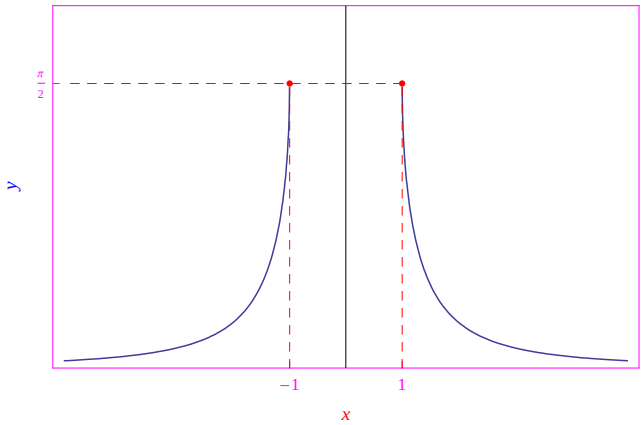
```
f[x_] := ArcSin[1/x^2]
```

```
Df[x_] = f'[x]
```

$$-\frac{2}{\sqrt{1-\frac{1}{x^4}}\ x^3}$$

Nei punti $P_1\left(-1, \frac{\pi}{2}\right)$ e $P_2\left(1, \frac{\pi}{2}\right)$ il grafico ha retta tangente verticale, poiché ivi la derivata diverge.

```
graficofunzione = Plot[
  f[x],
  {x, -5, 5},
  PlotRange → {0, 2},
  PlotStyle → Directive[Thickness[0.00255]],
  Frame → True,
  FrameLabel →
   {
    Style["x", Small, Red],
    Style["y", Small, Blue]
   },
  FrameTicks → {
    {
     {π / 2}, None},
    {
     {-1, 1}, None}},
  Epilog →
   {
    Red,
    Dashed,
    Line[{{-1, 0}, {-1, π / 2}}],
    Line[{{1, 0}, {1, π / 2}}],
    Line[{{1, π / 2}, {-5, π / 2}}],
    Point[{-1, π / 2}],
    Point[{1, π / 2}]
   }

 ]
```



■ L'istruzione GraphicsArray

L'istruzione **GraphicsArray** consente di visualizzare più oggetti grafici in un'unica cella di output. Ad esempio, supponiamo di voler visualizzare i grafici di $f(x) = \sin(x)$ e $g(x) = \sin(20\,x)$ in un'unica cella.

```
Clear[p1, p2]
```

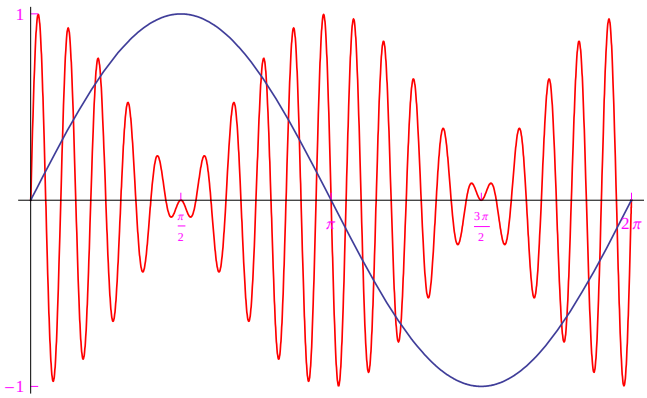
```
SetOptions[
  Plot,
  TicksStyle → Directive[Magenta, 7],
  AxesLabel →
   {
    Style["x", Small, Red],
    Style["y", Small, Blue]
   }
 ];
```

```
ticksx = {π / 2, π, 3/2 π, 2 π}; ticksy = {-1, 1};
```

```
(p1 = Plot[
    Sin[x],
    {x, 0, 2 π},
    PlotRange → {-1.2, 1.2},
    PlotStyle → Thickness[0.00255],
    Ticks → {
      ticksx,
      ticksy
     }
   ]; p2 = Plot[
    Sin[20 x] * Cos[x],
    {x, 0, 2 π},
    PlotStyle → {
      Red,
      Thickness[0.00255]
     }
   ]);
```

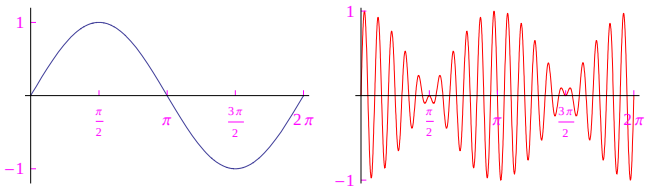
Se utilizziamo semplicemente il comando **Show**

```
Show[p2, p1]
```



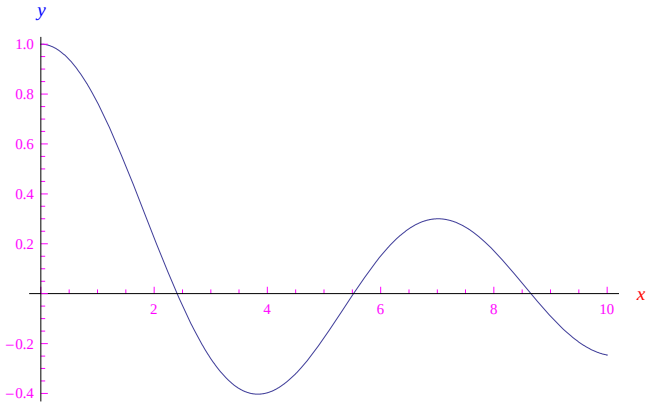
mentre noi li vogliamo separati. Definiamo

```
array = GraphicsArray[{p1, p2}]
```

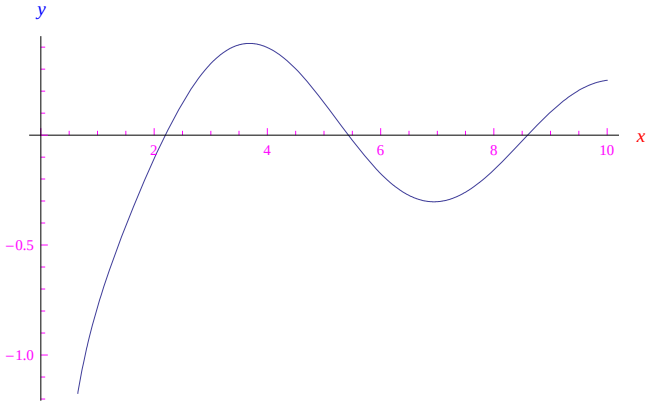


Un'utile opzione è **GraphicsArray** che permette di spaziare le subcelle. Ad esempio, definiamo i seguenti plots:

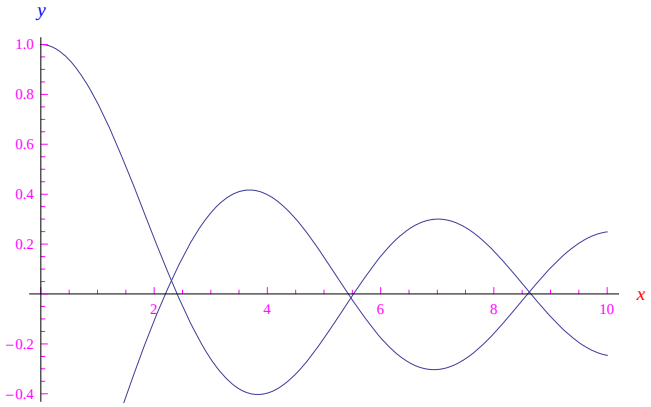
```
plot1 = Plot[
  BesselJ[0, x],
  {x, 0, 10}
 ]
```




```
plot2 = Plot[
  BesselY[1, x],
  {x, 0, 10}
 ]
```

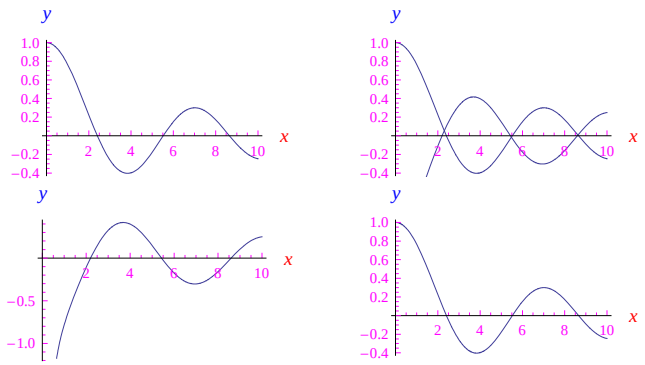



```
plot3 = Show[
  plot1, plot2
 ]
```



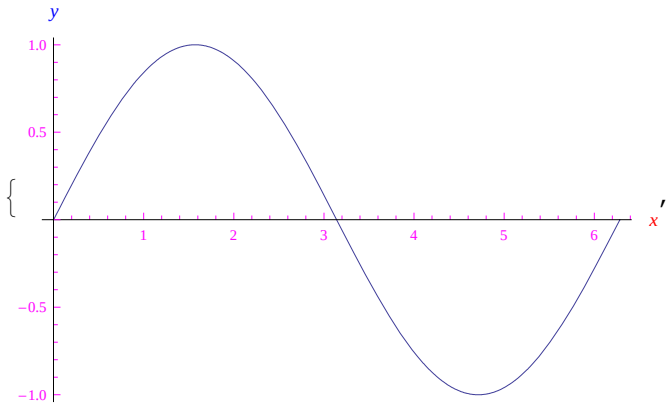
```
plotarray[h_, v_] := Show[
  GraphicsArray[
   {
    {plot1, plot3},
    {plot2, plot1}
   },
   GraphicsSpacing → {h, v}
  ]
 ]
```

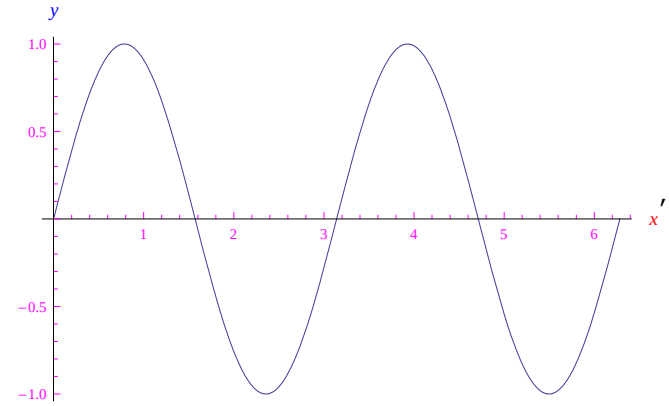
```
plotarray[0.3, 0]
```

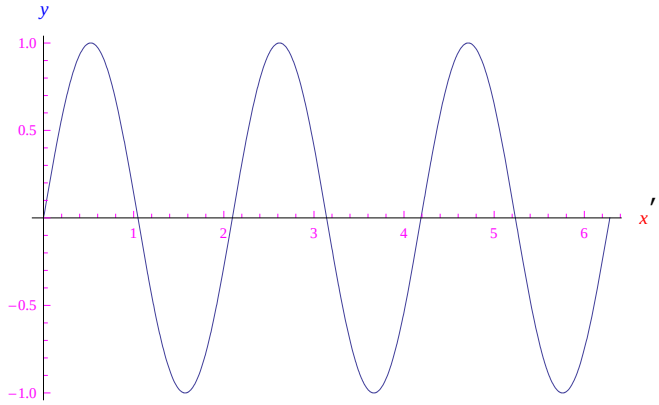


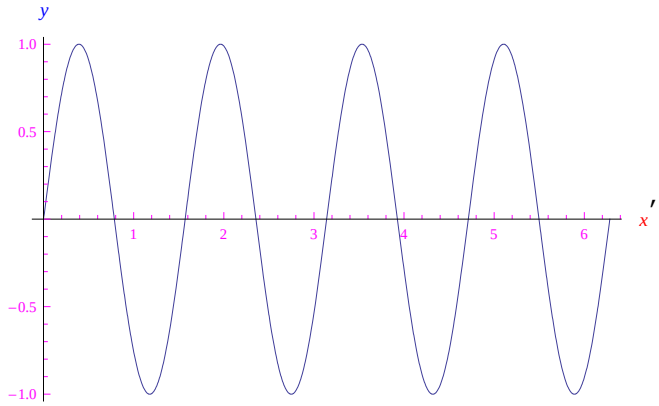

```
plots[n_] := Plot[Sin[n * x], {x, 0, 2 π}]
```

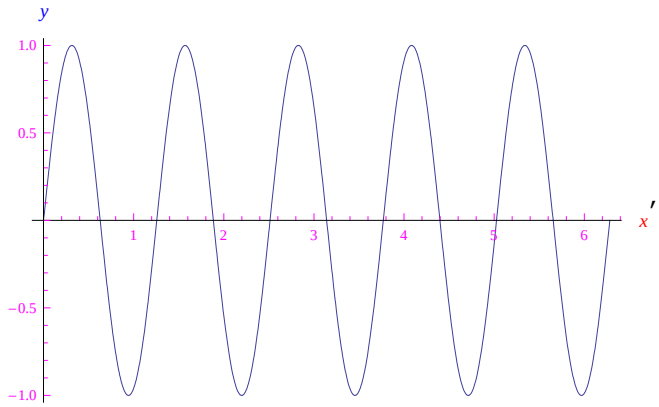
```
Table[
 plots[n],
 {n, 1, 10}
]
```

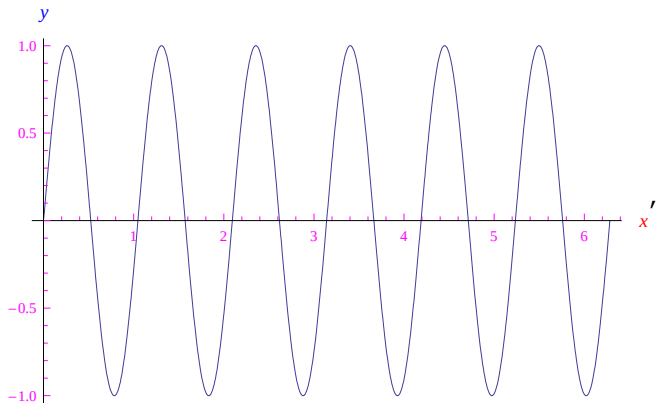


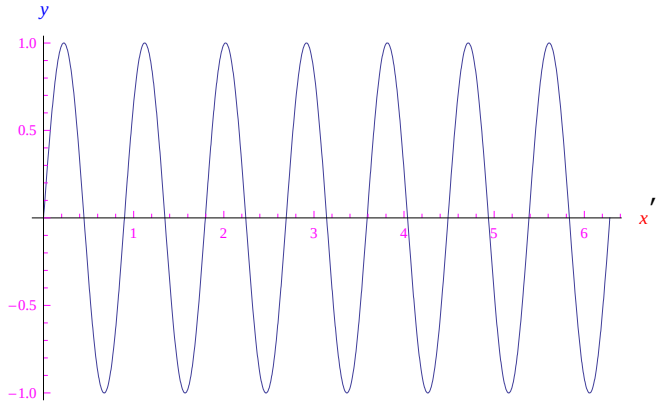



y
1.0
0.5
−0.5
−1.0
1
2
3
4
5
6
x'

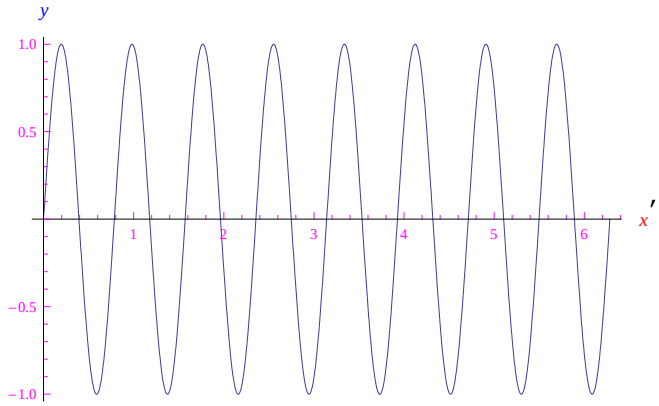

y
1.0
0.5
−0.5
−1.0
1
2
3
4
5
6
x'


y
1.0
0.5
−0.5
−1.0
1
2
3
4
5
6
x'

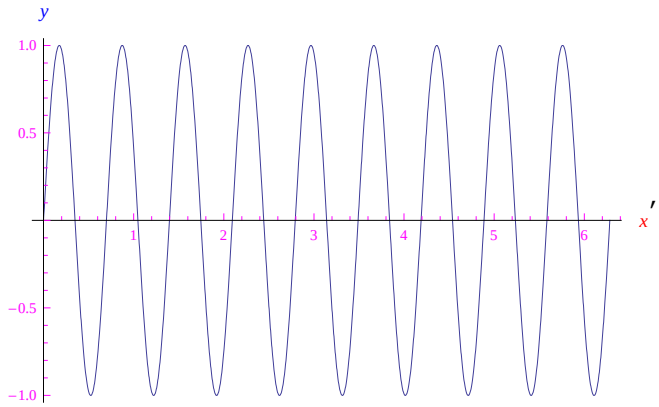

y
1.0
0.5
−0.5
−1.0
1
2
3
4
5
6
x'


y
1.0
0.5
−0.5
−1.0
x
1
2
3
4
5
6

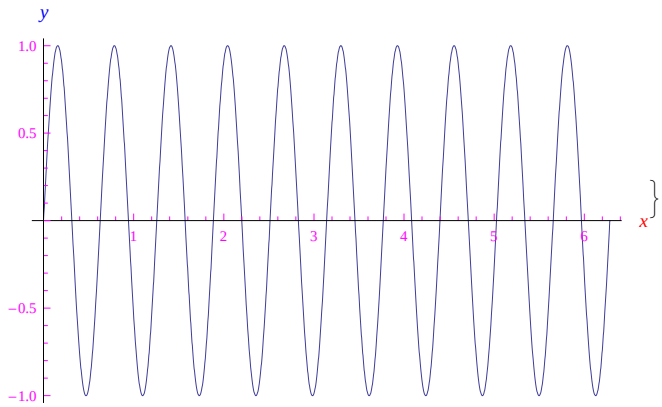
,


y
1.0
0.5
−0.5
−1.0
x
1
2
3
4
5
6

,

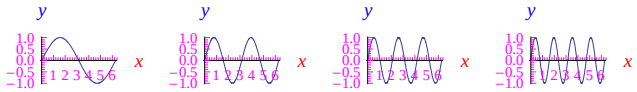

y
1.0
0.5
−0.5
−1.0
x
1
2
3
4
5
6

,


y
1.0
0.5
−0.5
−1.0
x
1
2
3
4
5
6

}

```
GraphicsArray[
 Table[
  plots[n],
  {n, 1, 4}
 ]
]
```



Applicazioni cinematiche

Vettore velocità

```
(*opzioni per i grafici*)

SetOptions[
  {
   ParametricPlot
  },
  TicksStyle -> Directive[
    Hue[5 / 6],
    10
   ]
 ];

(*definisco il vettore posizione
 del punto materiale*)

posizione[t_] := {x[t], y[t]}

(*quindi la velocità attraverso
 l'operatore di derivazione D*)

velocita[t_] = D[posizione[t], t];

(*assegno le equazioni orarie
 dei moti componenti*)

x[t_] := t^2; y[t_] := Cos[t]

(*verifichiamo*)

posizione[t]
```

$\{t^2, \text{Cos}[t]\}$

```
velocita[t]
```

{2 t, −Sin[t]}

```
(*il punto materiale parte a
 a velocità nulla*)

velocita[0]
```

{0, 0}

```
(*al termine*)
```

```
velocita[π]
```

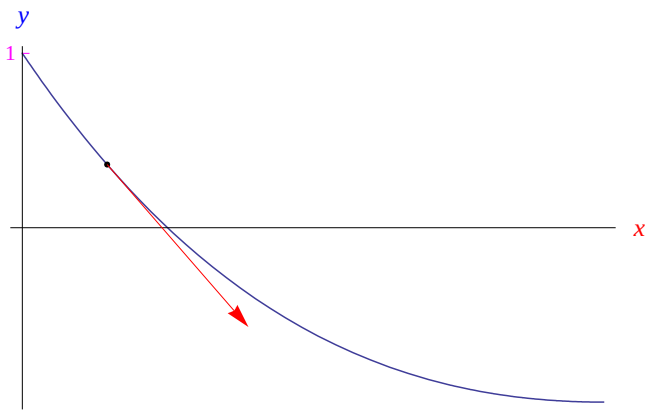
```
{2 π, 0}
```

```
(*definisco la traiettoria*)
```

```
traiettoria[τ_] := ParametricPlot[
  posizione[t],
  {t, 0, π},
  AspectRatio → 0.6,
  PlotRange → Automatic,
  PlotStyle → Thickness[0.00255],
  AxesLabel → {
    Style["x", Medium, Red],
    Style["y", Medium, Blue]
   },
  Ticks → {
    None,
    {
     1
    }
   },
  Epilog → {
    PointSize[0.01],
    Point[posizione[τ]],
    {
     Red,
     Arrow[{posizione[τ], posizione[τ] + velocita[τ]}]}
   }
 ]
```

```
(*a t=1.2*)
```

```
traiettoria[1.2]
```



■ Retta tangente

```
(*opzioni per i grafici*)
```

```
SetOptions[
  {
   Plot,
   ListPlot,
   ListLinePlot,
   ParametricPlot
  },
  TicksStyle -> Directive[
    Hue[5 / 6],
    10
   ]
 ];
```

```
f[x_] := Cosh[√x]
```

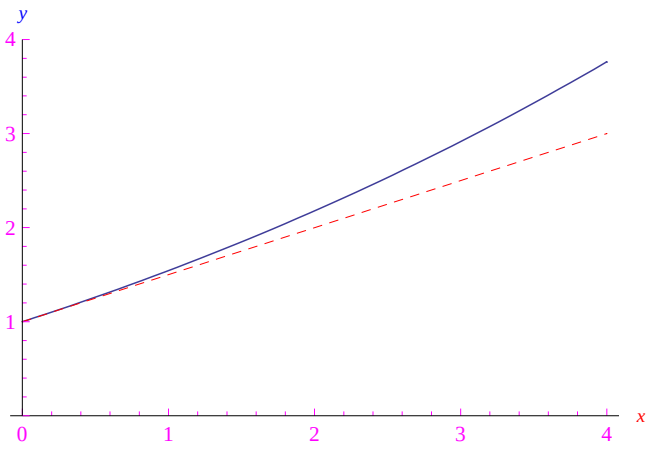
```
Df[x_] = f'[x]
```

$$\frac{\text{Sinh}\left[\sqrt{x}\right]}{2\sqrt{x}}$$

```
r[x_, x0_] := f[x0] + Df[x0] (x - x0)
```

$$r0[x\_] := \frac{x}{2} + 1$$

```
grafico = Plot[
  {f[x], r0[x]},
  {x, 0, 4},
  PlotRange → {0, 4},
  PlotStyle →
   {
    Thickness[0.00255],
    {
     Red,
     Dashed,
     Thickness[0.001555]
    }
   }
  ,
  PlotRange → Automatic,
  AxesLabel →
   {
    Style["x", Small, Red],
    Style["y", Small, Blue]
   },
  Ticks → Automatic,
  Epilog → {
    Red,
    Dashed,
    Line[
     {
      {0, -1}, {π^2, -1}
     }
    ]
   }

 ]
```




```
Clear[x, y, posizione, velocita, traiettoria]

posizione[t_] := {x[t], y[t]}

velocita[t_] = D[posizione[t], t];

x[t_] := t^2; y[t_] := Cos[t]
```

```
posizione[t]
```

$\{t^2, \text{Cos}[t]\}$

```
velocita[t]
```

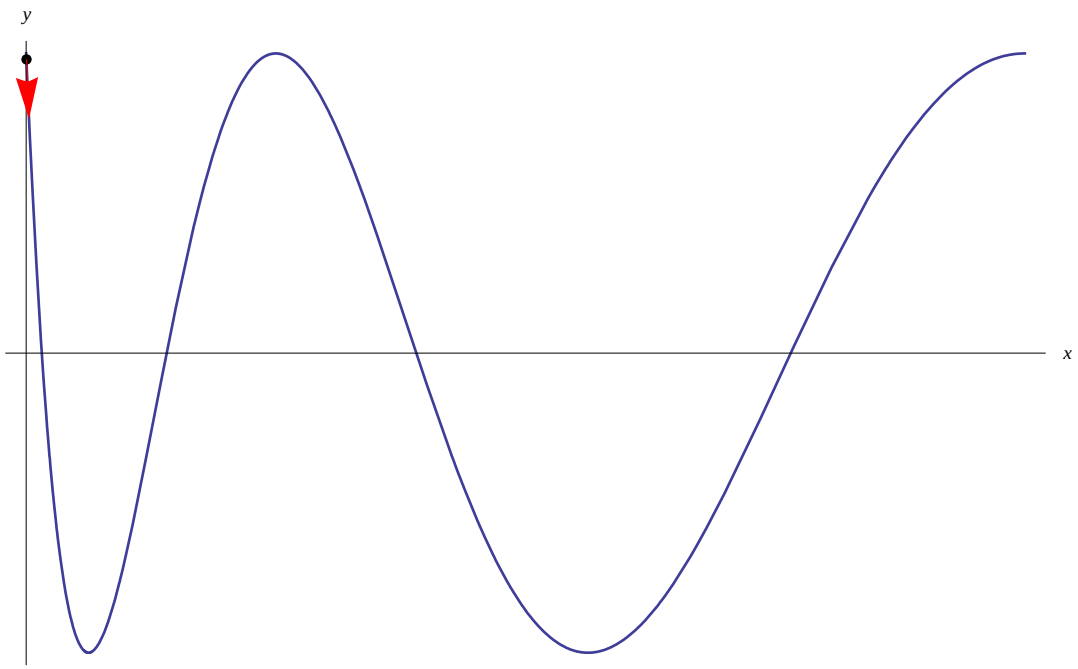
$\{2\,t, -\text{Sin}[t]\}$

```
velocita[0]
```

$\{0, 0\}$

```
traiettoria[τ_] := ParametricPlot[
  {posizione[t]}, {t, 0, 4 π},
  AspectRatio → 0.6,
  PlotRange → Automatic,
  PlotStyle → Thickness[0.00255],
  AxesLabel → {
    "x",
    "y"
   },
  ImageSize → {500, 500},
  Ticks → {
    None,
    None
   },
  Epilog → {
    PointSize[0.01],
    Point[posizione[τ]],

    {RGBColor[{1, 0, 0}], Arrow[{posizione[τ], posizione[τ] + velocita[τ]}]}
   }
 ]
```

```
traiettoria[0.2]
```



Cicloidi

```
SetOptions[
  {
   Plot,
   ParametricPlot
  },
  TicksStyle -> Directive[
    Hue[5 / 6],
    8
   ]
 ];
```

```
Solve[x^2 + y^2 - 2 y == 0, y]
```

$$\left\{\left\{y \to 1 - \sqrt{1 - x^2}\right\}, \left\{y \to 1 + \sqrt{1 - x^2}\right\}\right\}$$

```
Clear[eq]
```

```
eq[t_] := x^2 + y^2 - (2 * t) * x - 2 y + t^2
```

```
eq[2]
```

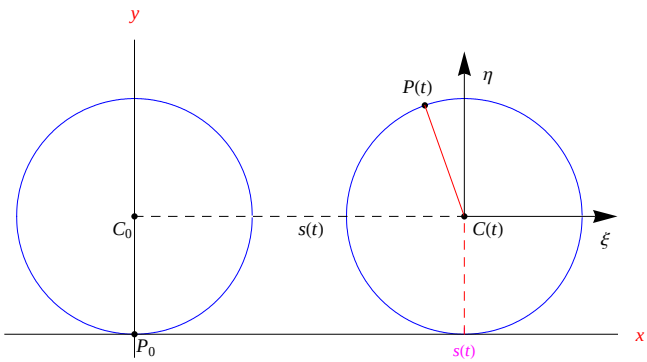
$4 - 4\,x + x^2 - 2\,y + y^2$

```
Solve[eq[2.8] == 0, y]
```

$\left\{\left\{y \to 0.5\left(2. - 2.\sqrt{3.8 - 1.\,x}\,\sqrt{-1.8 + x}\right)\right\}, \left\{y \to 0.5\left(2. + 2.\sqrt{3.8 - 1.\,x}\,\sqrt{-1.8 + x}\right)\right\}\right\}$

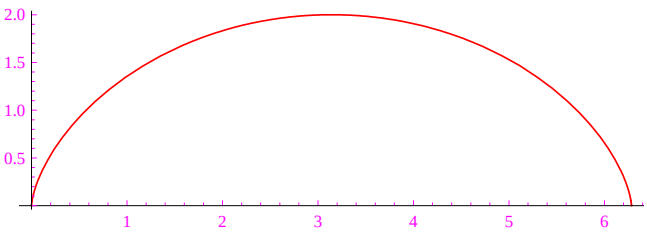
```
x[t_, R_, v0_] := v0 * t - R * Sin[(v0 * t)/R]; y[t_, R_, v0_] := R (1 - Cos[(v0 * t)/R])
xx[t_, R_, v0_] := {x[t, R, v0], y[t, R, v0]}
```

```
crf = Plot[
  {
   1 - Sqrt[1 - x^2], 1 + Sqrt[1 - x^2], 0.5` (2.` - 2.` Sqrt[3.8` - 1.` x] Sqrt[-1.7999999999999998` + x]),
   0.5` (2.` + 2.` Sqrt[3.8` - 1.` x] Sqrt[-1.7999999999999998` + x])
  },
  {x, -1, 4},
  AspectRatio → Automatic,
  PlotRange → {-0.2, 2.5},
  PlotStyle → {
    Blue
   },
  AxesLabel →
   {
    Style["x", Small, Red],
    Style["y", Small, Red]
   },
  Ticks →
   {
    {
     {2.8, "s(t)"}
    },
    None
   },
  Epilog →
   {
    Point[{0, 0}],
    Point[{0, 1}],
    Point[{2.8, 1}],
    Point[xx[2.8, 1, 1]],
    {
     Red,
     Dashed,
     Line[{{2.8, 0}, {2.8, 1}}]
    },
    {
     Red,
     Line[{{2.8, 1}, xx[2.8, 1, 1]}]
    },
    Arrow[{{2.8, 1}, {4.1, 1}}],
    Arrow[{{2.8, 1}, {2.8, 2.4}}],
    Text[Style["ξ", Small, Black], {4, .8}],
    Text[Style["η", Small, Black], {3, 2.2}],
    Text[Style["P(t)", Small, Black], {2.4, 2.1}],
    Text[Style["P0", Small, Black], {0.1, -0.1}],
    Text[Style["C(t)", Small, Black], {3, 0.9}],
    Text[Style["C0", Small, Black], {-0.1, 0.9}],
    {
     Dashed,
     Line[{{0, 1}, {2.8, 1}}],
     Text[Style["s(t)", Small, Black], {1.5, 0.9}]
    }
   }
 ]
```

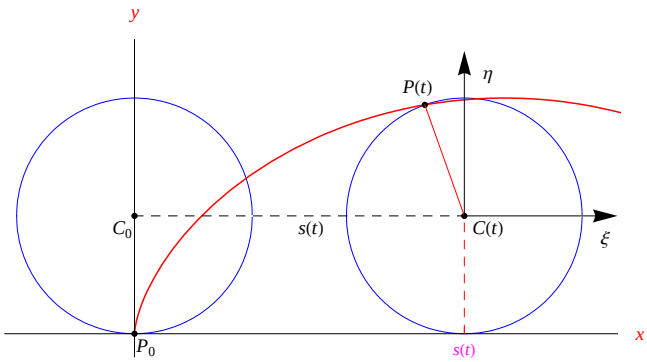



```
ciclo0[t, R_, v0_] := ParametricPlot[
  xx[t, R, v0],
  {t, 0, 2 π R / v0},
  PlotStyle → {
    Red,
    Thickness[0.00255]
   }
 ]
```

```
ciclo0[t, 1, 1]
```




```
Show[{crf, ciclo0[t, 1, 1]}]
```



La velocità del punto *P* si ottiene derivando rispetto al tempo la funzione vettoriale che abbiamo rappresentato in *Mathematica* con **xx[t, R, v0]**

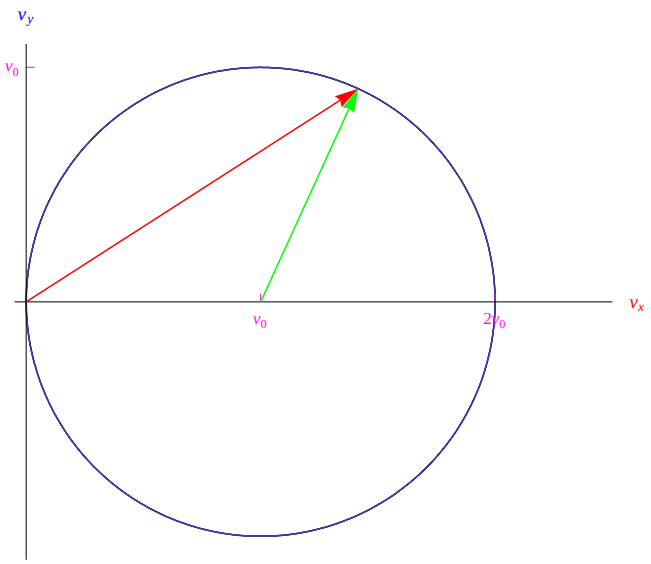
```
v[t_, R_, v0_] = D[xx[t, R, v0], t] // Simplify
```

$$\left\{v0 - v0\,\mathrm{Cos}\left[\frac{t\,v0}{R}\right],\ v0\,\mathrm{Sin}\left[\frac{t\,v0}{R}\right]\right\}$$

Grafichiamo il luogo geometrico dei punti descritto dal vettore velocità:

```
plotv[τ_, R_, v0_] := ParametricPlot[
  v[t, R, v0],
  {t, 0, 4 π},
  PlotRange → {{-0.1, 2.5 * v0}, {-2.2, 2.2}},
  PlotStyle -> Thickness[0.00255],
  AxesLabel → {
    Style["vx", Small, Red],
    Style["vy", Small, Blue]
   },
  Ticks →
   {
    {
     {v0, "v0"}, {2 v0, "2v0"}
    },
    {
     {v0, "v0"}
    }
   },
  Epilog → {
    Red,
    Thickness[0.00255],
    Arrow[{{0, 0}, v[τ, R, v0]}],
    Green,
    Arrow[{{v0, 0}, v[τ, R, v0]}]
   }
 ]
```

```
plotv[2, 2, 2]
```




```
Export["plotv.eps", plotv[2, 2, 2]];
```

```
plotv1[τ_, R_, v0_] := ParametricPlot[
  v[t, R, v0],
  {t, 0, 4 π},
  PlotRange → {{-0.1, 2.5 * v0}, {-2.2, 2.2}},
  PlotStyle -> Thickness[0.00255],
  AxesLabel → {
    Style["vx", Small, Red],
    Style["vy", Small, Blue]
   },
  Ticks →
   {
    {
     {v0, "v0"}, {2 v0, "2v0"}
    },
    {
     {v0, "v0"}
    }
   },
  Epilog → {
    Red,
    Thickness[0.00255],
    Arrow[{{0, 0}, v[τ, R, v0]}],
    Green,
    Arrow[{{v0, 0}, v[τ, R, v0]}]
   },
  ImageSize → {500, 500}
 ]
```

```
table = Table[
   plotv1[t, 2, 2],
   {t, 0, 2 π, 0.1}
  ];
```

```
Export["vcrf.gif", table]
```

vcrf.gif

Cicloidi ottenute con le matrici di rotazione

■ Cicloide

```
(*opzioni per i grafici*)
```

```
SetOptions[
  {
   ParametricPlot,
   Plot
  },
  TicksStyle → Directive[Hue[5 / 6], 9]
 ];
```

```
(*matrice di rotazione*)
```

```
B[t_] := {
  {Cos[t], Sin[t]},
  {-Sin[t], Cos[t]}
 }
```

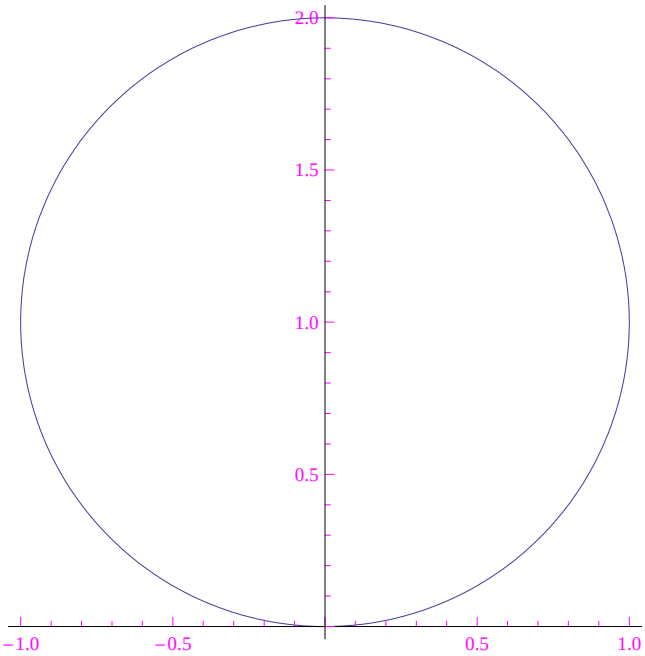
```
(*l'operatore di composizione . denota il prodotto righe per colonne*)
```

```
puntoruotato[centro_, t_, punto_] := centro + B[t].(punto - centro)
```

```
puntoruotato[{0, 1}, t, {0, 0}]
```

```
{-Sin[t], 1 - Cos[t]}
```

```
ParametricPlot[
 puntoruotato[{0, 1}, t, {0, 0}],
 {t, 0, 2 π}
]
```



Per generare la cicloide il punto ruotato deve essere traslato di $R\,\omega_0\,t$ nella direzione positiva dell'asse x. Per R=1 e $\omega_0 = 1\,\mathrm{rad/sec}$ si ha

```
traslazione[t_] := {t, 0}
```

```
cicloide[centro_, t_, punto_] := puntoruotato[centro, t, punto] + traslazione[t]
```

```
cicloide[{0, 1}, t, {0, 0}]
```

```
{t - Sin[t], 1 - Cos[t]}
```

ovvero una rappresentazione parametrica della cicloide.

```
velocita[t_] = ∂t cicloide[{0, 1}, t, {0, 0}]
```

```
{1 - Cos[t], Sin[t]}
```

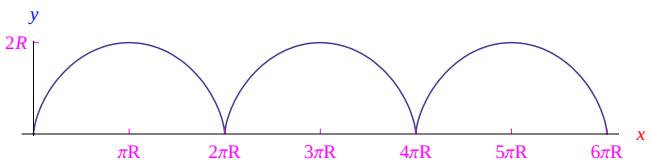
Ricerchiamo gli zeri del vettore velocità

```
Reduce[velocita[t] == 0, t]
```

```
C[1] ∈ Integers && t == 2 π C[1]
```

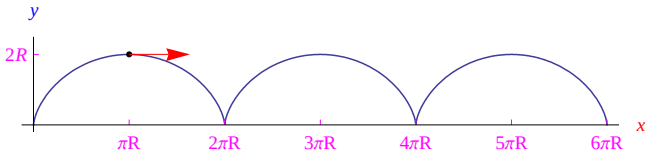
```
plot[centro_, punto_] := ParametricPlot[
  cicloide[centro, t, punto],
  {t, 0, 6 π},
  PlotStyle → Thickness[0.00255],
  AspectRatio → 1/(2 π),
  AxesLabel → {
    Style["x", Small, Red],
    Style["y", Small, Blue]
   },
  Ticks → {
    {
     {π, "πR"}, {2 π, "2πR"}, {3 π, "3πR"}, {4 π, "4πR"}, {5 π, "5πR"}
     , {6 π, "6πR"}},
    {
     {2, "2R"}
    }
   }
 ]
```

```
plot[{0, 1}, {0, 0}]
```




```
plotcicloide[τ_] := ParametricPlot[
  cicloide[{0, 1}, t, {0, 0}],
  {t, 0, 6 π},
  PlotStyle → Thickness[0.00255],
  AspectRatio → 1/(2 π),
  AxesLabel → {
    Style["x", Small, Red],
    Style["y", Small, Blue]
   },
  Ticks → {
    {
     {π, "πR"}, {2 π, "2πR"}, {3 π, "3πR"}, {4 π, "4πR"}, {5 π, "5πR"}
     , {6 π, "6πR"}},
    {
     {2, "2R"}
    }

   },
  PlotRange → {Automatic, {-0.2, 2.5}},
  Epilog → {
    PointSize[0.01],
    Point[cicloide[{0, 1}, τ, {0, 0}]],
    {
     Red,
     Arrow[{cicloide[{0, 1}, τ, {0, 0}], cicloide[{0, 1}, τ, {0, 0}] + velocita[τ]}]}
   }
 ]
```

```
plotcicloide[π]
```




```
Export["velmax.eps", %];

Clear[traslazione, velocita]
```

Rotolamento lungo una sinusoide smorzata da un esponenziale:

```
traslazione[t_] := {t, Sin[15 t] * Exp[-0.1 * t^2]}

velocita[t_] = ∂t cicloide[{0, 1}, t, {0, 0}]
```

$\{1 - \mathrm{Cos}[t],\ 15\, e^{-0.1 t^2}\, \mathrm{Cos}[15\, t] + \mathrm{Sin}[t] - 0.2\, e^{-0.1 t^2}\, t\, \mathrm{Sin}[15\, t]\}$

Plottiamo tale vettore nel piano $v_x\, v_y$. A tale scopo ridefiniamolo in funzione di un parametro a quale fattore di smorzamento e di una ampiezza b:

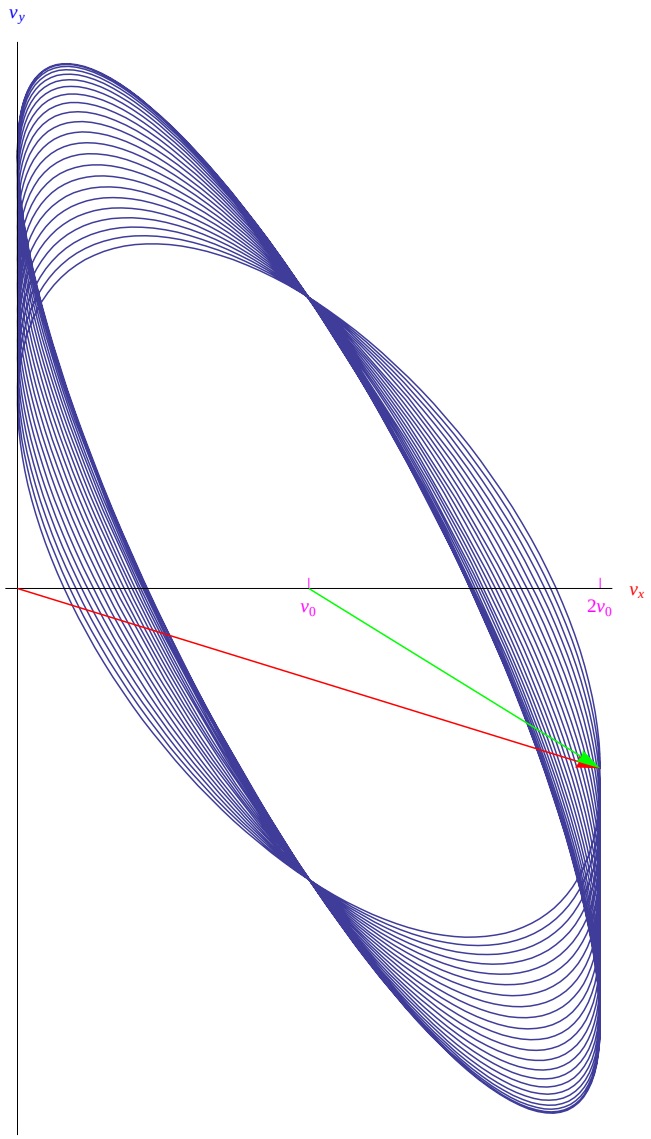
```
velocita1[t_, a_, b_] :=
 {1 - Cos[15 t], Sin[15 t] - 2 * a * b * Exp[-a * t^2] * Sin[15 t] + 15 * b * Exp[-a * t^2] * Cos[15 t]}

velocita1[t, 0, 0.2]
```

{1 - Cos[15 t], 3. Cos[15 t] + Sin[15 t]}

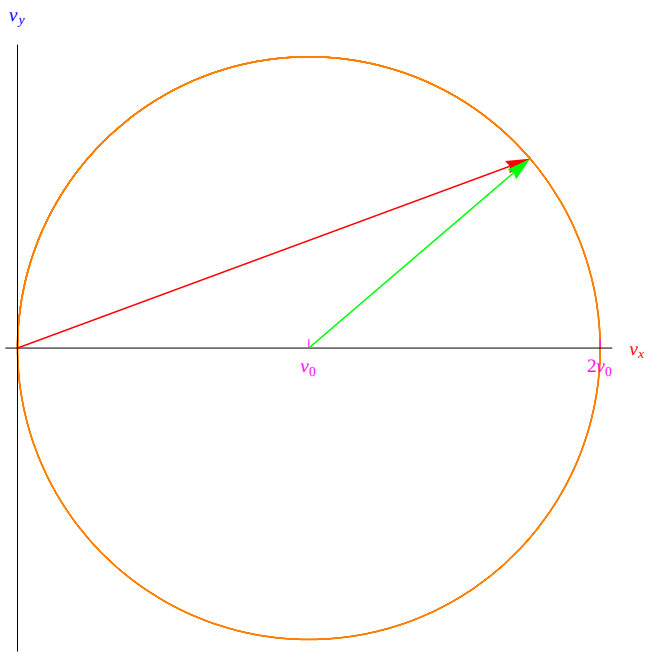
```
Clear[plotvel]

plotvel[τ_, a_, b_] := ParametricPlot[
  velocita1[t, a, b],
  {t, 0, τ},
  PlotRange → Automatic,
  PlotStyle -> Thickness[0.00255],
  AxesLabel → {
    Style["vx", Small, Red],
    Style["vy", Small, Blue]
   },
  Ticks →
   {
    {
     {1, "v0"}, {2, "2v0"}
    },
    {
     {2, "v0"}
    }
   },
  Epilog → {
    Red,
    Thickness[0.00255],
    Arrow[{{0, 0}, {1 - Cos[15 τ],
       Sin[15 τ] - 2 * a * b * Exp[-a * τ^2] * Sin[15 τ] + 15 * b * Exp[-a * τ^2] * Cos[15 τ]}}],
    Green,
    Arrow[{{1, 0}, velocita1[τ, a, b]}]
   }
 ]
```

Per un fattore nell'esponenzale $a = 10^{-2}$

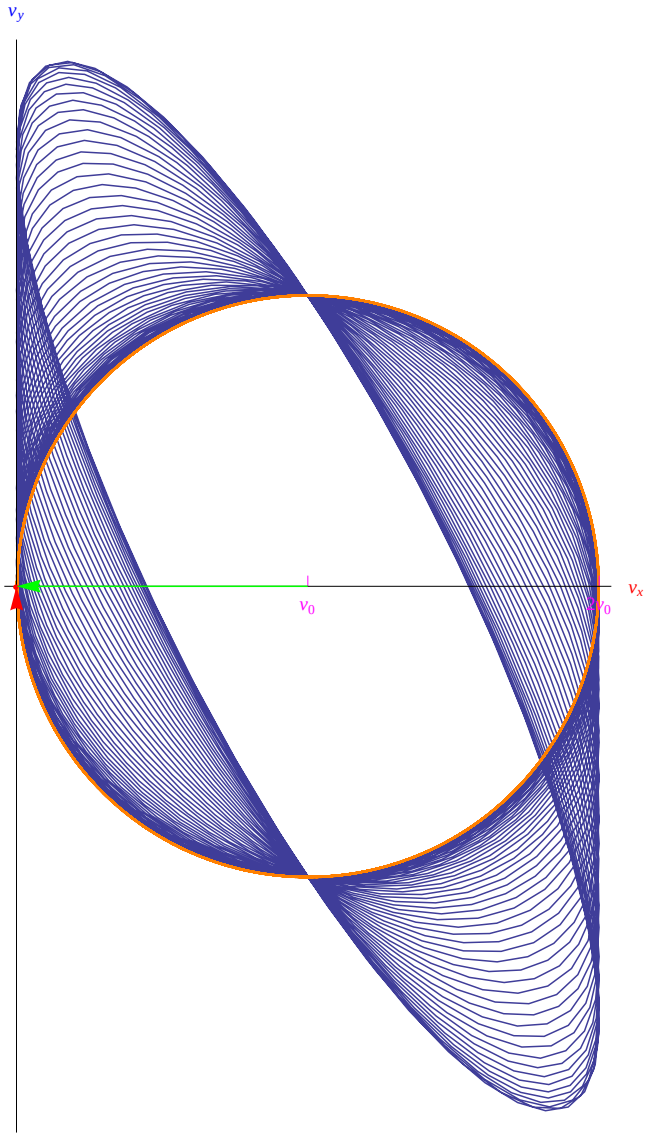
```
SetDirectory["G:\\Siti\\extrabyte2\\Mathematica\\GEOMETRIA_DIFF"];
plotvel[3 π, 10^-2, 0.1]
```



```
plotvel1[τ_, a_, b_] := ParametricPlot[
  velocita1[t, a, b],
  {t, 0, τ},
  PlotRange → Automatic,
  PlotStyle → Directive[Orange, Thickness[0.00255]],
  AxesLabel → {
    Style["vx", Small, Red],
    Style["vy", Small, Blue]
   },
  Ticks →
   {
    {
     {1, "v0"}, {2, "2v0"}
    },
    {
     {2, "v0"}
    }
   },
  Epilog → {
    Red,
    Thickness[0.00255],
    Arrow[{{0, 0}, {1 - Cos[15 τ],
       Sin[15 τ] - 2 * a * b * Exp[-a * τ^2] * Sin[15 τ] + 15 * b * Exp[-a * τ^2] * Cos[15 τ]}}],
    Green,
    Arrow[{{1, 0}, velocita1[τ, a, b]}]
   }
 ]

plotvel1[1, 0, 0]
```



```
Show[{
  plotvel[10.4 π, 10^-2, 0.1],
  plotvel1[10.4 π, 0, 0]
 }]
```




```
Clear[plotvel]
```

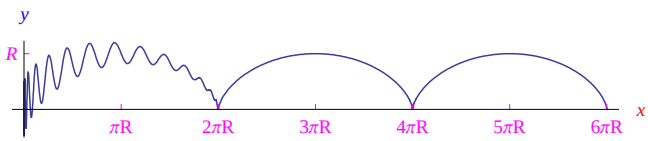
```
plotvel[τ_, a_, b_] := ParametricPlot[
  velocita1[t, a, b],
  {t, 0, τ},
  PlotRange → {{0, 2.1}, {-2, 2}},
  PlotStyle -> Thickness[0.00255],
  AxesLabel → {
    Style["vx", Small, Red],
    Style["vy", Small, Blue]
   },
  Ticks →
   {
    {
     {1, "v0"}, {2, "2v0"}
    },
    {
     {2, "v0"}
    }
   },
  Epilog → {
    Red,
    Thickness[0.00255],
    Arrow[{{0, 0}, {1 - Cos[15 τ],
       Sin[15 τ] - 2 * a * b * Exp[-a * τ^2] * Sin[15 τ] + 15 * b * Exp[-a * τ^2] * Cos[15 τ]}}]
   },
  ImageSize → {500, 500}
 ]
```

```
Table[plotvel[t, 10^-2, 0.1], {t, 0.1, 10 π}];
```

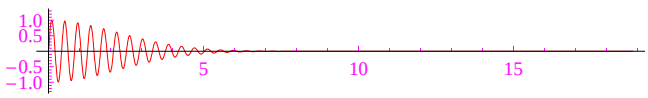
```
Export["complicato.gif", %]
```

```
complicato.gif
```

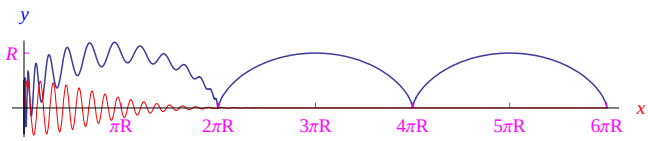
```
p1 = plot[{0, 1}, {0, 0}]
```




```
p2 = ParametricPlot[
  traslazione[t],
  {t, 0, 6 π},
  PlotStyle → {Red},
  PlotRange → All
 ]
```

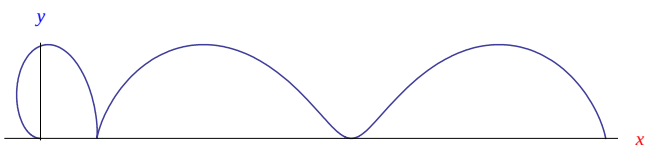



```
Show[{p1, p2}, PlotRange → All]
```



Esaminiamo il caso in cui il cerchio viene rotolato lungo una cicloide

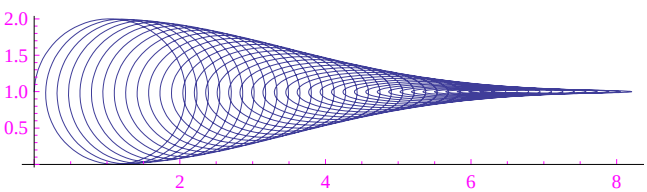
```
Clear[traslazione, velocita, plot, p1, p2, x, y]

traslazione[t_, R0_] := {t - R0 * Sin[t/R0], R0 * (1 - Cos[t/R0])}

cicloide3[centro_, t_, punto_, R0_] := puntoruotato[centro, t, punto] + traslazione[t, R0]

x[t_, v0_, R_, R0_, w0_] := w0 * t - R * Sin[v0 * t/R] - R0 * Sin[w0 * t/R0];

y[t_, v0_, R_, R0_, w0_] := R * (1 - Cos[v0 * t/R]) + R0 * (1 - Cos[w0 * t/R]);

cicloide3[t_, v0_, R_, R0_, w0_] := {x[t, v0, R, R0, w0], y[t, v0, R, R0, w0]}

plot[v0_, R_, R0_, w0_] := ParametricPlot[
  cicloide3[t, v0, R, R0, w0],
  {t, 0, 6 π},
  PlotStyle → Thickness[0.00255],
  AspectRatio → 1/(2 π),
  AxesLabel → {
    Style["x", Small, Red],
    Style["y", Small, Blue]
   },
  Ticks → None
 ]


SetDirectory["G:\\Siti\\extrabyte2\\Mathematica\\GEOMETRIA_DIFF"];

p1[R0_] := plot[1, 1, R0, 1]

p1[4]
```



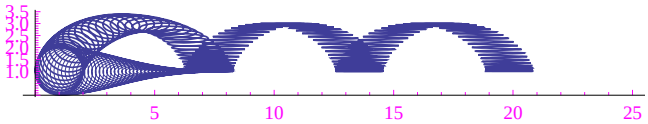

```
Export["w0.eps", %]
```

w0.eps

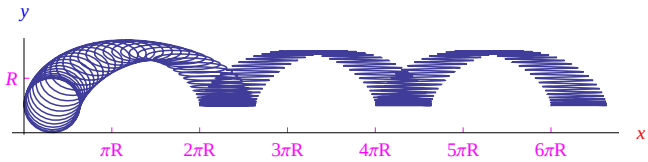
```
Clear[traslazione]

traslazione[t_] := {1 - Cos[40 t] + t, 1 + Sin[40 t] Exp[-0.1 t^2]}

p0 = ParametricPlot[traslazione[t], {t, 0, 2 π}]
```



```
Show[p0, p1, PlotRange → {{0, 25}, {0, 3.5}}]
```

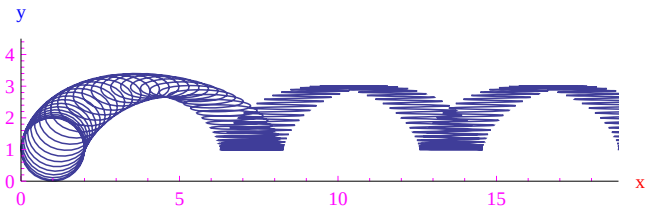



```
p1 = plot[{0, 1}, {0, 0}]
```




```
plotcicloide2[τ_] := ParametricPlot[
  cicloide[{0, 1}, t, {0, 0}],
  {t, 0, τ},
  PlotStyle → Thickness[0.00255],
  PlotRange → {{0, 6 π}, {0, 4.5}},
  AspectRatio → Automatic,
  AxesLabel → {
    Style["x", Small, Red],
    Style["y", Small, Blue]
   },
  Ticks → Automatic,
  PlotRange → {{0, 9}, {0, 9}},
  Axes → True
 ]
```

```
plotcicloide2[6 π]
```




```
SetDirectory["G:\\Siti\\extrabyte2\\Mathematica\\GEOMETRIA_DIFF"]
```

```
G:\Siti\extrabyte2\Mathematica\GEOMETRIA_DIFF
```

■ Generatore di cicloidi animate

Ora siamo in grado di programmare un *generatore di cicloidi animate*, seguendo il libro di S. Wagon (Guida a *Mathematica*), la cui routine utilizza un ciclo **Do**

```
? Do
```

Do[*expr*, {i_{max}}] evaluates *expr* i_{max} times.
Do[*expr*, {*i*, i_{max}}] evaluates *expr* with the variable *i* successively taking on the values 1 through i_{max} (in steps of 1).
Do[*expr*, {*i*, i_{min}, i_{max}}] starts with $i = i_{min}$.
Do[*expr*, {*i*, i_{min}, i_{max}, *di*}] uses steps *di*.
Do[*expr*, {*i*, {i_1, i_2, …}}] uses the successive values i_1, i_2, ….
Do[*expr*, {*i*, i_{min}, i_{max}}, {*j*, j_{min}, j_{max}}, …] evaluates *expr* looping over different values of *j*, etc. for each *i*. ≫

Il ciclo **Do** visualizza l'animazione grafica nel notebook di *Mathematica* e non può essere esportata. Modifichiamo, dunque, la routine di Wagon generando una gif animata. A tale scopo, riprendiamo le funzioni definite nella sezione "Cicloide"

```
B[t_] := {
  {Cos[t], Sin[t]},
  {-Sin[t], Cos[t]}
 }

puntoruotato[centro_, t_, punto_] := centro + B[t].(punto - centro)

traslazione[t_] := {t, 0}

cicloide[centro_, t_, punto_] := puntoruotato[centro, t, punto] + traslazione[t]
```

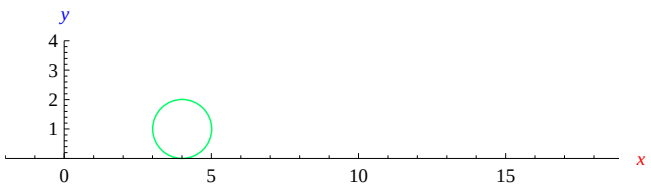
Definiamo una sorta di "cicloide unitaria":

```
cicloide1[t_] = cicloide[{0, 1}, t, {0, 0}]
```

```
{t - Sin[t], 1 - Cos[t] + Sin[t]}
```

La ruota rotola nella direzione positiva dell'asse x a partire dalla posizione di centro (0, 1) e raggio ovviamente unitario. In *Mathematica* è rappresentata dalla seguente funzione:

```
(*opzioni per i grafici*)

SetOptions[
  {
   Graphics,
   ParametricPlot,
   Plot
  },
  TicksStyle -> Directive[
    Hue[5 / 6],
    9
   ]
 ];

ruota[t_] := Graphics[
  {
   Hue[t/10],
   Thickness[0.00255],
   EdgeForm[Black],
   Circle[{t, 1}, 1]},
  PlotRange → {{-2, 6 π}, {0, 4}},
  Axes → True,
  AxesLabel →
   {
    Style["x", Small, Red],
    Style["y", Small, Blue]
   }
 ]

ruota[4]
```



Il punto del bordo della ruota che descrive la cicloide, è inizialmente in $P_0(0, 0)$ e al tempo t in $P(t)\,(t - \sin(t), 1 - \cos(t))$.

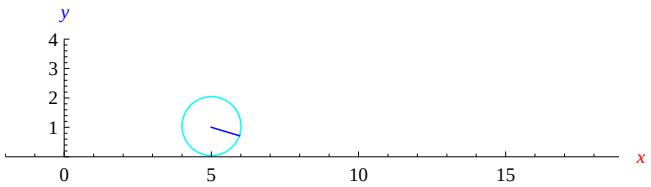
Quindi rappresentiamo il raggio "rotolante" da un segmento di estremi $(t, 1)$ e $P(t)$:

```
raggio[t_] := Graphics[
  {
   Blue,
   Thickness[0.0025],
   Line[{{t, 1}, cicloide1[t]}]
  }
 ]
```

Testiamo il codice:

```
test[t_] := Show[
  {
   ruota[t],
   raggio[t]
  }
 ]
```

```
test[5]
```

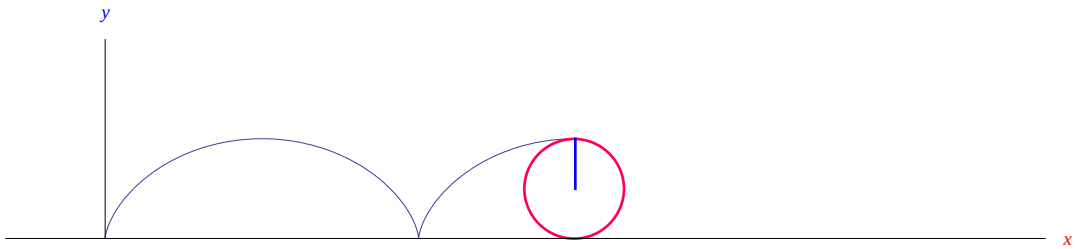



```
plotcicloide[t_] := ParametricPlot[
  cicloide1[τ],
  {τ, 0, t},
  PlotRange → {{-2, 6 π}, {0, 4}}
 ]
```

Dopo aver creato una lista di valori di **`cicloanimata[t]`** utilizzando **`Table`**, l'output può essere esportato come gif animata. È preferibile aumentare la dimensione dell'immagine con l'istruzione **`ImageSize`**.

```
cicloanimata[t_] := Show[
  {
   plotcicloide[t],
   ruota[t],
   raggio[t]
  },
  AxesLabel →
   {
    Style["x", Small, Red],
    Style["y", Small, Blue]
   },
  Ticks → None,
  ImageSize → {500, 500}
 ]
```

```
cicloanimata[3 π]
```



Generiamo la lista di fotogrammi, ricordando di inserire il terminatore **;** che blocca la visualizzazione dell'output.

```
tb = Table[
   cicloanimata[t],
   {t, 0.1, 6 π, 0.2}
  ];
```

Esportiamo l'output in formato gif.

```
Export["tb.gif", tb]
```

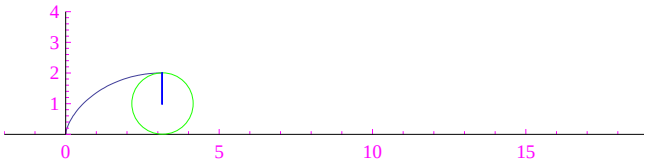
```
tb.gif
```

In alternativa, si può utilzzare **Epilog**

```
Clear[plotcicloide, cicloanimata]
```

```
cicloanimata[t_] := ParametricPlot[
  cicloide1[τ],
  {τ, 0, t},
  PlotRange → {{-2, 6 π}, {0, 4}},
  Epilog → {
    {
     Blue,
     Thickness[0.0025],
     Line[{{t, 1}, cicloide1[t]}]
    },
    (*ruota*)
    {
     Hue[t/10],
     EdgeForm[Black],
     Circle[{t, 1}, 1]
    },
    (*raggio*)
    {
     Blue,
     Thickness[0.0025],
     Line[{{t, 1}, cicloide1[t]}]
    }
   }
 ]
```

```
cicloanimata[π]
```



La routine può essere modificata per generare l'animazione grafica di una cicloide generalizzata. Ricordiamo che in tal caso dobbiamo modificare il vettore traslazione definito dalla funzione **traslazione[t]**.

```
Clear[traslazione, plotcicloide, ruota, cicloanimata, raggio, cicloide, cicloide1]
```

```
traslazione[t_] := {t, Sin[5 t]}
```

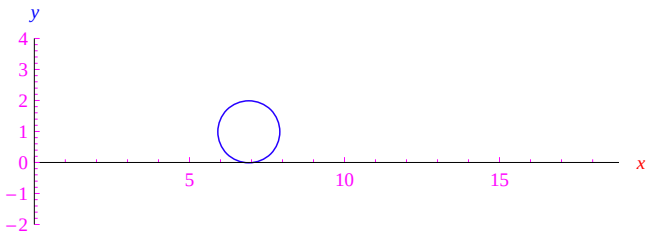
```
cicloide[centro_, t_, punto_] := puntoruotato[centro, t, punto] + traslazione[t]
```

```
cicloide1[t_] = cicloide[{0, 1}, t, {0, 0}]
```

```
{t - Sin[t], 1 - Cos[t] + Sin[5 t]}
```

```
ruota[t_] := Graphics[
  {
   Hue[t/10],
   Thickness[0.00255],
   EdgeForm[Black],
   Circle[{t, 1 + Sin[5 t]}, 1]},
  PlotRange → {{0, 6 π}, {-2, 4}},
  Axes → True,
  AxesLabel →
   {
    Style["x", Small, Red],
    Style["y", Small, Blue]
   }
 ]
```

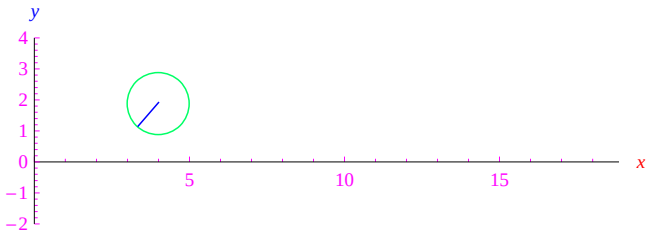
```
ruota[2.2 π]
```




```
raggio[t_] := Graphics[
  {
   Blue,
   Thickness[0.0025],
   Line[{{t, 1 + Sin[5 t]}, {t + Cos[t], 1 + Sin[5 t] + Sin[t]}}]
  }
 ]
```

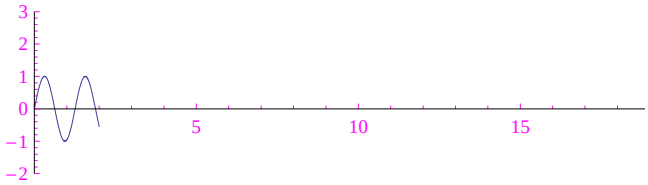
```
test1[t_] := Show[
  {
   ruota[t],
   raggio[t]
  }
 ]
```

```
test1[4]
```



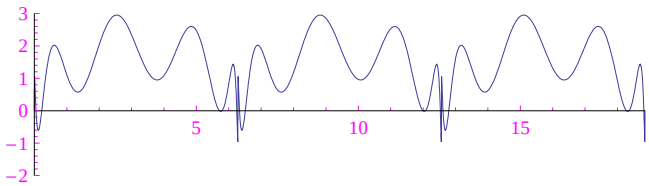

```
plottraslazione[t_] := ParametricPlot[
  traslazione[τ],
  {τ, 0, t},
  PlotRange → {{0, 6 π}, {-2, 3}}
 ]
```

```
plottraslazione[2]
```




```
plotcicloide[t_] := ParametricPlot[
  cicloide1[τ],
  {τ, 0, t},
  PlotRange → {{0, 6 π}, {-2, 3}}
 ]
```

```
plotcicloide[6 π]
```




```
cicloanimata[t_] := Show[
  {

   plotcicloide[t],
   ruota[t]
  },
  AxesLabel →
   {
    Style["x", Small, Red],
    Style["y", Small, Blue]
   },
  Ticks → None
 ]
```

```
cicloanimata[π]
```

■ Trocoide

```
Clear[traslazione, puntoruotato, B]
```

```
(*opzioni per i grafici*)
```

```
SetOptions[
  {
   ParametricPlot,
   Plot
  },
  TicksStyle → Directive[Hue[5 / 6], 9],
  FrameStyle → Directive[Hue[5 / 6], 9]
 ];
```

```
(*matrice di rotazione*)
```

$$B[t\_, R\_] := \left\{ \left\{\mathrm{Cos}\left[\frac{t}{R}\right], \mathrm{Sin}\left[\frac{t}{R}\right]\right\}, \left\{-\mathrm{Sin}\left[\frac{t}{R}\right], \mathrm{Cos}\left[\frac{t}{R}\right]\right\} \right\}$$

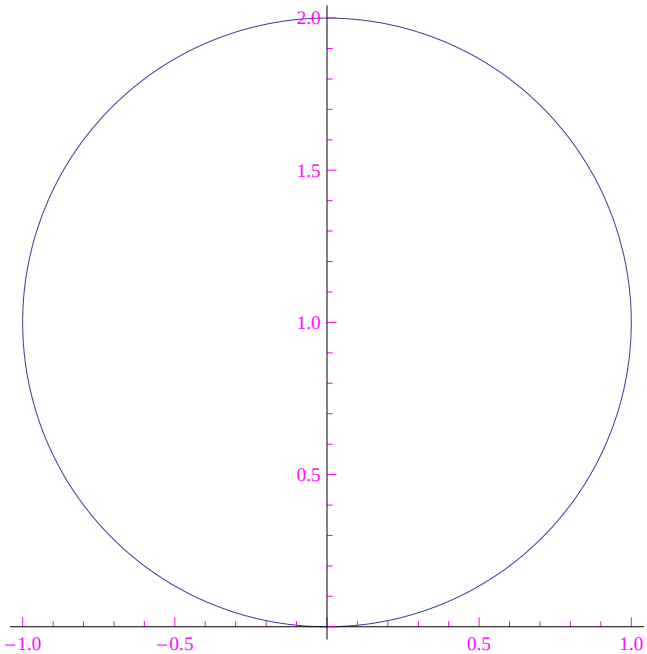
Se il raggio della ruota è R, il raggio della circonferenza a cui appartiene il punto mobile è $R_0 > R$

```
puntoruotato[t_, R0_, R_] := {0, R} + B[t, R].({0, R - R0} - {0, R})
```

```
puntoruotato[t, 2, 1]
```

```
{-2 Sin[t], 1 - 2 Cos[t]}
```

```
ParametricPlot[
 puntoruotato[t, 1, 1],
 {t, 0, 2 π}
]
```



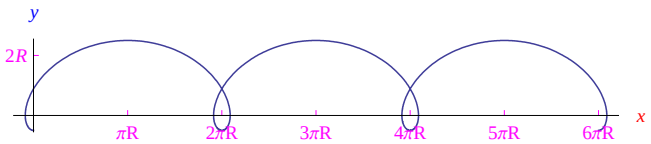
$$\mathrm{traslazione}[t\_, R\_] := \left\{\frac{t}{R}, 0\right\}$$

```
trocoide[t_, R0_, R_] = puntoruotato[t, R0, R] + traslazione[t, R]
```

$$\left\{\frac{t}{R} - R0\,\mathrm{Sin}\left[\frac{t}{R}\right], R - R0\,\mathrm{Cos}\left[\frac{t}{R}\right]\right\}$$

```
SetDirectory["G:\\Siti\\extrabyte2\\Mathematica\\GEOMETRIA_DIFF"];
```

```
ParametricPlot[
 trocoide[t, 1.5, 1],
 {t, 0, 6 π},
 PlotStyle → Thickness[0.00255],
 AxesLabel → {
   Style["x", Small, Red],
   Style["y", Small, Blue]
  },
 Ticks → {
   {
    {π, "πR"}, {2 π, "2πR"}, {3 π, "3πR"}, {4 π, "4πR"}, {5 π, "5πR"}
    , {6 π, "6πR"}},
   {
    {2, "2R"}
   }
  }
]
```




```
Export["trocoide.eps", %]
```

trocoide.eps

```
Clear[velocita]
```

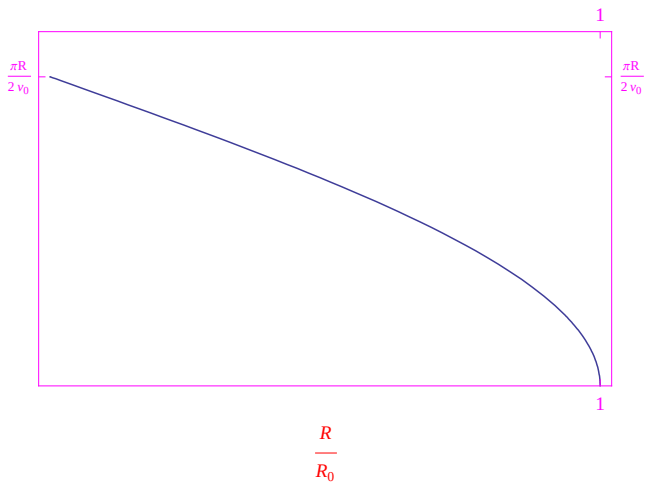
```
velocita[t_, R0_, R_] = D[trocoide[t, R0, R], t] // Simplify
```

$$\left\{\frac{1 - R0\,\mathrm{Cos}\left[\frac{t}{R}\right]}{R}, \frac{R0\,\mathrm{Sin}\left[\frac{t}{R}\right]}{R}\right\}$$

```
Reduce[velocita[t, 1.5, 1] == {0, 0}]
```

Sin[t] == 0. && Cos[t] == 0.666667

```
arccos = Plot[
  ArcCos[x],
  {x, 0, 1},
  PlotRange → {0, 1.8},
  PlotStyle → {
    Directive[Thickness[0.00255]]
   },
  Frame → True,
  FrameLabel →
   {
    Style["R/R0", Small, Red]
   },
  FrameTicks →
   {
    {
     1
    },
    {
     {π/2, "πR/(2 v0)"}
    }
   }
 ]
```




```
Export["tstar.eps", arccos];
```

Istante in cui si annulla la componente v_x del vettore velocità (poniamo $v_0 = 1$ nelle appropriate unità di misura):

```
t1[R0_, R_] := R * ArcCos[R/R0]
```

Posizione della particella all'istante t_1

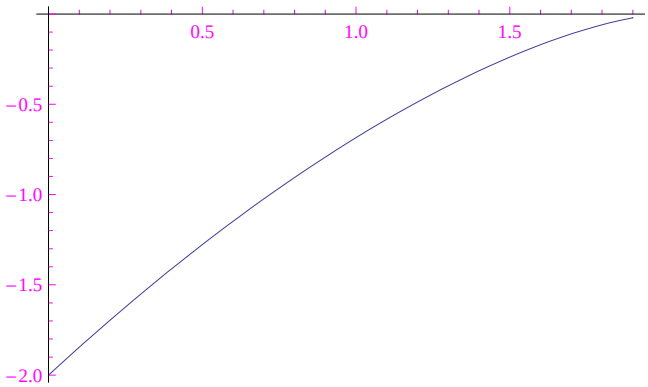
```
vxnull[R0_, R_] = trocoide[t1[R0, R], R0, R]
```

$$\left\{-\sqrt{1-\frac{R^2}{R0^2}}\ R0 + \text{ArcCos}\left[\frac{R}{R0}\right],\ 0\right\}$$

L'ascissa della particella al tempo t_1 è minore di 0 per ogni valore di $R \in (0, R_0)$

```
x[R0_, R_] := R * ArcCos[R/R0] - Sqrt[R0^2 - R^2]
```

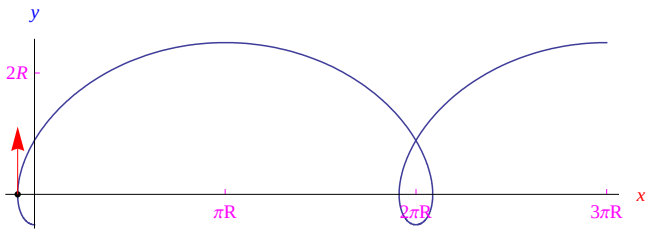
```
plotx = Plot[
  x[2, R],
  {R, 0, 1.9}
 ]
```




```
vxnull1[R0_, R_] := {x[R0, R], 0}
```

```
x1[R0_, R_] := x[R0, R] + 2 π
```

```
ParametricPlot[
 trocoide[t, 1.5, 1],
 {t, 0, 3 π},
 PlotStyle → Thickness[0.00255],
 AxesOrigin → {0, 0},
 AxesLabel → {
   Style["x", Small, Red],
   Style["y", Small, Blue]
  },
 Ticks → {
   {
    {π, "πR"}, {2 π, "2πR"}, {3 π, "3πR"}, {4 π, "4πR"}, {5 π, "5πR"}
    , {6 π, "6πR"}},
   {
    {2, "2R"}
   }
  },
 Epilog → {
   Point[vxnull1[1.5, 1]],
   {
    Red,
    Arrow[{trocoide[t1[1.5, 1], 1.5, 1],
      trocoide[t1[1.5, 1], 1.5, 1] + velocita[t1[1.5, 1], 1.5, 1]}]
   }
  }

]
```

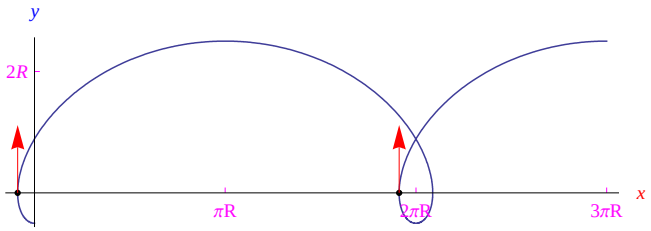



```
Export["vnull.eps", %]
```

```
vnull.eps
```

Per ottenere il punto successivo dobbiamo eseguire una traslazione di 2πR poiché tale è il periodo della trocoide:

```
ParametricPlot[
 trocoide[t, 1.5, 1],
 {t, 0, 3 π},
 PlotStyle → Thickness[0.00255],
 AxesOrigin → {0, 0},
 AxesLabel → {
   Style["x", Small, Red],
   Style["y", Small, Blue]
  },
 Ticks → {
   {
    {π, "πR"}, {2 π, "2πR"}, {3 π, "3πR"}, {4 π, "4πR"}, {5 π, "5πR"}
    , {6 π, "6πR"}},
   {
    {2, "2R"}
   }
  },
 Epilog → {
   Point[vxnull1[1.5, 1]],
   Point[{x1[1.5, 1], 0}],
   {
    Red,
    Arrow[{trocoide[t1[1.5, 1], 1.5, 1],
      trocoide[t1[1.5, 1], 1.5, 1] + velocita[t1[1.5, 1], 1.5, 1]}],
    Arrow[{trocoide[t1[1.5, 1] + 2 π, 1.5, 1],
      trocoide[t1[1.5, 1] + 2 π, 1.5, 1] + velocita[t1[1.5, 1], 1.5, 1]}]
   }
  }

]
```



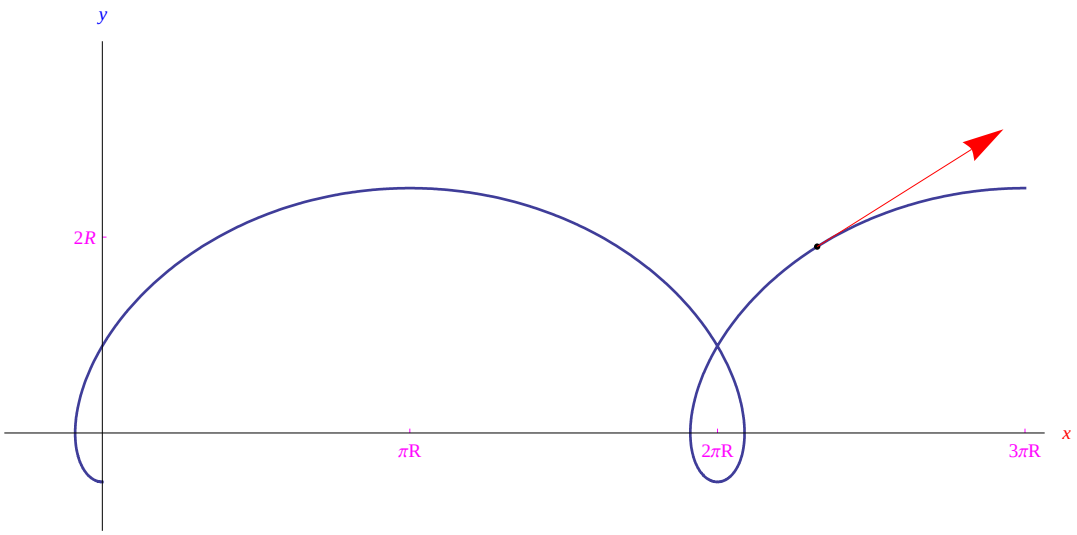

```
Export["vnull1.eps", %]
```

```
vnull1.eps
```

A questo punto è possibile realizzare un'animazione grafica in modo da visualizzare l'andamento del vettore velocità e relativi punti di inversione del moto.

```
trocoide1[τ_] := ParametricPlot[
  trocoide[t, 1.5, 1],
  {t, 0, 3 π},
  PlotStyle → Thickness[0.00255],
  PlotRange → {{-1, 3 π + 0.2}, {-1, 4}},
  AxesLabel → {
    Style["x", Small, Red],
    Style["y", Small, Blue]
   },
  Ticks → {
    {
     {π, "πR"}, {2 π, "2πR"}, {3 π, "3πR"}, {4 π, "4πR"}, {5 π, "5πR"}
     , {6 π, "6πR"}},
    {
     {2, "2R"}
    }
   },
  Epilog → {
    Point[trocoide[τ, 1.5, 1]],
    {
     Red,
     Arrow[{trocoide[τ, 1.5, 1], trocoide[τ, 1.5, 1] + velocita[τ, 1.5, 1]}]
    }
   },
  ImageSize → {500, 500}
 ]
```

```
trocoide1[8.5]
```




```
table = Table[trocoide1[τ], {τ, 0, 8.5, 0.2}];
Export["trocoide.gif", table]
```

trocoide.gif

Epicicloide

```
SetOptions[
  ParametricPlot,
  TicksStyle -> Directive[Hue[5 / 6], 8]];
```

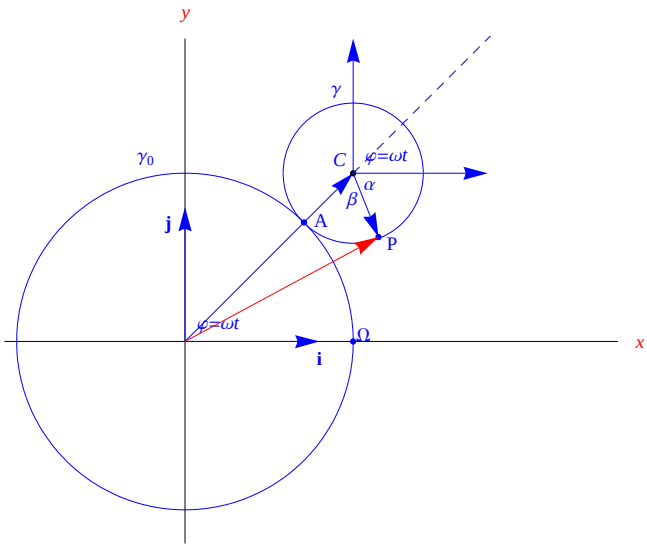
L'epicicloide è la traiettoria percorsa da un punto del bordo di una monetina che rotola (senza strisciare) sul bordo di una moneta.

```
moneta = Plot[
  {
```

$$\sqrt{1-x^2},\ -\sqrt{1-x^2},\ 1+\sqrt{\frac{1}{1.55^4}-(x-1)^2},\ 1-\sqrt{\frac{1}{1.55^4}-(x-1)^2}$$

```
},
{x, -1, 2.5},
AspectRatio → Automatic,
PlotRange → {-1.2, 1.8},
PlotStyle → {
  Blue
 },
AxesLabel →
 {
  Style["x", Small, Red],
  Style["y", Small, Red]
 },
Ticks → None,
Epilog →
 {
  {
   Blue,
   Arrow[{{0, 0}, {1, 1}}]
  },
  {
   Point[{1, 1}],
   Text[Style["C", Small, Blue], {0.92, 1.08}]
  },
  {
   Blue,
   Dashed,
   Line[{{1, 1}, {2, 2}}]
  },
  {
   Blue,
   Arrow[{{0, 0}, {0.8, 0}}],
   Arrow[{{0, 0}, {0, 0.8}}],
   Text[Style["φ=ωt", Small, Blue], {0.2, 0.1}],
   Arrow[{{1, 1}, {1 + 0.8, 1}}],
   Arrow[{{1, 1}, {1, 1 + 0.8}}],
   Text[Style["φ=ωt", Small, Blue], {1.2, 1.1}],
   Arrow[{{1, 1}, {1.15, 0.62}}],
   Point[{1.15, 0.62}],
   Text[Style["P", Small, Blue], {1.23, 0.58}],
   Text[Style["γ0", Small, Blue], {-.23, 1.1}],
   Text[Style["γ", Small, Blue], {0.9, 1.5}],
   Text[Style["i", Small, Bold], {0.8, -0.1}],
   Text[Style["j", Small, Bold], {-0.1, 0.7}],
   Text[Style["α", Small, Blue], {1.1, 0.94}],
   Text[Style["β", Small, Blue], {0.99, 0.84}],
   Point[{1, 0}],
   Text[Style["Ω", Small, Blue], {1.06, 0.04}],
   Point[{1/Sqrt[2], 1/Sqrt[2]}],
   Text[Style["A", Small, Blue], {1/Sqrt[2] + 0.1, 1/Sqrt[2] + 0.01}]
  },
  {
   Red,
```

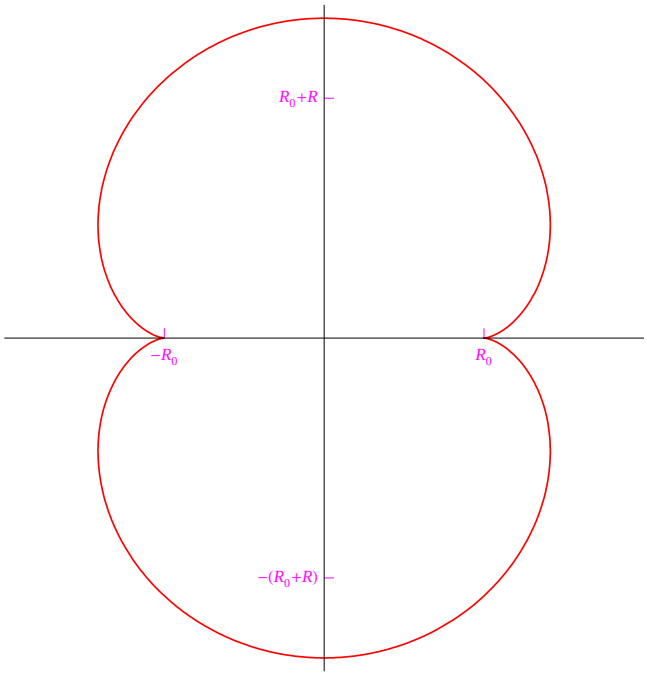
```
      Arrow[{{0, 0}, {1.15, 0.62}}]
     }
    }

  ]
```



Equazioni parametriche dell'epicicloide:

```
x[t_, R_, R0_, w0_] := (R0 + R) * Cos[w0 * t] - R * Cos[(R0 + R)/R * w0 * t];

y[t_, R_, R0_, w0_] := (R0 + R) * Sin[w0 * t] - R * Sin[(R0 + R)/R * w0 * t];

xx[t_, R_, R0_, w0_] := {x[t, R, R0, w0], y[t, R, R0, w0]}

epi[R_, R0_] := ParametricPlot[
  xx[t, R, R0, 1],
  {t, 0, 2 π},
  PlotRange → {{-4, 4}, Automatic},
  PlotStyle → {
    Red,
    Thickness[0.00255]
   },
  Ticks →
   {
    {
     {R0, "R0"}, {-R0, "-R0"}
    },
    {
     {R0 + R, "R0+R"}, {-(R0 + R), "-(R0+R)"}
    }
   }
 ]
```

```
epi[1, 2]
```

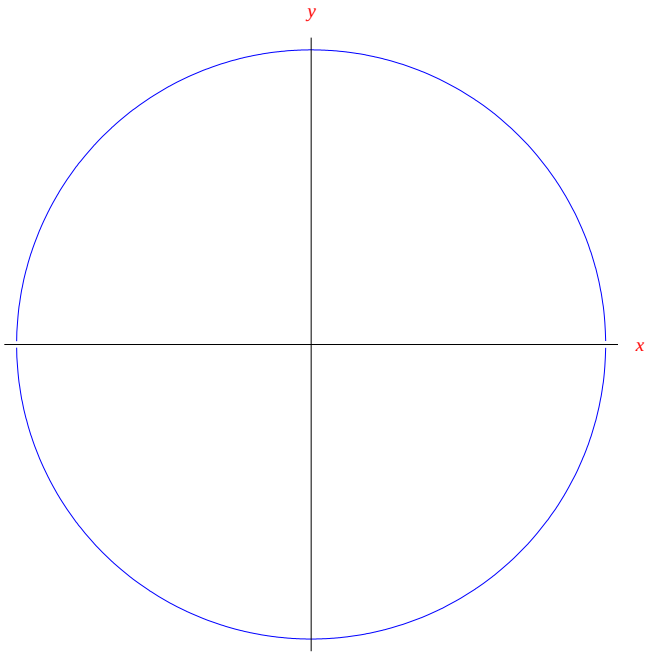


Velocità vettoriale

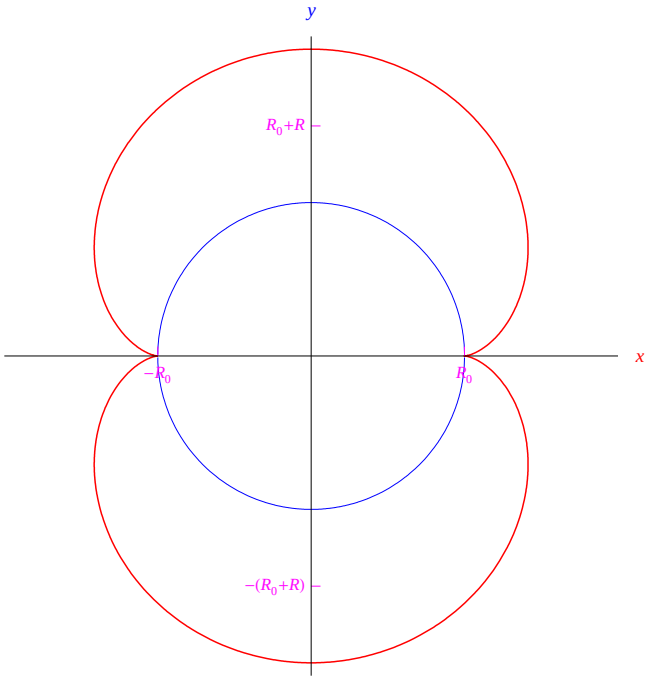
```
v[t_, R_, R0_, w0_] = ∂t xx[t, R, R0, w0] // Simplify
```

$$\left\{(R+R0)\,w0\left(-\mathrm{Sin}[t\,w0]+\mathrm{Sin}\left[\frac{(R+R0)\,t\,w0}{R}\right]\right),\ (R+R0)\,w0\left(\mathrm{Cos}[t\,w0]-\mathrm{Cos}\left[\frac{(R+R0)\,t\,w0}{R}\right]\right)\right\}$$

```
moneta0 = Plot[
  {
   Sqrt[4 - x^2], -Sqrt[4 - x^2]
  },
  {x, -6, 6},
  AspectRatio → Automatic,
  PlotRange → Automatic,
  PlotStyle → {
    Blue
   },
  AxesLabel →
   {
    Style["x", Small, Red],
    Style["y", Small, Red]
   },
  Ticks → None
 ]
```



```
Show[
 {
  epi[1, 2],
  moneta0
 },
 AxesLabel →
  {
   Style["x", Small, Red],
   Style["y", Small, Blue]
  }
]
```



Accelerazione vettoriale

```
a[t_, R_, R0_, w0_] = ∂t v[t, R, R0, w0] // Simplify
```

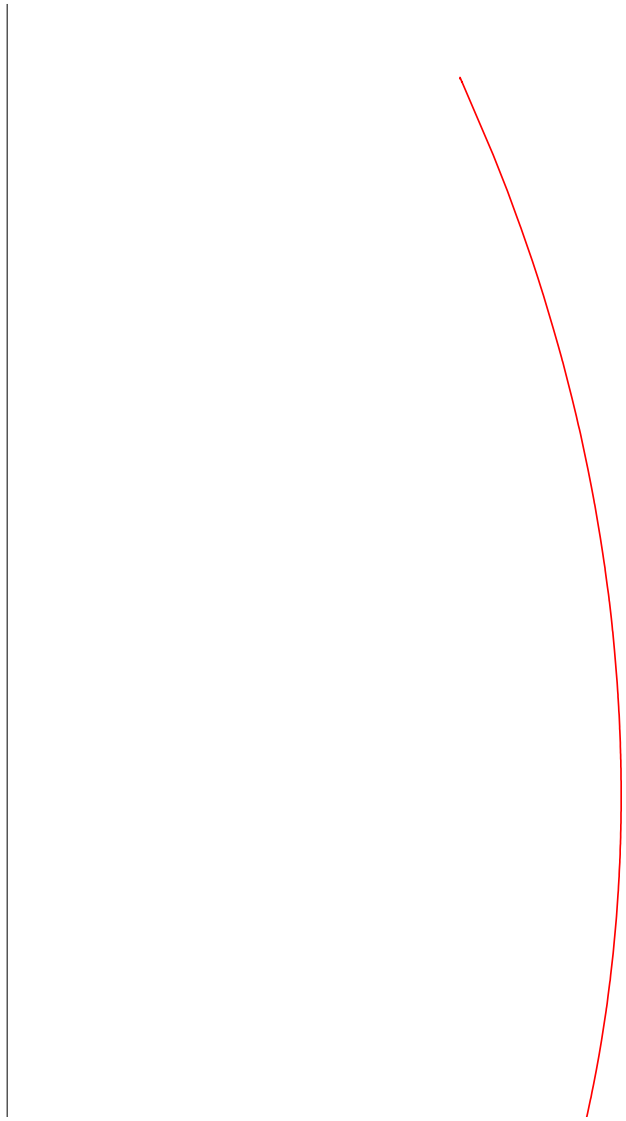
$$\left\{(R+R0)\,w0\left(-w0\,\mathrm{Cos}[t\,w0]+\frac{(R+R0)\,w0\,\mathrm{Cos}\left[\frac{(R+R0)\,t\,w0}{R}\right]}{R}\right),\ (R+R0)\,w0\left(-w0\,\mathrm{Sin}[t\,w0]+\frac{(R+R0)\,w0\,\mathrm{Sin}\left[\frac{(R+R0)\,t\,w0}{R}\right]}{R}\right)\right\}$$

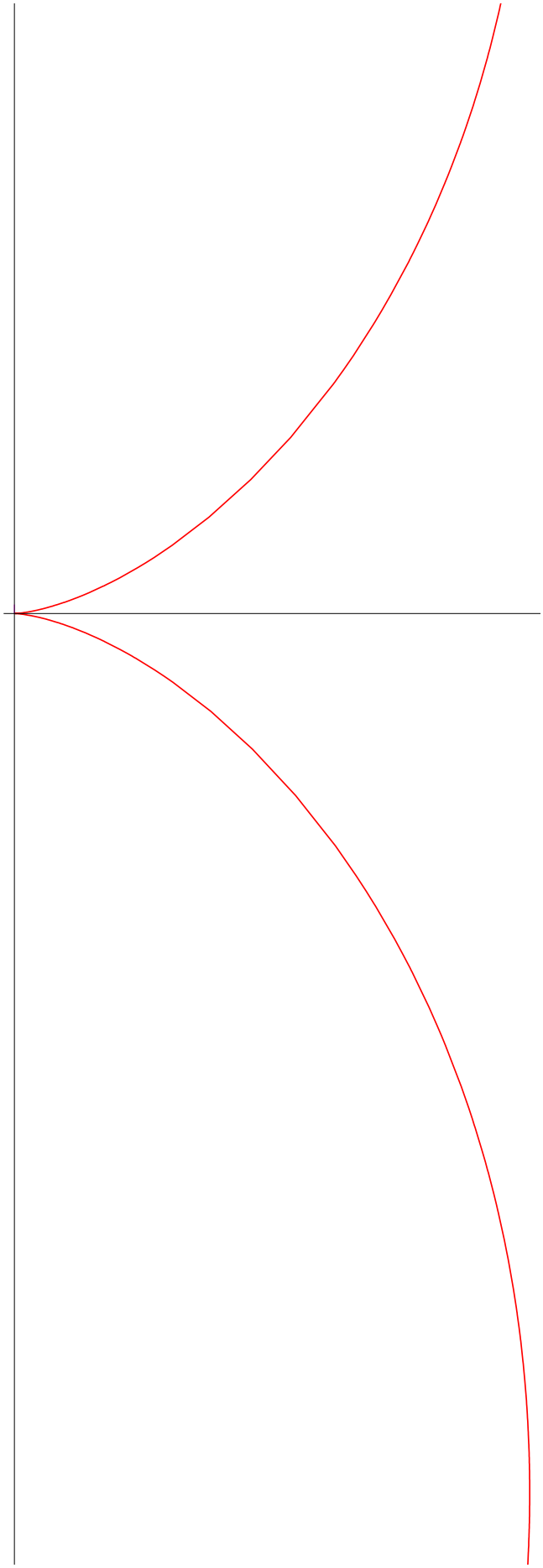
```
a[0, 1, 2, 1]
```

{6, 0}

```
epimin[R_, R0_] := ParametricPlot[
  xx[t, R, R0, 1],
  {t, -1, 1},
  PlotRange → {Automatic, Automatic},
  PlotStyle → {
    Red,
    Thickness[0.00255]
   },
  Ticks →
   {
    {
     {R0, "R0"}, {-R0, "-R0"}
    },
    {
     {R0 + R, "R0+\!\(\*
StyleBox[\"R\",\nFontSlant->\"Italic\"]\)"}, {-(R0 + R), "-(R0+\!\(\*
StyleBox[\"R\",\nFontSlant->\"Italic\"]\)\!\(\*
StyleBox[\")\",\nFontSlant->\"Italic\"]\)"}
    }
   }
 ]
```

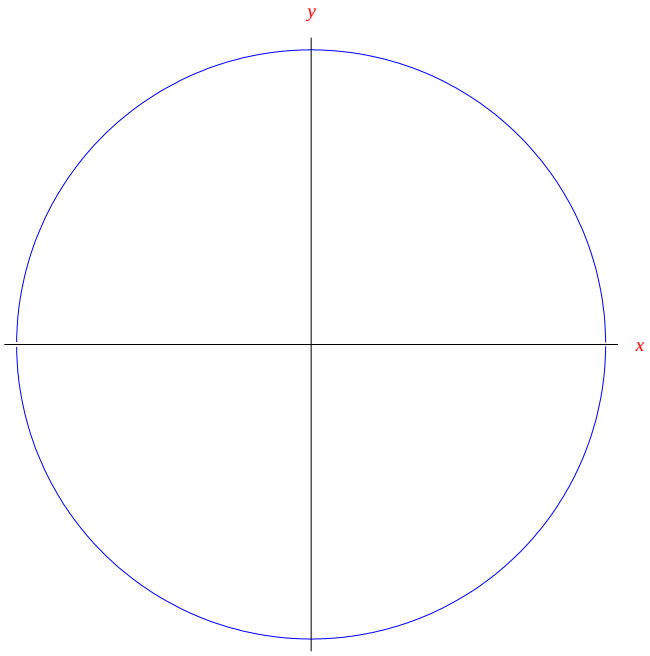
```
epimin[1, 2]
```



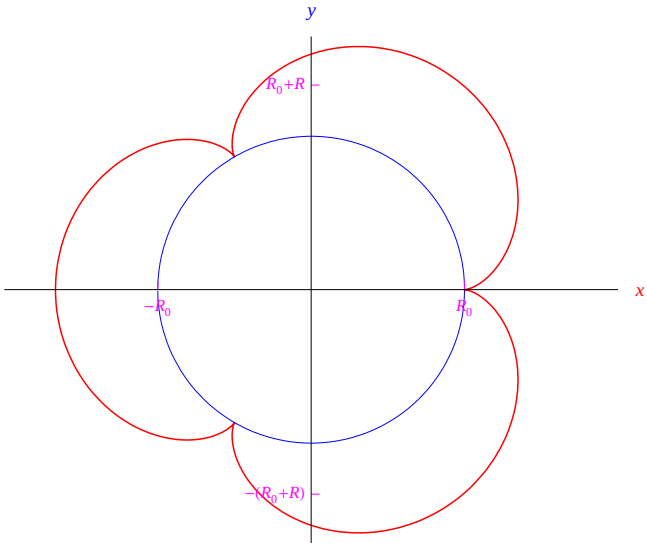


```
epi1[R_, R0_] := ParametricPlot[
  xx[t, R, R0, 1],
  {t, 0, 2 π},
  PlotRange → {{-6, 6}, Automatic},
  PlotStyle → {
    Red,
    Thickness[0.00255]
   },
  Ticks →
   {
    {
     {R0, "R0"}, {-R0, "-R0"}
    },
    {
     {R0 + R, "R0+R"}, {-(R0 + R), "-(R0+R)"}
    }
   }
 ]
```

```
moneta01 = Plot[
  {
   Sqrt[9 - x^2], -Sqrt[9 - x^2]
  },
  {x, -6, 6},
  AspectRatio → Automatic,
  PlotRange → Automatic,
  PlotStyle → {
    Blue
   },
  AxesLabel →
   {
    Style["x", Small, Red],
    Style["y", Small, Red]
   },
  Ticks → None
 ]
```

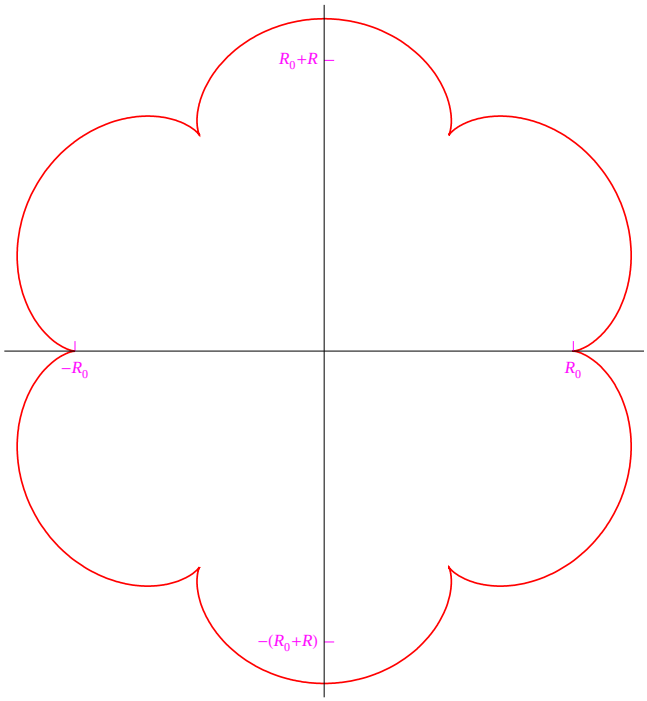


```
tripla = Show[
  {
   epi1[1, 3],
   moneta01
  },
  AxesLabel →
   {
    Style["x", Small, Red],
    Style["y", Small, Blue]
   }
 ]
```

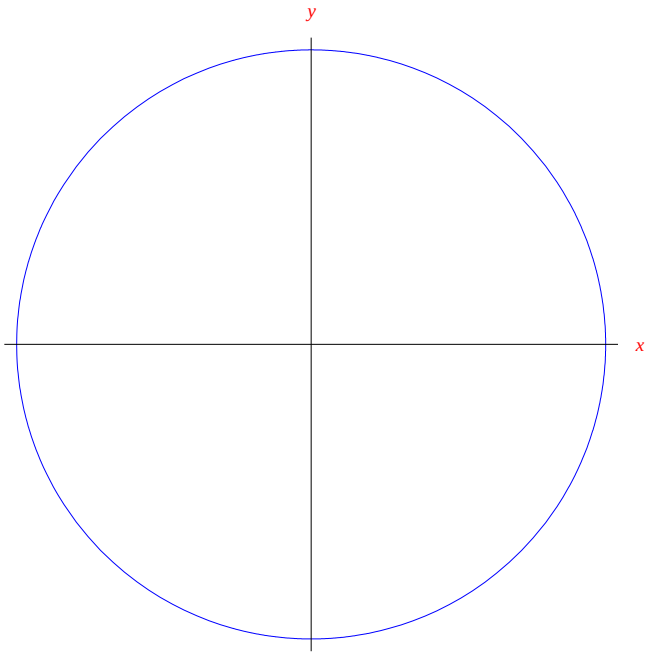



```
epi2[R_, R0_] := ParametricPlot[
  xx[t, R, R0, 1],
  {t, 0, 2 π},
  PlotRange → {Automatic, Automatic},
  PlotStyle → {
    Red,
    Thickness[0.00255]
   },
  Ticks →
   {
    {
     {R0, "R0"}, {-R0, "-R0"}
    },
    {
     {R0 + R, "R0+\!\(\*
StyleBox[\"R\",\nFontSlant->\"Italic\"]\)"}, {-(R0 + R), "-(R0+\!\(\*
StyleBox[\"R\",\nFontSlant->\"Italic\"]\)\!\(\*
StyleBox[\")\",\nFontSlant->\"Italic\"]\)"}
    }
   }
 ]
```

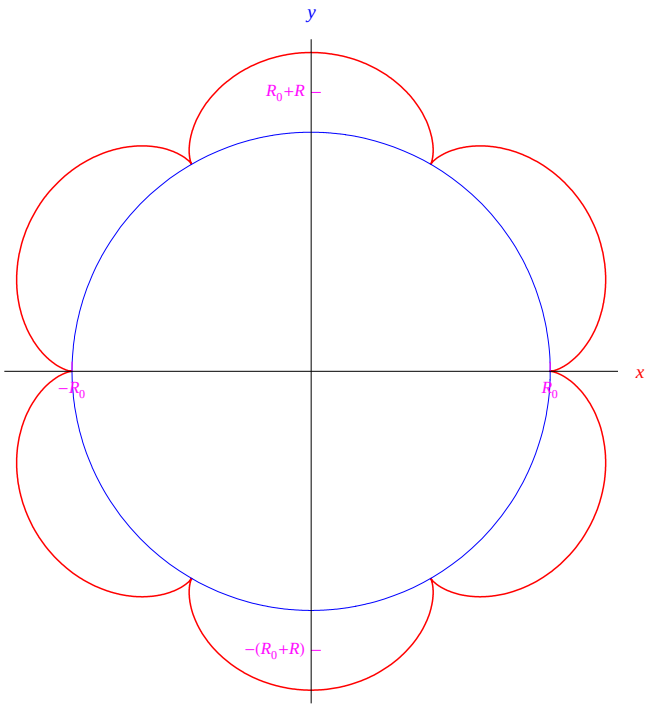
```
epi2[1, 6]
```



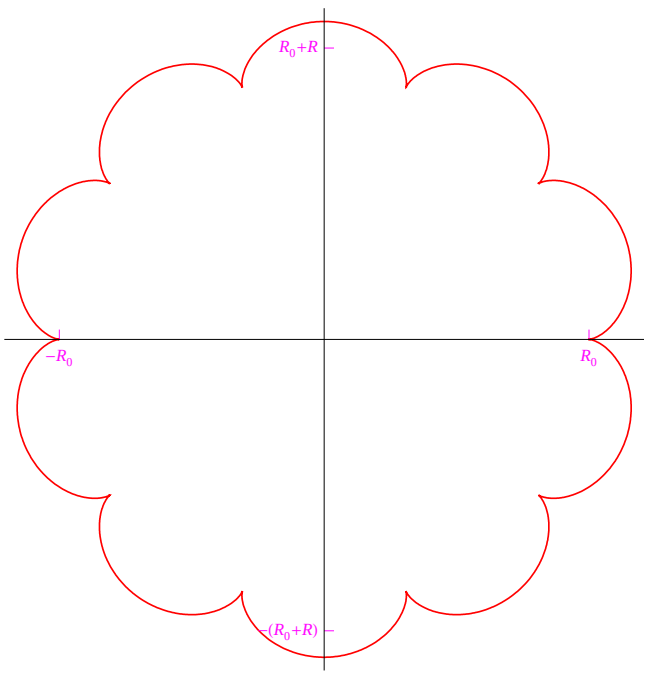
```
moneta02 = Plot[
  {
   Sqrt[36 - x^2], -Sqrt[36 - x^2]
  },
  {x, -6, 6},
  AspectRatio → Automatic,
  PlotRange → Automatic,
  PlotStyle → {
    Blue
   },
  AxesLabel →
   {
    Style["x", Small, Red],
    Style["y", Small, Red]
   },
  Ticks → None
 ]
```



```
Show[
 {
  epi2[1, 6],
  moneta02
 }
 ,
 AxesLabel →
  {
   Style["x", Small, Red],
   Style["y", Small, Blue]
  }
]
```



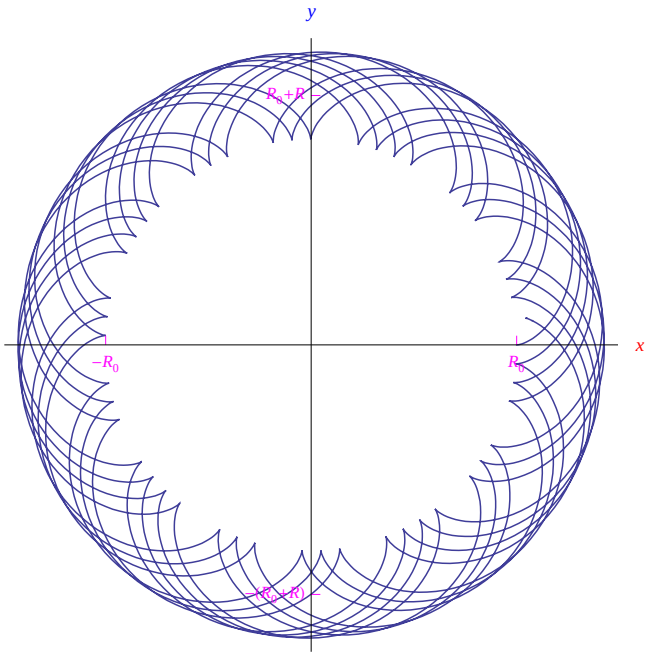

```
epi2[1, 10]
```



```
nonperiodica[a_, R_, R0_] := ParametricPlot[
  xx[t, R, R0, 1],
  {t, 0, a},
  PlotRange → {{-7, 7}, {-7, 7}},
  PlotStyle → {
    Thickness[0.00255]
   },
  Ticks → {
    {
     {R0, "R0"}, {-R0, "-R0"}
    },
    {
     {R0 + R, "R0+\!\(\*
StyleBox[\"R\",\nFontSlant->\"Italic\"]\)"}, {-(R0 + R), "-(R0+\!\(\*
StyleBox[\"R\",\nFontSlant->\"Italic\"]\)\!\(\*
StyleBox[\")\",\nFontSlant->\"Italic\"]\)"}
    }
   },
  AxesLabel →
   {
    Style["x", Small, Red],
    Style["y", Small, Blue]
   }
 ]
```

La traiettoria è periodica se e solo se il rapporto tra i raggi è un intero naturale.

```
nonperiodica[20 π, 1, Sqrt[7 π]]
```

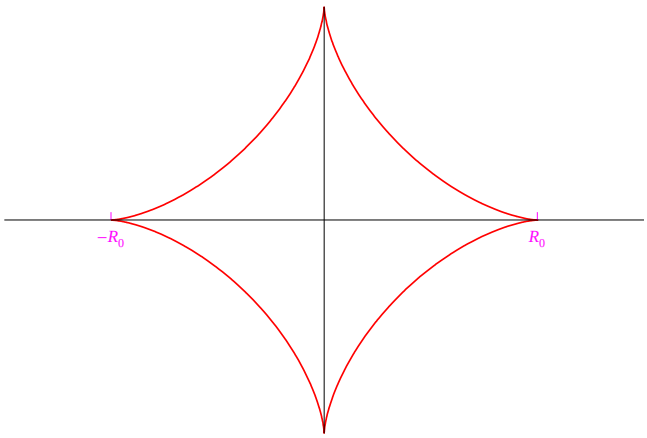


Ipocicloide

Il cerchio di raggio R ruota sul bordo del cerchio di raggio R_0 dall'interno. La routine è la medesima di quella dell'epicicloide, a patto di utilizzare valori $R < 0$.

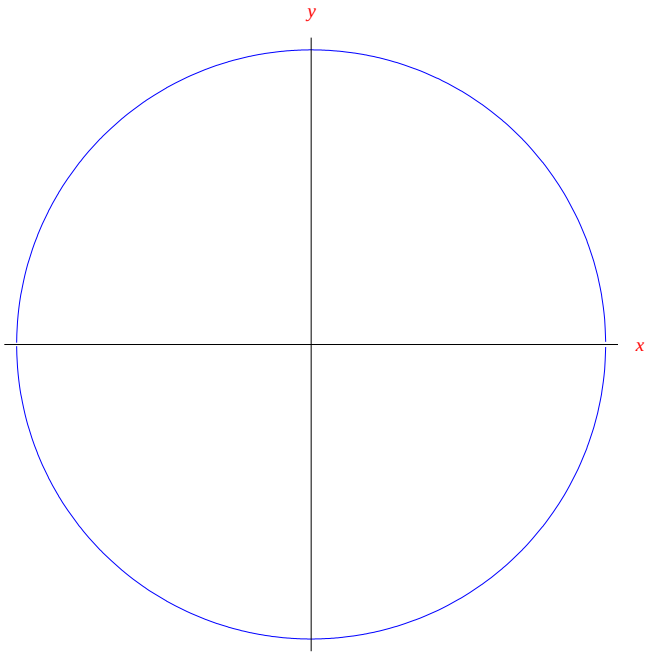
```
ipo[R_, R0_] := ParametricPlot[
  xx[t, R, R0, 1],
  {t, 0, 2 π},
  PlotRange → {{-6, 6}, {-4, 4}},
  PlotStyle → {
    Red,
    Thickness[0.00255]
   },
  Ticks →
   {
    {
     {R0, "R0"}, {-R0, "-R0"}
    },
    None
   }
 ]
```

```
ipo[-1, 4]
```

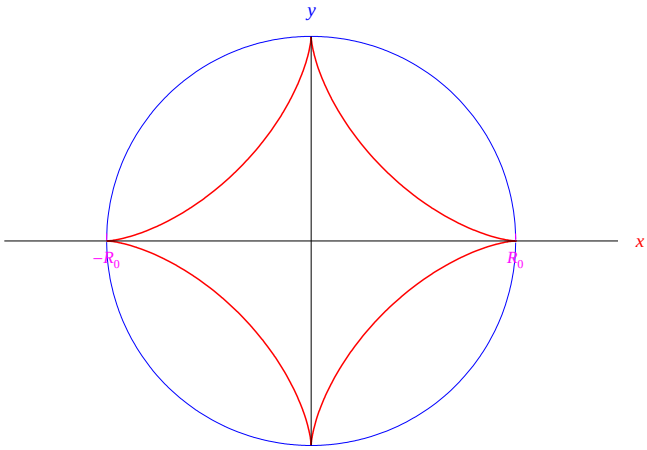



```
SetDirectory["G:\\Siti\\extrabyte2\\Mathematica\\GEOMETRIA_DIFF"];
```

```
cerchiomax = Plot[
  {
   Sqrt[16 - x^2], -Sqrt[16 - x^2]
  },
  {x, -14, 14},
  AspectRatio → Automatic,
  PlotRange → Automatic,
  PlotStyle → {
    Blue
   },
  AxesLabel →
   {
    Style["x", Small, Red],
    Style["y", Small, Red]
   },
  Ticks → None
 ]
```



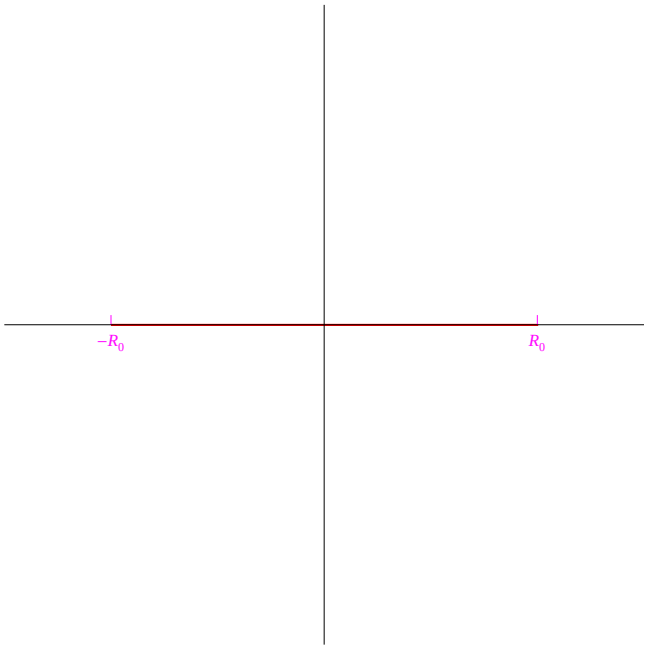
```
ipociclo1 = Show[
  {
   ipo[-1, 4],
   cerchiomax
  },
  AxesLabel →
   {
    Style["x", Small, Red],
    Style["y", Small, Blue]
   }
 ]
```




```
Export["ipociclo1.eps", ipociclo1];
```

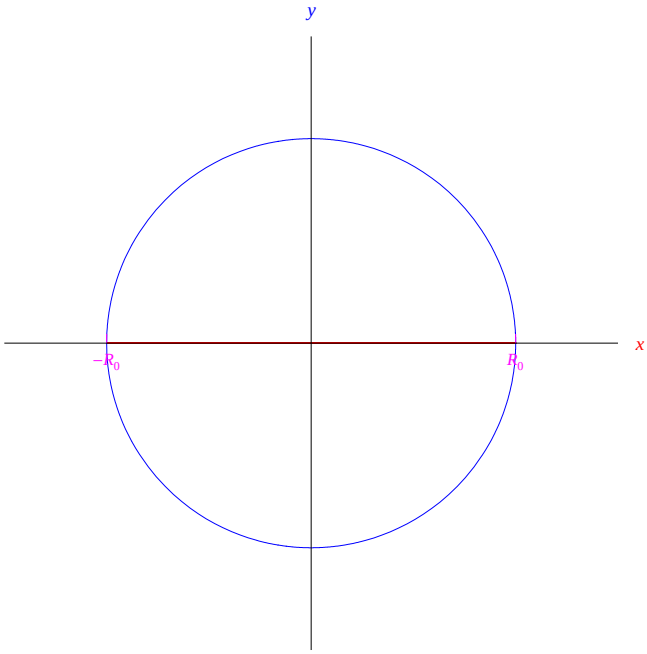
```
ipo1[R_, R0_] := ParametricPlot[
  xx[t, R, R0, 1],
  {t, 0, 2 π},
  PlotRange → {{-1.5, 1.5}, {-1.5, 1.5}},
  PlotStyle → {
    Red,
    Thickness[0.00255]
   },
  Ticks →
   {
    {
     {R0, "R0"}, {-R0, "-R0"}
    },
    None
   }
 ]
```

```
ipo1[-1 / 2, 1]
```




```
cerchiounitario = Plot[
  {
   Sqrt[1 - x^2], -Sqrt[1 - x^2]
  },
  {x, -1, 1},
  AspectRatio → Automatic,
  PlotRange → Automatic,
  PlotStyle → {
    Blue
   },
  AxesLabel →
   {
    Style["x", Small, Red],
    Style["y", Small, Red]
   },
  Ticks → None
 ];
```

```
ipociclo2 = Show[
  {
   ipo1[-1 / 2, 1],
   cerchiounitario
  },
  AxesLabel →
   {
    Style["x", Small, Red],
    Style["y", Small, Blue]
   }
 ]
```




```
Export["armonico.eps", %];
```

```
vel[t_, R_, R0_] = ∂t xx[t, R, R0, 1]
```

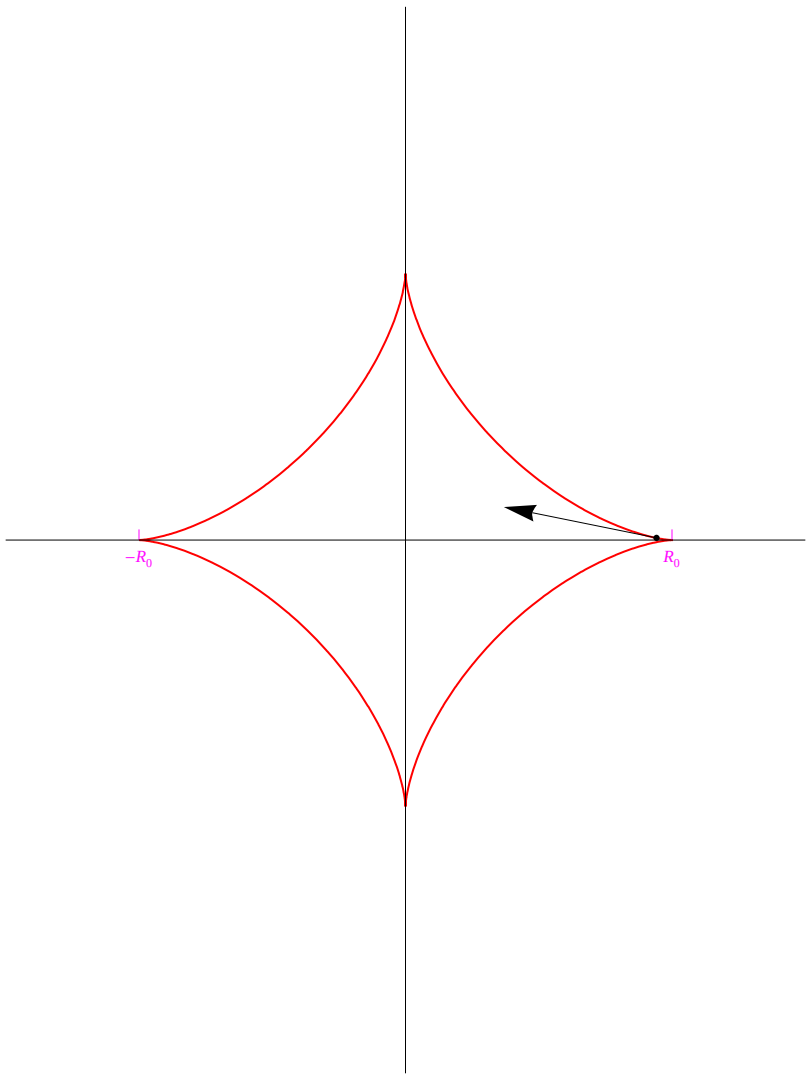
$$\left\{-(R+R0)\,\mathrm{Sin}[t]+(R+R0)\,\mathrm{Sin}\left[\frac{(R+R0)\,t}{R}\right],\ (R+R0)\,\mathrm{Cos}[t]-(R+R0)\,\mathrm{Cos}\left[\frac{(R+R0)\,t}{R}\right]\right\}$$

```
vel[1, -1, 4.]
```

{-2.94777, 4.59088}

```
ipo2[τ_] := ParametricPlot[
  xx[t, -1, 4, 1],
  {t, 0, 2 π},
  PlotRange → {{-6, 6}, {-8, 8}},
  PlotStyle → {
    Red,
    Thickness[0.00255]
   },
  Ticks →
   {
    {
     {4, "R0"}, {-4, "-R0"}
    },
    None
   },
  Epilog →
   {
    Point[xx[τ, -1, 4, 1]],
    Arrow[{xx[τ, -1, 4, 1], xx[τ, -1, 4, 1] + vel[τ, -1, 4]}]
   },
  ImageSize → {500, 500}
 ]
```

```
ipo2[0.2]
```



```
tb = Table[ipo2[τ], {τ, 0, 2 π, 0.1}];
```

```
Export["ipoanimata.gif", tb]
```

```
ipoanimata.gif
```

Applicazioni alla funzione zeta di Riemann

```
SetDirectory["G:\\Siti\\extrabyte2\\Mathematica\\GEOMETRIA_DIFF"];
```

```
SetOptions[
  {
   Plot,
   ParametricPlot
  },
  TicksStyle → Directive[Magenta, 7],
  FrameStyle → Directive[Magenta, 7]
 ];
```

```
? Zeta
```

Zeta[*s*] gives the Riemann zeta function $\zeta(s)$.
Zeta[*s*, *a*] gives the generalized Riemann zeta function $\zeta(s, a)$. ≫

```
Clear[f, posizione, velocita, φ1, φ2]
```

$$\varphi1[t\_] = \mathrm{Re}\left[\mathrm{Zeta}\left[\frac{1}{2} + i * t\right]\right];\ \varphi2[t\_] = \mathrm{Im}\left[\mathrm{Zeta}\left[\frac{1}{2} + i * t\right]\right];$$

```
posizione[t_] := {φ1[t], φ2[t]}
```

$$\partial_t \varphi1[t]$$

$$i\ \mathrm{Re}'\left[\mathrm{Zeta}\left[\frac{1}{2} + i\ t\right]\right]\ \mathrm{Zeta}'\left[\frac{1}{2} + i\ t\right]$$

```
velocita[t_] = {∂t φ1[t], ∂t φ2[t]};
```

```
posizione[10.]
```

```
{1.5449, -0.115336}
```

```
velocita[10.]
```

```
{(0.00359344 - 0.360907 i) Re′[1.5449 - 0.115336 i],
 (0.00359344 - 0.360907 i) Im′[1.5449 - 0.115336 i]}
```

```
Clear[plotriemann];
```

```
plotriemann[τ_] := ParametricPlot[
  posizione[t],
  {t, 0, 50},
  PlotStyle → Directive[Thickness[0.00255]],
  AxesLabel →
   {
    Style["ξ", Small, Red],
    Style["η", Small, Blue]
   },
  Epilog →
   {
    Red,
    PointSize[0.02],
    Point[posizione[τ]]
   }
 ]
```

```
ttt = Table[plotriemann[τ], {τ, 0, 50}];
```

```
Export["plotriemann.gif", ttt];
```

Analisi Matematica 1 - Calcolo di limiti

■ Calcolo di limiti di funzioni razionali fratte

```
Limit[
 f[x],
 x → x0
]
```

```
Limit[f[x], x → x0]
```

```
Limit[
 (*espressione analitica
  della funzione*)
 (x^2 - 2 * x + 1)/(x - 1),
 (*punto di accumulazione a cui
  tende la variabile x*)
 x → 1
]
```

```
0
```

```
lim[f_, x0_, k_] :=
 Limit[
  (*espressione analitica
   della funzione*)
  f[x],
  (*punto di accumulazione a cui
   tende la variabile x*)
  x → x0,
  (*limite destro: k=-1;
  limite sinistro: k=+1 *)
  Direction → k
 ]
```

Se non siamo interessati ai limiti destro ($x \to x^{+}{}_{0}$) e sinistro (($x \to x^{-}{}_{0}$)), inseriamo k=1, ripetendo poi il calcolo per k=-1.

```
Clear[f]
```

$$f[x\_] := \frac{x^2 - 3x + 2}{x^2 + x - 6}$$

```
lim[f, 2, 1]
```

$$\frac{1}{5}$$

```
lim[f, 2, -1]
```

$$\frac{1}{5}$$

Calcolando il limite con carta e penna, vediamo che il rapporto si presenta nella forma indeterminata 0/0. Per rimuovere l'indeterminazione dobbiamo scomporre in fattori i polinomi a numeratore e denominatore. Possiamo controllare la correttezza dei nostri calcoli, nel seguente modo:

```
g[x_] = f[x] // Factor
```

$$\frac{-1 + x}{3 + x}$$

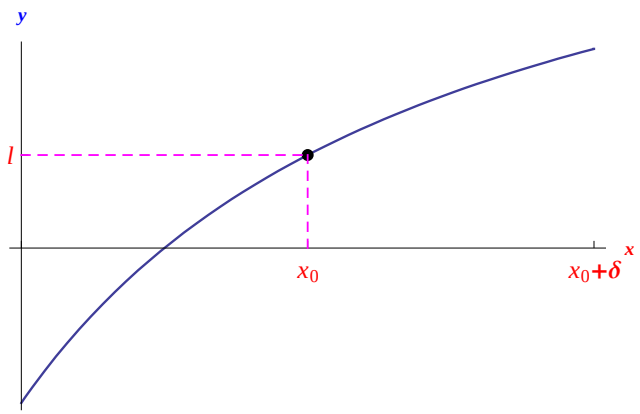
```
lim[g, 2, 1]
```

$$\frac{1}{5}$$

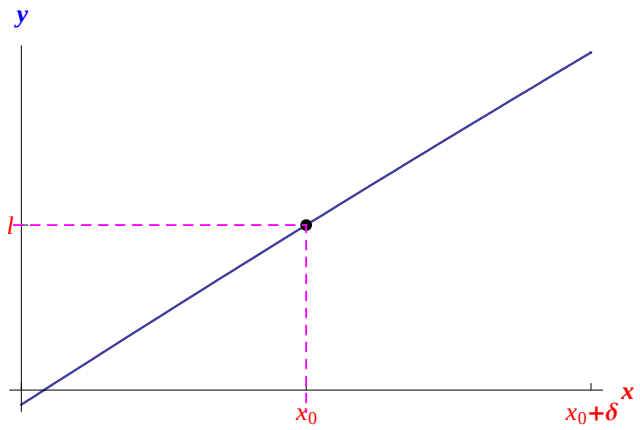
Per avere un'idea più precisa del comportamento della funzione in un intorno di x_0 tracciamo un grafico "locale":

```
local[f_, x0_, δ_] := Block[
  (*dichiaro le variabili locali*)
  {
   xmin,
   xmax,
   plot
  },
  (*esprimo le variabili locali
   in termini delle variabili di input*)
  xmin = x0 - δ;
  xmax = x0 + δ;
  plot = Plot[
    (*scrivo la funzione*)
    f[x],
    (*traccio il grafico in un intorno di x0*)
    {x, xmin, xmax},
    PlotRange → Automatic,
    AxesLabel → {
      Style[x, Small, Bold, Red],
      Style[y, Small, Bold, Blue]
     },
    LabelStyle → Directive[Red, Bold]
    ,
    PlotStyle → {
      Thickness[0.004]
     },
    Ticks → {
      {
```

```
        {x0, "x0"}, {x0 + δ, "x0+δ"}, {x0 - δ, "x0-δ"}
      },
      {
       {lim[f, 2, 1], "l"}
      }
     },

    TicksStyle → Directive[
      FontFamily → "Times New Roman",
      FontSize → 12
     ],
    Epilog → {
      {
       PointSize[0.02], Point[{x0, lim[f, 2, 1]}]
      },
      {
       Dashed,
       Hue[5 / 6],
       Thickness[0.003],
       Line[{{x0, 0}, {x0, lim[f, 2, 1]}}]
      },
      {
       Dashed,
       Hue[5 / 6],
       Thickness[0.003],
       Line[{{0, lim[f, 2, 1]}, {x0, lim[f, 2, 1]}}]
      }

     }
   ]
 ]

local[f, 2, 2]
```

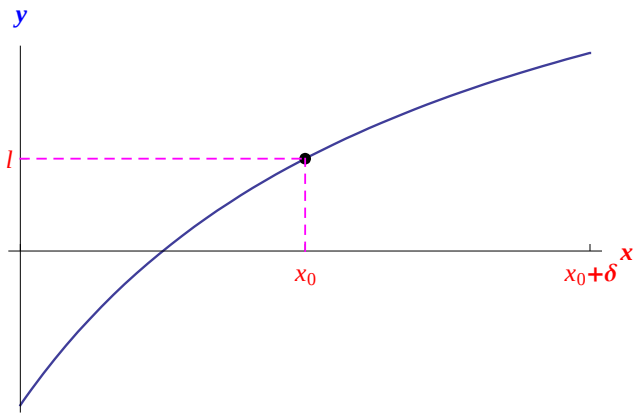


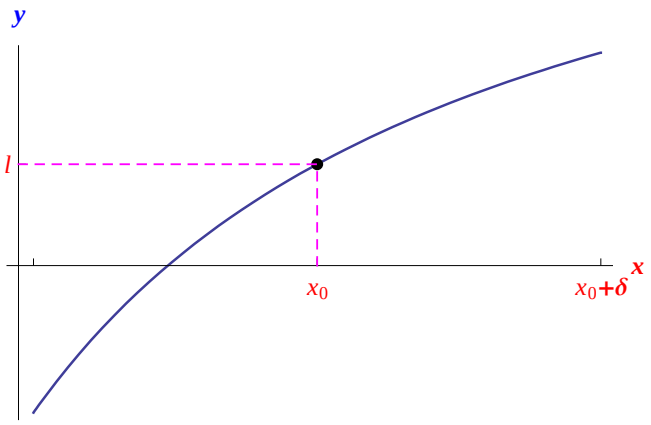
```
local[f, 2, 0.1]
```

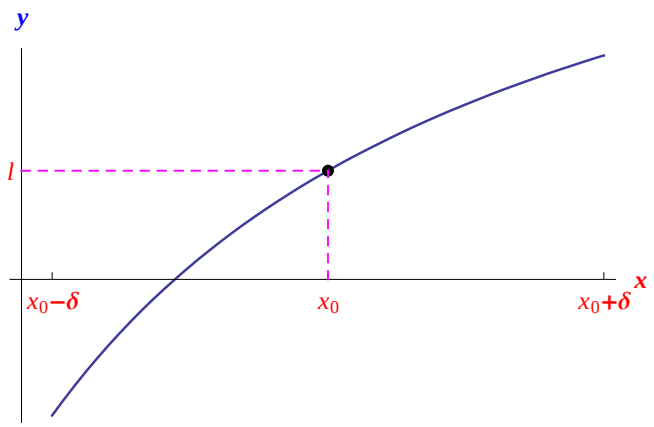


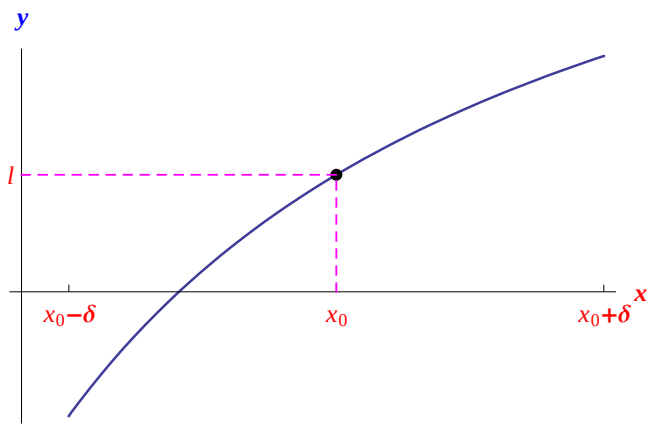
L'utilizzo di un ciclo **Do** ci permette di creare un'animazione grafica per valori decrescenti dell'ampiezza dell'intorno:

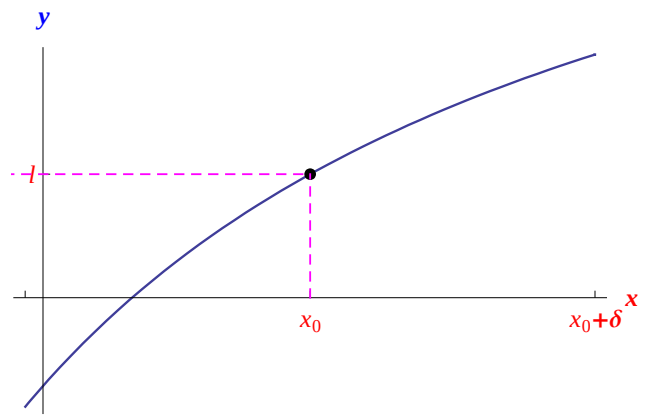
```
animazione = Do[
   (*stampa a video il grafico
    per assegnati valori di δ*)
   Print[
    local[f, 2, δ]],
   (*iteratore*)
   {δ, 2, 0.2, -0.1}
  ];
```

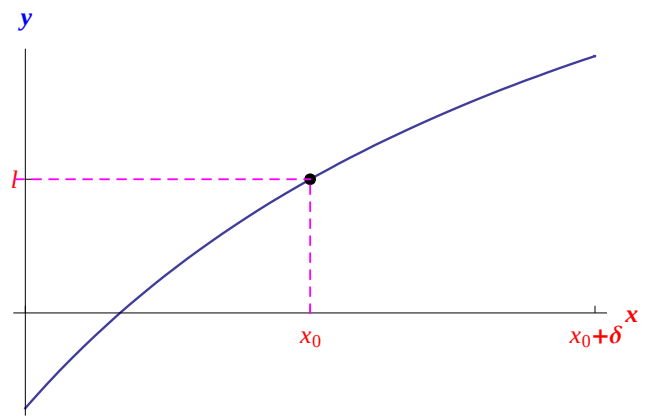


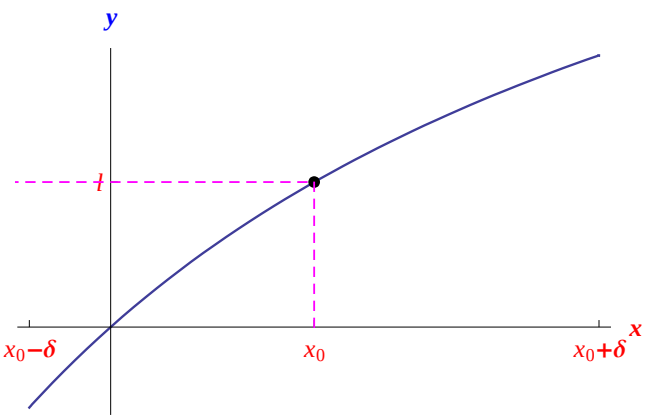


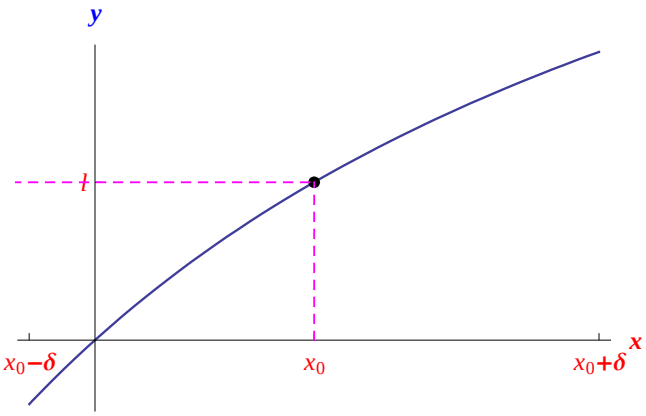

y
l
x
$x_0-\delta$
x_0
$x_0+\delta$

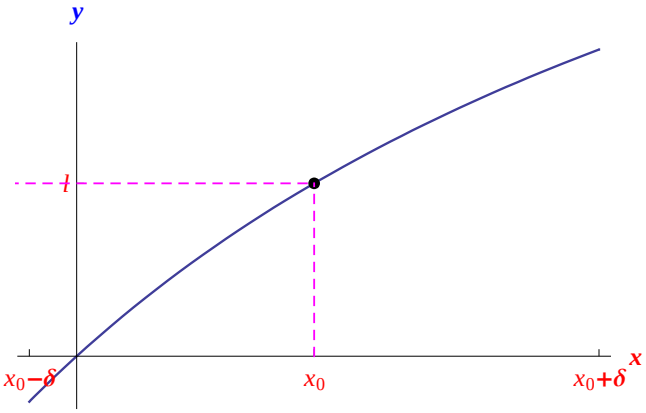

y
l
x
$x_0-\delta$
x_0
$x_0+\delta$

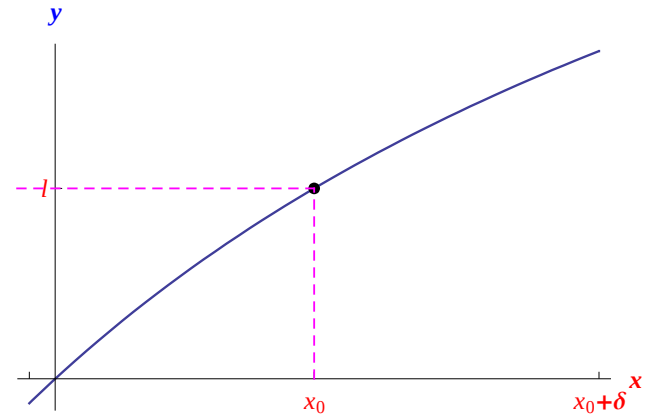

y
l
x
x_0
$x_0+\delta$

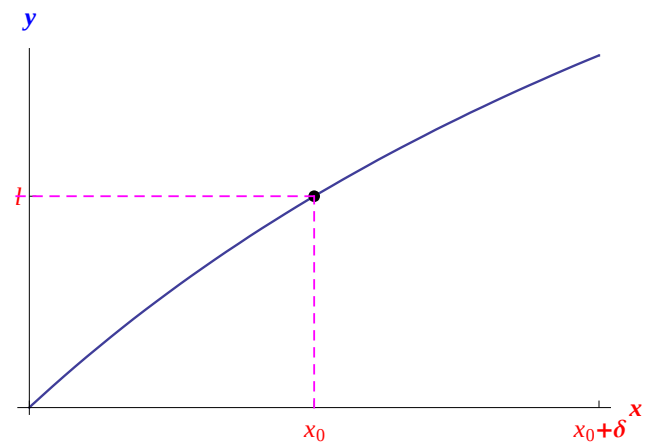

y
l
x
x_0
$x_0+\delta$

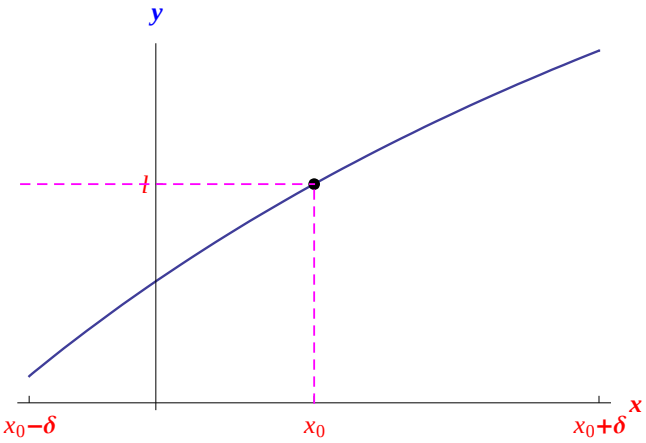

y
l
x
$x_0-\delta$
x_0
$x_0+\delta$

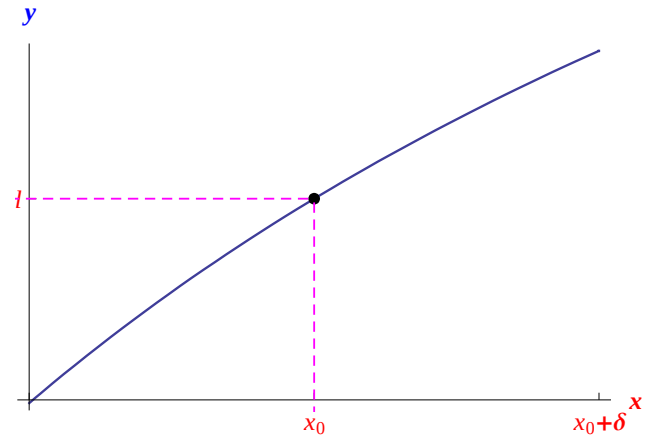

y
l
x
$x_0-\delta$
x_0
$x_0+\delta$

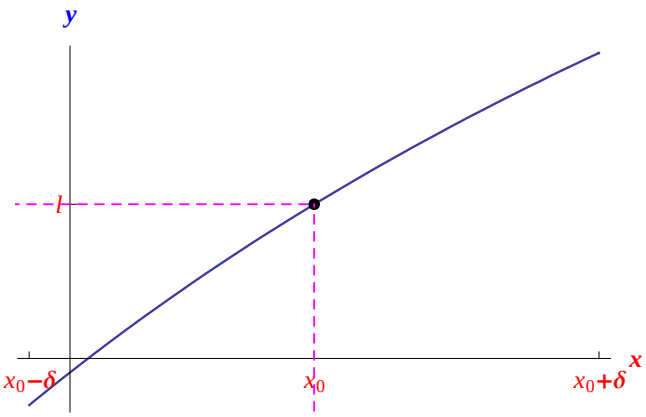

y
l
x
$x_0-\delta$
x_0
$x_0+\delta$

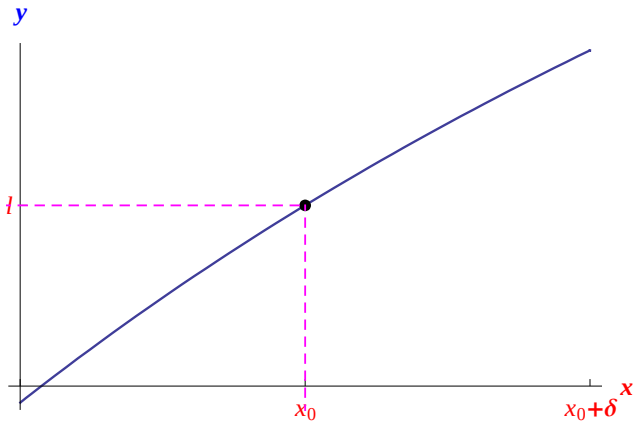

y
l
x
x_0
$x_0+\delta$

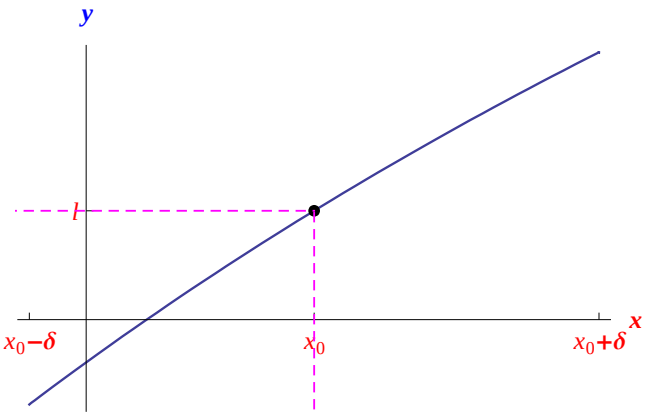

y
l
x
x_0
$x_0+\delta$

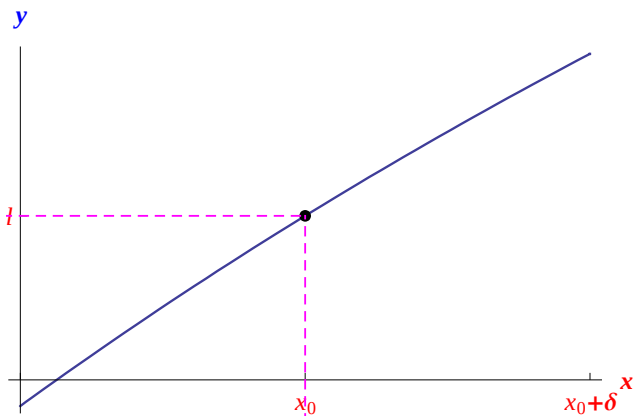

y
l
x
$x_0-\delta$
x_0
$x_0+\delta$

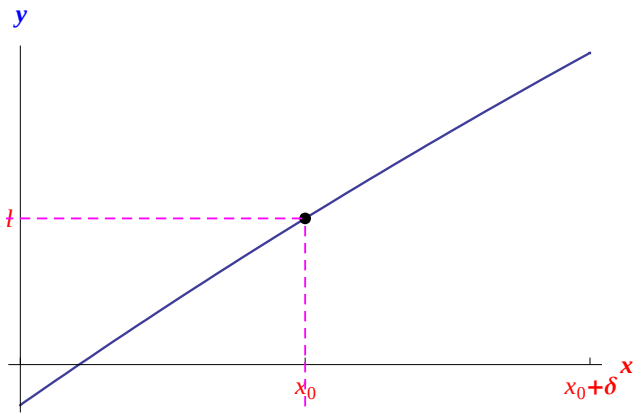

y
l
x
x_0
$x_0+\delta$

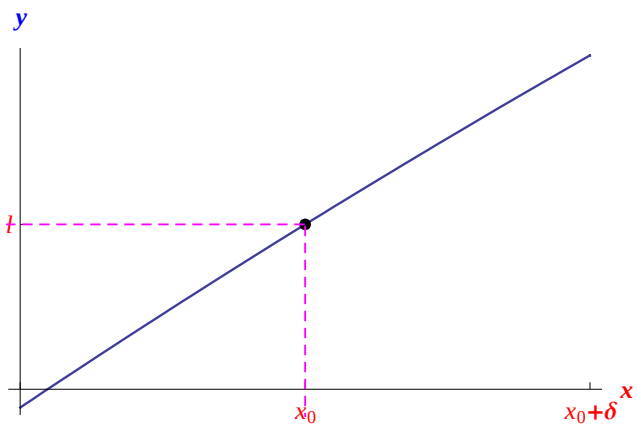

y
l
x
$x_0-\delta$
x_0
$x_0+\delta$


y
l
x
x_0
$x_0+\delta$


y
l
x
$x_0-\delta$
x_0
$x_0+\delta$


y
l
x
x_0
$x_0+\delta$


y
l
x
x_0
$x_0+\delta$




```
Clear[f]
```

$$f[x\_] := \frac{x^2 - x - 6}{x^3 + 5x^2 + 8x + 4}$$

```
lim[f, -2, 1]
```

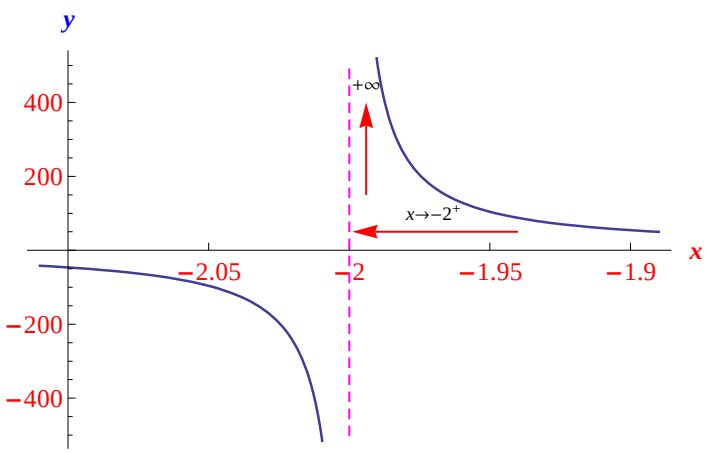
$-\infty$

```
lim[f, -2, -1]
```

∞

```
Clear[local]

local[f_, x0_, δ_] := Block[
  (*dichiaro le variabili locali*)
  {
   xmin,
   xmax,
   plot
  },
  (*esprimo le variabili locali
   in termini delle variabili di input*)
  xmin = x0 - δ;
  xmax = x0 + δ;
  plot = Plot[
    (*scrivo la funzione*)
    f[x],
    (*traccio il grafico in un intorno di x0*)
    {x, xmin, xmax},
    PlotRange → Automatic,
    AxesLabel → {
      Style["x", Medium, Bold, Red],
      Style["y", Medium, Bold, Blue]
     },
    LabelStyle → Directive[Red, Bold]
    ,
    PlotStyle → {
      Thickness[0.004]
     },
    Ticks → {
      {-2.05, -2, -1.95, -1.90},
      Automatic
     },
    TicksStyle → Directive[
```

```
      FontFamily → "Times New Roman",
      FontSize → 12
     ],
    Exclusions →
     {
      x = x0
     },
    Epilog → {

      {
       Dashed,
       Hue[5 / 6],
       Thickness[0.003],
       Line[{{x0, -500}, {x0, 500}}]
      },
      {
       Hue[1],
       Thickness[0.003],
       Arrow[{{-1.94, 50}, {-1.999, 50}}]
      },
      {
       Hue[1],
       Thickness[0.003],
       Arrow[{{-1.994, 150}, {-1.994, 400}}]
      },
      Text[
       "x→-2⁺",
       {-1.97, 100}
      ],
      Text[
       "+∞",
       {-1.994, 450}
      ]
     }
   ]
 ]

local[f, -2, -0.11]
```



```
Clear[
 (*cancello f dal kernel*)
 f,
 (*la variabili x va in conflitto con x0,
 quindi la rimuovo dal kernel*)
 x
]
```

$$f[x\_] := \frac{x^4 - 8x^2 + 16}{x^3 - 8}$$

```
lim[f, 2, -1]
```

0

$$g[x\_] = \frac{x^4 - 8x^2 + 16}{x^3 - 8} \text{ // Factor}$$

$$\frac{(-2 + x)\,(2 + x)^2}{4 + 2x + x^2}$$

```
lim[g, 2, -1]
```

0

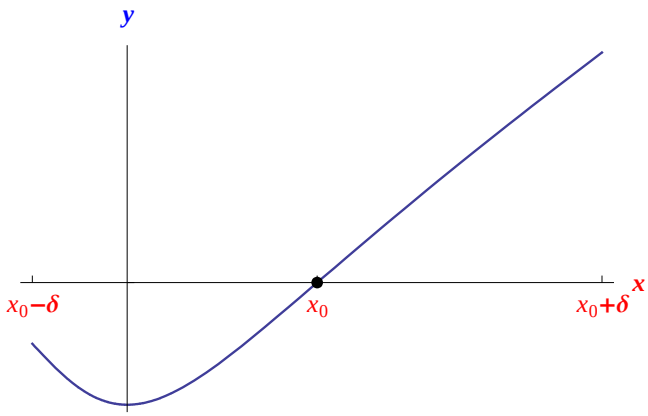
```
lim[g, 2, 1]
```

0

```
Clear[local]
```

```
local[f_, x0_, δ_] := Block[
  (*dichiaro le variabili locali*)
  {
   xmin,
   xmax,
   plot
  },
  (*esprimo le variabili locali
   in termini delle variabili di input*)
  xmin = x0 - δ;
  xmax = x0 + δ;
  plot = Plot[
    (*scrivo la funzione*)
    f[x],
    (*traccio il grafico in un intorno di x0*)
    {x, xmin, xmax},
    PlotRange → Automatic,
    AxesLabel → {
      Style[x, Medium, Bold, Red],
      Style[y, Medium, Bold, Blue]
     },
    LabelStyle → Directive[Red, Bold]
    ,
    PlotStyle → {
      Thickness[0.004]
     },
    Ticks → {
      {
       {x0, "x0"}, {x0 + δ, "x0+δ"}, {x0 - δ, "x0-δ"}
      },
      {
       {lim[f, 2, 1], "l"}
      }
     },

    TicksStyle → Directive[
      FontFamily → "Times New Roman",
      FontSize → 12
     ],
    Epilog → {
      {
       PointSize[0.02], Point[{x0, lim[f, 2, 1]}]
      }

     }
   ]
 ]
```

```
local[f, 2, 3]
```



Funzioni divergenti al finito

```
Clear[f, plot]
```

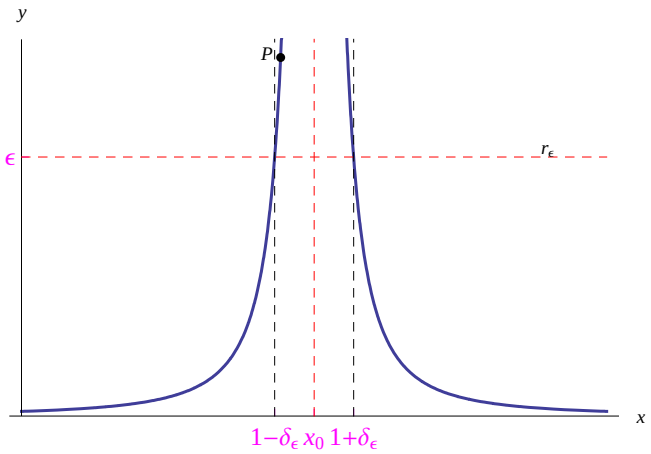
$$f[x\_, x0\_] := \frac{1}{(x - x0)^2}$$

$$\delta[\epsilon\_] := \frac{1}{\sqrt{\epsilon}}$$

```
plot[ϵ_] := Plot[
  f[x, 1],
  {x, 0, 2},
  PlotRange →
   {
    0, 80
   },
  AxesLabel →
   {
    "x", "y"
   },
  PlotStyle → Thickness[0.005],
  Ticks →
   {
    (*asse x*)
    {
     {1, "x0"},
     {1 - δ[ϵ], "1-δϵ"},
     {1 + δ[ϵ], "1+δϵ"}
    },
    (*asse y*)
    {
     {ϵ, "ϵ"}
    }
   },
  TicksStyle → Directive[
    Hue[5 / 6], 12
   ],
  Exclusions →
   {
    x = 1
   },
```

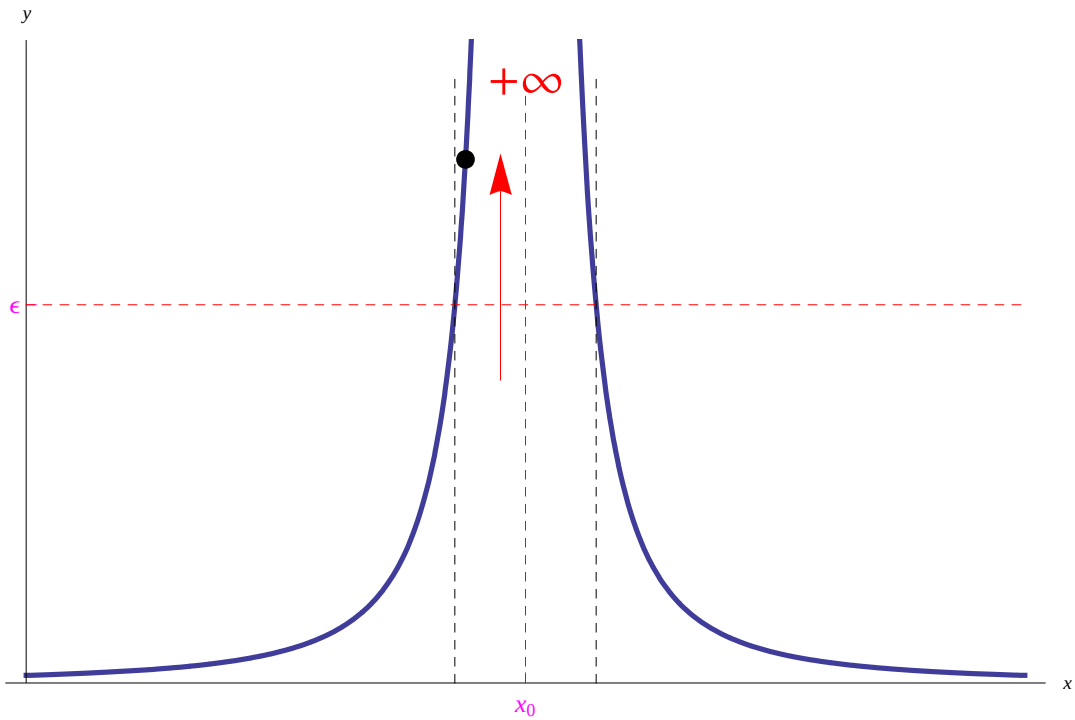
```
  Epilog →
   {
    (*asintoto verticale*)
    {
     RGBColor[1, 0, 0], Dashed, Line[{{1, 0}, {1, 80}}]
    },
    (*retta orizzontale rϵ*)
    {
     RGBColor[1, 0, 0], Dashed, Line[{{0, ϵ}, {2, ϵ}}]
    },
    Text["rϵ", {1.8, 57}],
    (*generico punto del grafico con x0-δϵ<x<x+δϵ*)
    {
     PointSize[0.015], Point[{1 - 0.85 δ[ϵ], f[1 - 0.85 δ[ϵ], 1]}]
    },
    Text["P", {1 - 1.20 δ[ϵ], 77}],
    (*linee verticali per delimitare l'intorno di x0*)
    {
     Dashed, Line[{{1 - δ[ϵ], 0}, {1 - δ[ϵ], 80}}]
    },
    {
     Dashed, Line[{{1 + δ[ϵ], 0}, {1 + δ[ϵ], 80}}]
    }
   }
 ]
```

```
plot[55]
```




```
(*replot per l'animazione grafica*)

Clear[plot]

plot[ϵ_] := Plot[
  f[x, 1],
  {x, 0, 2},
  PlotRange →
   {
    0, 85
   },
  AxesLabel →
   {
    "x", "y"
   },
```

```
  PlotStyle → Thickness[0.005],
  (*aumento la dimensione dei fotogrammi*)
  ImageSize →
   {
    500, 500
   },
  Ticks →
   {
    (*asse x*)
    {
     {1, "x0"}
    },
    (*asse y*)
    {
     {ϵ, "ϵ"}
    }
   },
  TicksStyle → Directive[
    Hue[5 / 6], 12
   ],
  Exclusions →
   {
    x = 1
   },

  Epilog →
   {
    (*asintoto verticale*)
    {
     RGBColor[1, 0, 0], Dashed, Line[{{1, 0}, {1, 80}}]
    },
    (*retta orizzontale rϵ*)
    {
     RGBColor[1, 0, 0], Dashed, Line[{{0, ϵ}, {2, ϵ}}]
    },
    (*generico punto del grafico con x0-δϵ<x<x+δϵ*)
    {
     PointSize[0.018], Point[{1 - 0.85 δ[ϵ], f[1 - 0.85 δ[ϵ], 1]}]
    },
    (*linee verticali per delimitare l'intorno di x0*)
    {
     Dashed, Line[{{1 - δ[ϵ], 0}, {1 - δ[ϵ], 80}}]
    },
    {
     Dashed, Line[{{1 + δ[ϵ], 0}, {1 + δ[ϵ], 80}}]
    },
    (*freccia +∞*)
    {
     RGBColor[1, 0, 0], Arrow[{{0.95, ϵ - 10}, {0.95, ϵ + 20}}]
    },
    (*+∞*)
    Text[Style["+∞", Large, RGBColor[1, 0, 0]], {1, 80}]
   }
 ]

plot[50]
```



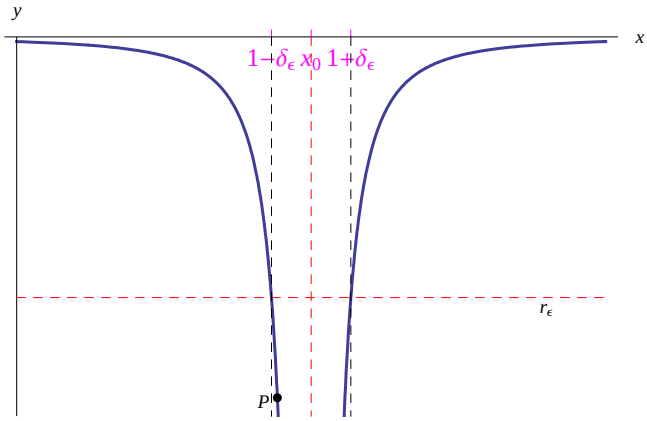

```
movie = Table[
   plot[ϵ],
   {ϵ, 10, 58}
  ];

(*l'output va esportato in formato gif, dopo aver settato la directory
 di destinazione con il comando SetDiretcory[]*)

(*ciclo Do per riprodurre in notebook l'animazione*)

Do[
 Print[
  plot[ϵ]
 ],
 {ϵ, 10, 11}
]

Clear[f, plot, f]
```

$$f[x\_, x0\_] := -\frac{1}{(x - x0)^2}$$

$$\delta[\epsilon\_] := \frac{1}{\sqrt{\epsilon}}$$

```
plot[ϵ_] := Plot[
  f[x, 1],
  {x, 0, 2},
  PlotRange →
   {
    -80, 0
   },
  AxesLabel →
   {
    "x", "y"
   },
  PlotStyle → Thickness[0.005],
  Ticks →
   {
    (*asse x*)
    {
     {1, "x0"},
     {1 - δ[ϵ], "1-δϵ"},
     {1 + δ[ϵ], "1+δϵ"}
    },
    (*asse y*)
    {
     {ϵ, "ϵ"}
    }
   },
  TicksStyle → Directive[
    Hue[5 / 6], 12
   ],
  Exclusions →
   {
    x = 1
   },

  Epilog →
   {
    (*asintoto verticale*)
    {
     RGBColor[1, 0, 0], Dashed, Line[{{1, 0}, {1, -80}}]
    },
    (*retta orizzontale rϵ*)
    {
     RGBColor[1, 0, 0], Dashed, Line[{{0, -ϵ}, {2, -ϵ}}]
    },
    Text["rϵ", {1.8, -57}],
    (*generico punto del grafico con x0-δϵ<x<x+δϵ*)
    {
     PointSize[0.015], Point[{1 - 0.85 δ[ϵ], f[1 - 0.85 δ[ϵ], 1]}]
    },
    Text["P", {1 - 1.20 δ[ϵ], -77}],
    (*linee verticali per delimitare l'intorno di x0*)
    {
     Dashed, Line[{{1 - δ[ϵ], 0}, {1 - δ[ϵ], -80}}]
    },
    {
     Dashed, Line[{{1 + δ[ϵ], 0}, {1 + δ[ϵ], -80}}]
    }
   }
 ]
```

```
plot[55]
```




```
Export["divergente_neg.eps", plot[55]]
```

divergente_neg.eps

```
(*replot per l'animazione grafica*)

Clear[plot]

plot[ϵ_] := Plot[
  f[x, 1],
  {x, 0, 2},
  PlotRange →
   {
    -85, 0
   },
  AxesLabel →
   {
    "x", "y"
   },
  PlotStyle → Thickness[0.005],
  (*aumento la dimensione dei fotogrammi*)
  ImageSize →
   {
    500, 500
   },
  Ticks →
   {
    (*asse x*)
    {
     {1, "x0"}
    },
    (*asse y*)
    {
     {ϵ, "ϵ"}
    }
   },
  TicksStyle → Directive[
    Hue[5 / 6], 12
   ],
  Exclusions →
   {
    x = 1
   },
```

```
  Epilog →
   {
    (*asintoto verticale*)
    {
     RGBColor[1, 0, 0], Dashed, Line[{{1, 0}, {1, -80}}]
    },
    (*retta orizzontale rϵ*)
    {
     RGBColor[1, 0, 0], Dashed, Line[{{0, -ϵ}, {2, -ϵ}}]
    },
    (*generico punto del grafico con x0-δϵ<x<x+δϵ*)
    {
     PointSize[0.018], Point[{1 - 0.85 δ[ϵ], f[1 - 0.85 δ[ϵ], 1]}]
    },
    (*linee verticali per delimitare l'intorno di x0*)
    {
     Dashed, Line[{{1 - δ[ϵ], 0}, {1 - δ[ϵ], -80}}]
    },
    {
     Dashed, Line[{{1 + δ[ϵ], 0}, {1 + δ[ϵ], -80}}]
    },
    (*freccia +∞*)
    {
     RGBColor[1, 0, 0], Arrow[{{0.95, ϵ - 10}, {0.95, ϵ + 20}}]
    },
    (*+∞*)
    Text[Style["+∞", Large, RGBColor[1, 0, 0]], {1, -80}]
   }
 ]

(*generazione dei fotogrammi*)

movie = Table[
   plot[ϵ],
   {ϵ, 10, 58}
  ];

(*l'output va esportato in formato gif, dopo aver settato la directory
 di destinazione con il comando SetDiretcory[]*)

(*ciclo Do per riprodurre in notebook l'animazione*)

Do[
 Print[
  plot[ϵ]
 ],
 {ϵ, 10, 12}
]
```

■ Teorema di unicità del limite

```
Clear[f, δ]

f[x_] := √x

δ[ϵ_] := 2 √2 ϵ - ϵ^2
```

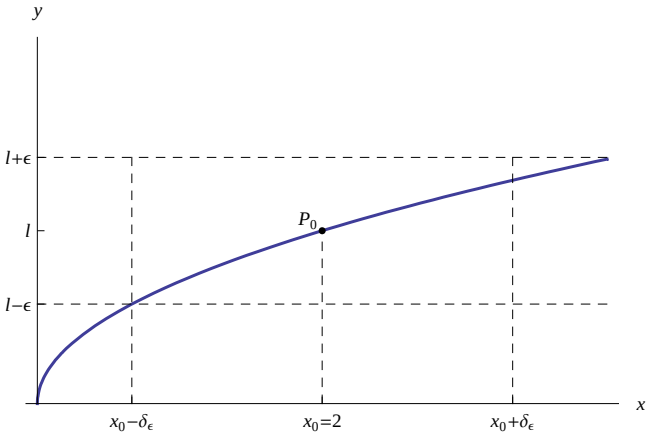
Sappiamo che $\lim_{x\to 2} f(x) = \sqrt{2}$. Infatti:

$$\delta(\epsilon) = 2\sqrt{2}\,\epsilon - \epsilon^2 \;=>\; \left(0 < |x-2| < \delta(\epsilon) \;=>\; \left|\sqrt{x} - \sqrt{2}\right| < \epsilon,\;\; \forall\, \epsilon \in \left(0, 2\sqrt{2}\right)\right)$$

Ciò è illustrato dal grafico:

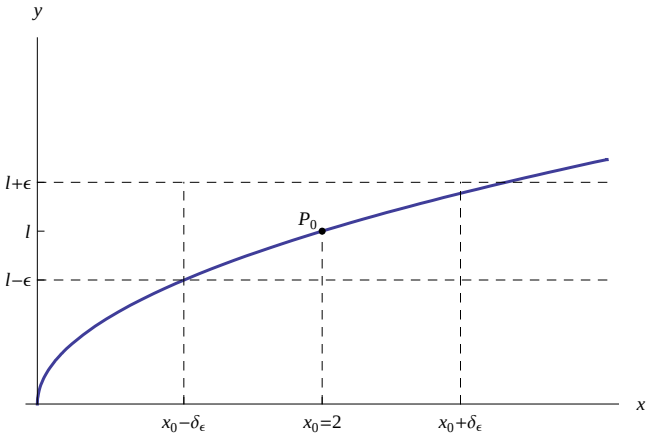
```
plot[ϵ_] := Plot[
  f[x],
  {x, 0, 4},
  PlotStyle → Thickness[0.005],
  AxesLabel →
   {
    "x", "y"
   },
  Ticks →
   {
    {
     {2 - δ[ϵ], "x0-δϵ"},
     {2 + δ[ϵ], "x0+δϵ"},
     {2, "x0=2"}
    },
    {
     {√2, "l"},
     {√2 + ϵ, "l+ϵ"},
     {√2 - ϵ, "l-ϵ"}
    }
   },
  PlotRange → {0, 3}
  ,
  Epilog →
   {
    {Dashed, Line[{{0, √2 + ϵ}, {4, √2 + ϵ}}]},
    {Dashed, Line[{{0, √2 - ϵ}, {4, √2 - ϵ}}]},
    {Dashed, Line[{{2, 0}, {2, √2}}]},
    {Dashed, Line[{{2 - δ[ϵ], 0}, {2 - δ[ϵ], √2 + ϵ}}]},
    {Dashed, Line[{{2 + δ[ϵ], 0}, {2 + δ[ϵ], √2 + ϵ}}]},
    Text["P0", {2 - 0.1, 0.1 + √2}],
    {
     PointSize[0.012], Point[{2, √2}]
    }
   }
 ]
```

```
plot[0.6]
```



Se proviamo a diminuire ϵ vediamo che per $0 < |x - x_0| < \delta_\epsilon$ i punti $(x, f(x))$ cadono comunque nella regione $\mathcal{R}_\epsilon = (x_0 - \delta_\epsilon, x_0 + \delta_\epsilon) \times (l - \epsilon, l + \epsilon)$

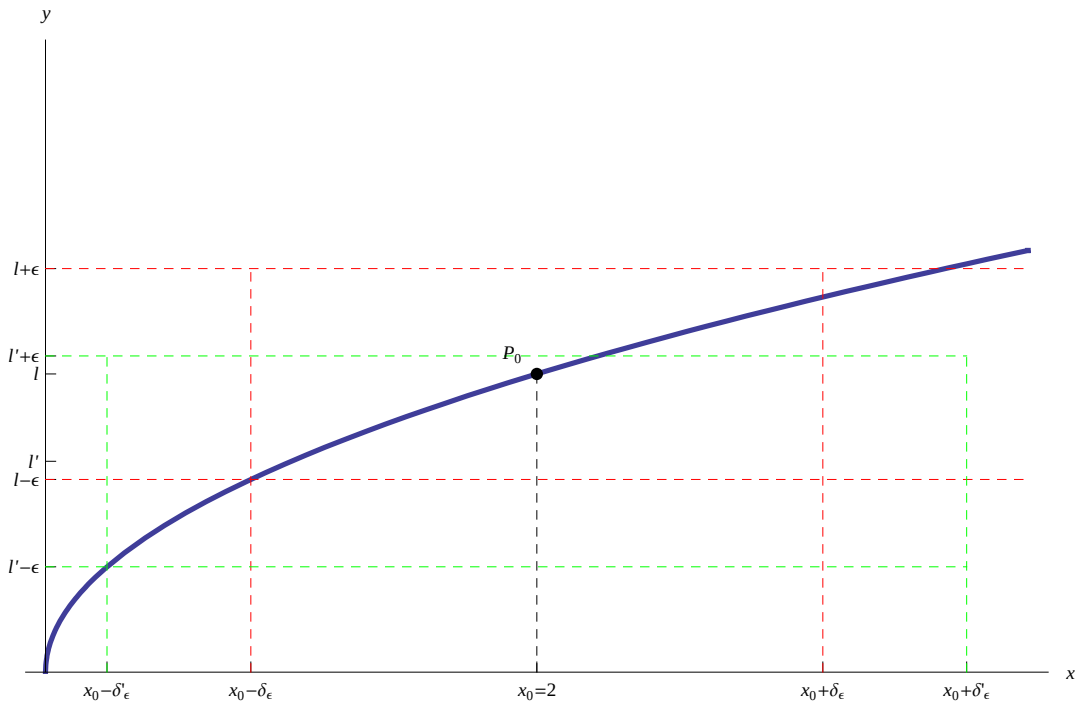
```
plot[0.4]
```



Supponiamo ora, per assurdo, che $f(x)$ tenda a due limiti distinti. Precisamente: $l = \sqrt{2}$ e $l' = 1$. Deve essere $\left|\sqrt{x} - 1\right| < \epsilon$, cioè $1 - 2\epsilon + \epsilon^2 < x < 1 + 2\epsilon + \epsilon^2$. Quindi poniamo $\delta'_\epsilon = 2\epsilon - \epsilon^2 + 1$ che risulta > 0 se $\epsilon \in \left(0, \frac{1+\sqrt{5}}{2}\right)$. Segue che $0 < \left|x - 2\right| < \delta'_\epsilon$ non implica $\left|\sqrt{x} - 1\right| < \delta'_\epsilon$. Verifichiamo ciò per via grafica:

```
δ1[ϵ_] := 2 ϵ - ϵ^2 + 1

plot1[ϵ_] := Plot[
  f[x],
  {x, 0, 4},
  PlotStyle → Thickness[0.005],
  ImageSize →
   {
    500,
    500
   },
  AxesLabel →
   {
    "x", "y"
   },
  Ticks →
```

```
    {
     {
      {2 - δ[ϵ], "x0-δϵ"},
      {2 + δ[ϵ], "x0+δϵ"},
      {2, "x0=2"},
      {2 - δ1[ϵ], "x0-δ'ϵ"},
      {2 + δ1[ϵ], "x0+δ'ϵ"}
     },
     {
      {√2, "l"},
      {1, "l'"},
      {√2 + ϵ, "l+ϵ"},
      {√2 - ϵ, "l-ϵ"},
      {1 + ϵ, "l'+ϵ"},
      {1 - ϵ, "l'-ϵ"}
     }
    },
   PlotRange → {0, 3},
   Epilog →
    {
     {Dashed, Line[{{2, 0}, {2, √2}}]},
     {Red, Dashed, Line[{{0, √2 + ϵ}, {4, √2 + ϵ}}]},
     {Red, Dashed, Line[{{0, √2 - ϵ}, {4, √2 - ϵ}}]},
     {Red, Dashed, Line[{{2 - δ[ϵ], 0}, {2 - δ[ϵ], √2 + ϵ}}]},
     {Red, Dashed, Line[{{2 + δ[ϵ], 0}, {2 + δ[ϵ], √2 + ϵ}}]},
     (*linea orizzontale da l'-ϵ*)
     {Green, Dashed, Line[{{0, 1 - ϵ}, {2 + δ1[ϵ], 1 - ϵ}}]},
     (*linea orizzontale da l'+ϵ*)
     {Green, Dashed, Line[{{0, 1 + ϵ}, {2 + δ1[ϵ], 1 + ϵ}}]},
     (*linee verticali*)
     {Green, Dashed, Line[{{2 - δ1[ϵ], 0}, {2 - δ1[ϵ], 1 + ϵ}}]},
     {Green, Dashed, Line[{{2 + δ1[ϵ], 0}, {2 + δ1[ϵ], 1 + ϵ}}]},
     Text["P0", {2 - 0.1, 0.1 + √2}],
     {
      PointSize[0.012], Point[{2, √2}]
     }
    }
  ]

plot1[0.5]
```



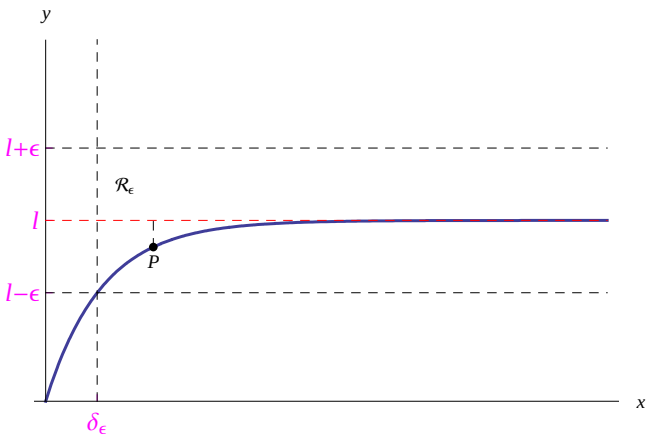
Da tale grafico vediamo che per $l' = 1$ la definizione di limite viene violata. Creiamo ora una lista di fotogrammi da esportare in formato gif, dopo aver indicato la directory di destinazione.

```
movie = Table[
   plot1[ϵ],
   {ϵ, 1, 0.1, -0.01}
  ];
```

Funzioni convergenti all'infinito

```
Clear[f, δ, plot, movie]

f[x_] := 1 - e^-x

δ[ϵ_] := -Log[ϵ]

plot[ϵ_] := Plot[
  f[x],
  {x, 0, 10},
  PlotRange →
   {
    0, 2
   },
  AxesLabel →
```

```
    {
     "x", "y"
    },
   PlotStyle → Thickness[0.005],
   Ticks →
    {
     (*asse x*)
     {
      {δ[ϵ], "δϵ"}
     },
     (*asse y*)
     {
      {1 - ϵ, "l-ϵ"},
      {1, "l"},
      {1 + ϵ, "l+ϵ"}
     }
    },
   TicksStyle → Directive[
     Hue[5 / 6], 12
    ],
   Epilog →
    {
     (*asintoto orizzontale*)
     {
      RGBColor[1, 0, 0], Dashed, Line[{{0, 1}, {10, 1}}]
     },
     (*intorno di l=1*)
     (*prima linea orizzontale*)
     {
      Dashed, Line[{{0, 1 - ϵ}, {10, 1 - ϵ}}]
     },
     (*seconda linea orizzontale*)
     {
      Dashed, Line[{{0, 1 + ϵ}, {10, 1 + ϵ}}]
     },
     (*intorno di +∞*)
     {
      Dashed, Line[{{δ[ϵ], 0}, {δ[ϵ], 2}}]
     },
     (*regione ℛϵ*)
     {
      Text["ℛϵ", {δ[ϵ] + 0.5, 1.2}]
     },
     (*generico punto del grafico con
       x>δϵ*)
     {
      PointSize[0.015], Point[{δ[ϵ] + 1, f[δ[ϵ] + 1]}],
      Text["P", {δ[ϵ] + 1.0, f[δ[ϵ] + 1] - 0.08}]
     },
     (*distanza dall'asintoto*)
     {
      Dashed, Line[{{δ[ϵ] + 1, f[δ[ϵ] + 1]}, {δ[ϵ] + 1, 1}}]
     }
    }
  ]

plot[0.4]
```

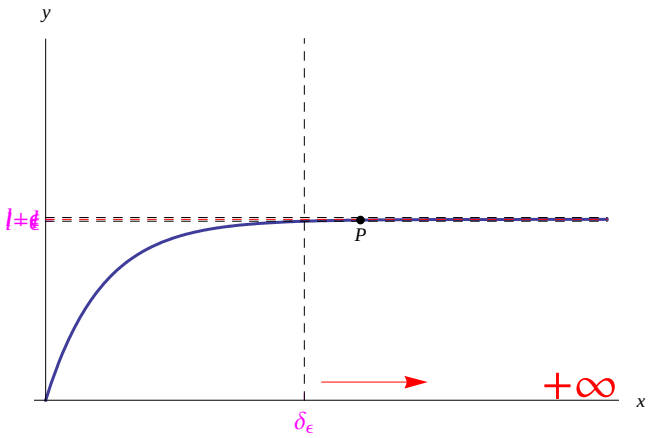



```
SetDirectory["D:\\Siti\\extrabyte2\\Mathematica\\ANALISI1"];

Export["converg_infinity.eps", plot[0.4]]
```

converg_infinity.eps

```
(*replot per l'animazione grafica*)

Clear[plot]

plot[ϵ_] := Plot[
  f[x],
  {x, 0, 10},
  PlotRange →
   {
    0, 2
   },
  AxesLabel →
   {
    "x", "y"
   },
  PlotStyle → Thickness[0.005],
  Ticks →
   {
    (*asse x*)
    {
     {δ[ϵ], "δϵ"}
    },
    (*asse y*)
    {
     {1 - ϵ, "l-ϵ"},
     {1, "l"},
     {1 + ϵ, "l+ϵ"}
    }
   },
  TicksStyle → Directive[
    Hue[5 / 6], 12
   ],
  Epilog →
   {
    (*asintoto orizzontale*)
    {
     RGBColor[1, 0, 0], Dashed, Line[{{0, 1}, {10, 1}}]
    },
    (*intorno di l=1*)
```

```
    (*prima linea orizzontale*)
    {
     Dashed, Line[{{0, 1 - ϵ}, {10, 1 - ϵ}}]
    },
    (*seconda linea orizzontale*)
    {
     Dashed, Line[{{0, 1 + ϵ}, {10, 1 + ϵ}}]
    },
    (*intorno di +∞*)
    {
     Dashed, Line[{{δ[ϵ], 0}, {δ[ϵ], 2}}]
    },
    (*generico punto del grafico con
      x>δϵ*)
    {
     PointSize[0.015], Point[{δ[ϵ] + 1, f[δ[ϵ] + 1]}],
     Text["P", {δ[ϵ] + 1.0, f[δ[ϵ] + 1] - 0.08}]
    },
    (*distanza dall'asintoto*)
    {
     Dashed, Line[{{δ[ϵ] + 1, f[δ[ϵ] + 1]}, {δ[ϵ] + 1, 1}}]
    },
    (*freccia che tende a +∞*)
    {
     RGBColor[1, 0, 0], Arrow[{{δ[ϵ] + 0.3, 0.1}, {δ[ϵ] + 2.2, 0.1}}]
    },
    (*+∞*)
    Text[Style["+∞", Large, RGBColor[1, 0, 0]], {9.5, 0.1}]
   }
 ]

plot[0.01]
```




```
(*generazione dei fotogrammi*)

movie = Table[
   plot[ϵ],
   {ϵ, 0.6, 0.002, -0.01}
  ];

SetDirectory[
  "D:\\Siti\\extrabyte2\\Mathematica\\ANALISI1\\MATHEMATICA\\LIMITI\\asintoto_orizz"];
```

```
Export["asintoto_orizz.gif", movie]
```

asintoto_orizz.gif

```
(*l'output va esportato in formato gif, dopo aver settato la directory
 di destinazione con il comando SetDiretcory[]*)
```

```
(*ciclo Do per riprodurre in notebook l'animazione*)
```

```
Do[
 Print[
  plot[ϵ]
 ],
 {ϵ, 10, 12}
]
```

Limiti di funzioni irrazionali. Fattore razionalizzante

Sia data la funzione $f(x) = \sqrt[n]{p(x)} \pm \sqrt[n]{q(x)}$, dove $p(x)$ e $q(x)$ sono polinomi.

$$\texttt{f1[x\_, n\_, p\_, q\_] :=} \sqrt[n]{p} + \sqrt[n]{q}$$

Il fattore razionalizzante è

$$\texttt{R1[x\_, n\_, p\_, q\_] := Sum}\left[(-1)^{k+1} * \sqrt[n]{p^{n-k} * q^{k-1}},\ \{k, n\}\right]$$

$$\texttt{f2[x\_, n\_, p\_, q\_] :=} \sqrt[n]{p} - \sqrt[n]{q}$$

Il fattore razionalizzante è

$$\texttt{R2[x\_, n\_, p\_, q\_] := Sum}\left[\sqrt[n]{p^{n-k} * q^{k-1}},\ \{k, n\}\right]$$

Esempio: il fattore razionalizzante di $f(x) = \sqrt{x-1} - \sqrt{2x}$ è

```
R2[x, 2, x - 1, 2 x]
```

$$\sqrt{-1+x} + \sqrt{2}\,\sqrt{x}$$

```
R2[x, 3, x - 1, 2 x]
```

$$\left((-1+x)^2\right)^{1/3} + 2^{1/3}\left((-1+x)\,x\right)^{1/3} + 2^{2/3}\left(x^2\right)^{1/3}$$

```
Table[
  R2[x, n, x - 1, 2 x],
  {n, 2, 10}
 ] // TableForm
```

$$\sqrt{-1+x}+\sqrt{2}\,\sqrt{x}$$

$$\left((-1+x)^2\right)^{1/3}+2^{1/3}\left((-1+x)\,x\right)^{1/3}+2^{2/3}\left(x^2\right)^{1/3}$$

$$\left((-1+x)^3\right)^{1/4}+2^{1/4}\left((-1+x)^2\,x\right)^{1/4}+\sqrt{2}\left((-1+x)\,x^2\right)^{1/4}+2^{3/4}\left(x^3\right)^{1/4}$$

$$\left((-1+x)^4\right)^{1/5}+2^{1/5}\left((-1+x)^3\,x\right)^{1/5}+2^{2/5}\left((-1+x)^2\,x^2\right)^{1/5}+2^{3/5}\left((-1+x)\,x^3\right)^{1/5}+2^{4/5}\left(x^4\right)^{1/5}$$

$$\left((-1+x)^5\right)^{1/6}+2^{1/6}\left((-1+x)^4\,x\right)^{1/6}+2^{1/3}\left((-1+x)^3\,x^2\right)^{1/6}+\sqrt{2}\left((-1+x)^2\,x^3\right)^{1/6}+2^{2/3}\left((-1+x)\,x^4\right)^{1/6}+$$

$$\left((-1+x)^6\right)^{1/7}+2^{1/7}\left((-1+x)^5\,x\right)^{1/7}+2^{2/7}\left((-1+x)^4\,x^2\right)^{1/7}+2^{3/7}\left((-1+x)^3\,x^3\right)^{1/7}+2^{4/7}\left((-1+x)^2\,x^4\right)^{1/7}$$

$$\left((-1+x)^7\right)^{1/8}+2^{1/8}\left((-1+x)^6\,x\right)^{1/8}+2^{1/4}\left((-1+x)^5\,x^2\right)^{1/8}+2^{3/8}\left((-1+x)^4\,x^3\right)^{1/8}+\sqrt{2}\left((-1+x)^3\,x^4\right)^{1/8}$$

$$\left((-1+x)^8\right)^{1/9}+2^{1/9}\left((-1+x)^7\,x\right)^{1/9}+2^{2/9}\left((-1+x)^6\,x^2\right)^{1/9}+2^{1/3}\left((-1+x)^5\,x^3\right)^{1/9}+2^{4/9}\left((-1+x)^4\,x^4\right)^{1/9}$$

$$\left((-1+x)^9\right)^{1/10}+2^{1/10}\left((-1+x)^8\,x\right)^{1/10}+2^{1/5}\left((-1+x)^7\,x^2\right)^{1/10}+2^{3/10}\left((-1+x)^6\,x^3\right)^{1/10}+2^{2/5}\left((-1+x)^5\,x\right.$$

```
R2[x, 10, x - 1, 2 x]
```

$$\left((-1+x)^9\right)^{1/10}+2^{1/10}\left((-1+x)^8\,x\right)^{1/10}+2^{1/5}\left((-1+x)^7\,x^2\right)^{1/10}+$$
$$2^{3/10}\left((-1+x)^6\,x^3\right)^{1/10}+2^{2/5}\left((-1+x)^5\,x^4\right)^{1/10}+\sqrt{2}\left((-1+x)^4\,x^5\right)^{1/10}+$$
$$2^{3/5}\left((-1+x)^3\,x^6\right)^{1/10}+2^{7/10}\left((-1+x)^2\,x^7\right)^{1/10}+2^{4/5}\left((-1+x)\,x^8\right)^{1/10}+2^{9/10}\left(x^9\right)^{1/10}$$

```
f[x_] = f2[x, 4, x^4 + 1, x^4]
```

$$-\left(x^4\right)^{1/4}+\left(1+x^4\right)^{1/4}$$

```
R2[x, 4, x^4 + 1, x^4] /. {p → x^4 + 1, q → x^4}
```

$$\left(x^{12}\right)^{1/4}+\left(x^8\left(1+x^4\right)\right)^{1/4}+\left(x^4\left(1+x^4\right)^2\right)^{1/4}+\left(\left(1+x^4\right)^3\right)^{1/4}$$

```
Limit[
 f[x],
 x → -∞
]
```

0

```
Clear[f]
```

```
f[x_] = f2[x, 12, x^12 + 1, x^12]
```

$$-\left(x^{12}\right)^{1/12}+\left(1+x^{12}\right)^{1/12}$$

```
R[x_] = R2[x, 12, x^12 + 1, x^12]
```

$$\left(x^{132}\right)^{1/12}+\left(x^{120}\left(1+x^{12}\right)\right)^{1/12}+\left(x^{108}\left(1+x^{12}\right)^2\right)^{1/12}+\left(x^{96}\left(1+x^{12}\right)^3\right)^{1/12}+$$
$$\left(x^{84}\left(1+x^{12}\right)^4\right)^{1/12}+\left(x^{72}\left(1+x^{12}\right)^5\right)^{1/12}+\left(x^{60}\left(1+x^{12}\right)^6\right)^{1/12}+\left(x^{48}\left(1+x^{12}\right)^7\right)^{1/12}+$$
$$\left(x^{36}\left(1+x^{12}\right)^8\right)^{1/12}+\left(x^{24}\left(1+x^{12}\right)^9\right)^{1/12}+\left(x^{12}\left(1+x^{12}\right)^{10}\right)^{1/12}+\left(\left(1+x^{12}\right)^{11}\right)^{1/12}$$

```
Clear[ff]
```

$$\mathbf{ff[x\_] := \frac{f[x] * R[x]}{R[x]}}$$

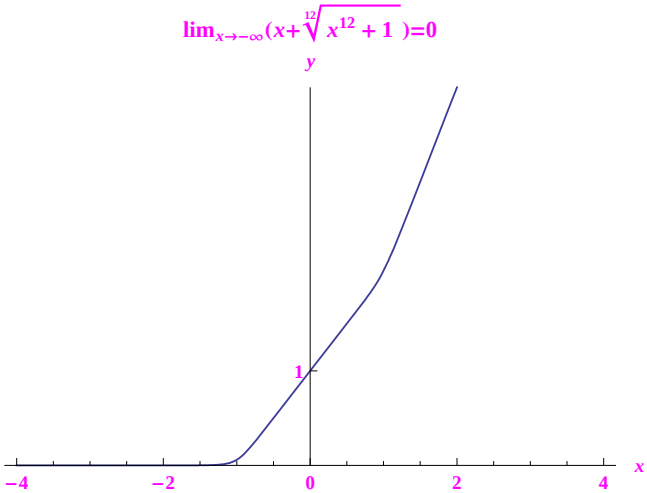
```
Limit[
 ff[x],
 x → -∞
]
```

0

```
Limit[
 f[x],
 x → -∞
]
```

0

```
plot = Plot[x + (1 + x^12)^(1/12),
  {x, -4, 4},
  PlotRange → {0, 4},
  PlotStyle → Thickness[0.003],
  AxesLabel →
   {"x", "y"},
  PlotLabel → "lim_{x→-∞} (x+∛(x^12 + 1))=0", LabelStyle → Directive[Bold, Hue[5 / 6]],
  Ticks →
   {
    Automatic,
    {1}
   }
 ]
```




```
Clear[g]
```

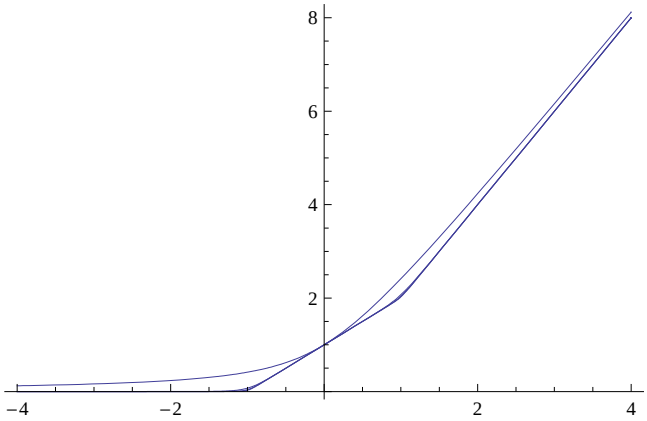
$g[x\_, n\_] := x + \sqrt[2n]{x^{2n} + 1}$

```
lim[n_] := Limit[g[x, n], x → -∞]
```

```
Table[lim[n], {n, 1, 12}]
```

{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}

```
plotg = Plot[
  Table[
   g[x, n],
   {n, 1, 14, 4}
  ],
  {x, -4, 4}
 ]
```



Grafica tridimensionale

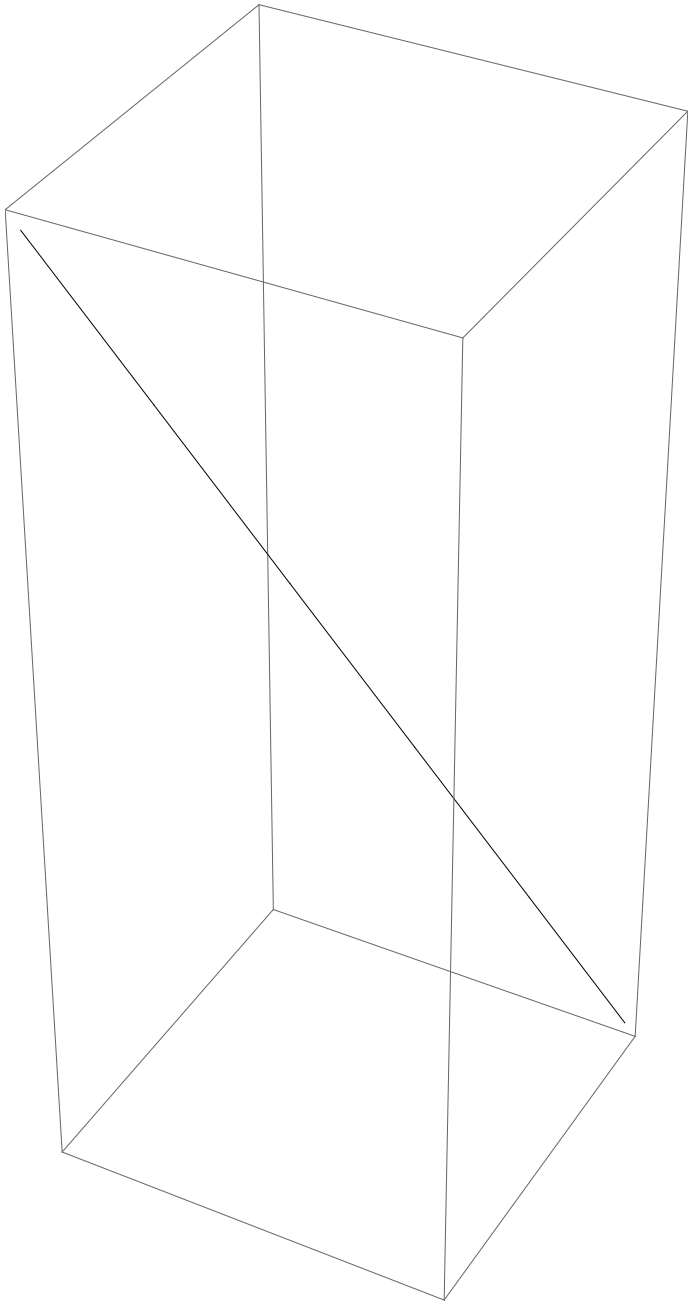
■ Principali direttive grafiche

```
Clear[segmento]
```

```
segmento = Line[{{2, 1, -1}, {0, -1, 4}}]
```

```
Line[{{2, 1, -1}, {0, -1, 4}}]
```

```
Graphics3D[segmento]
```



```
Graphics3D // Options
```

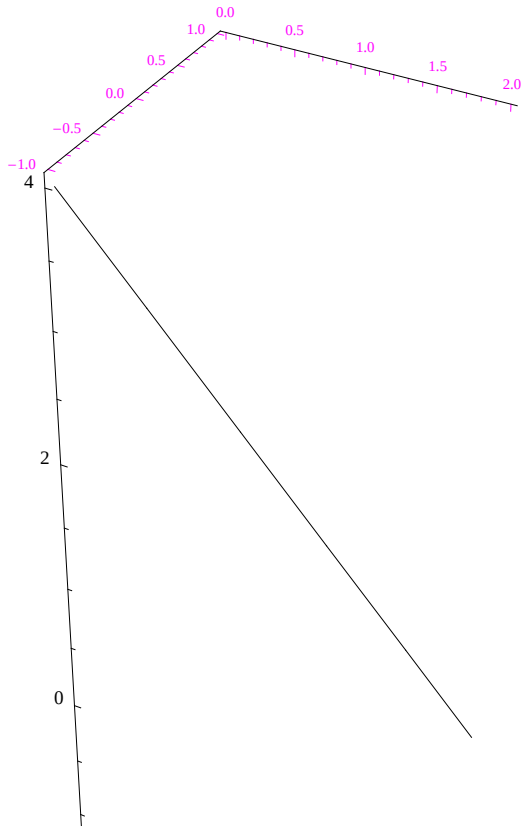
```
{AlignmentPoint → Center, AspectRatio → Automatic, Axes → False,
 AxesEdge → Automatic, AxesLabel → None, AxesStyle → {}, Background → None,
 BaselinePosition → Automatic, BaseStyle → {}, Boxed → True, BoxRatios → Automatic,
 BoxStyle → {}, ColorOutput → Automatic, ContentSelectable → Automatic,
 ControllerLinking → Automatic, ControllerMethod → Automatic, ControllerPath → Automatic,
 DisplayFunction :→ $DisplayFunction, Epilog → {}, FaceGrids → None, FaceGridsStyle → {},
 FormatType :→ TraditionalForm, ImageMargins → 0., ImagePadding → All,
 ImageSize → Automatic, LabelStyle → {}, Lighting → Automatic, Method → Automatic,
 PlotLabel → None, PlotRange → All, PlotRangePadding → Automatic, PlotRegion → Automatic,
 PreserveImageOptions → Automatic, Prolog → {}, RotationAction → Fit,
 SphericalRegion → False, Ticks → Automatic, TicksStyle → {}, ViewAngle → Automatic,
 ViewCenter → {1/2, 1/2, 1/2}, ViewMatrix → Automatic, ViewPoint → {1.3, -2.4, 2.},
 ViewRange → All, ViewVector → Automatic, ViewVertical → {0, 0, 1}}
```

```
? Graphics3D
```

Graphics3D[*primitives*, *options*] represents a three-dimensional graphical image. ≫

```
SetOptions[
  Graphics3D,
  TicksStyle → Directive[Hue[5 / 6], 7]
 ];
```

```
Graphics3D[
 segmento,
 Boxed → False,
 Axes → True
]
```

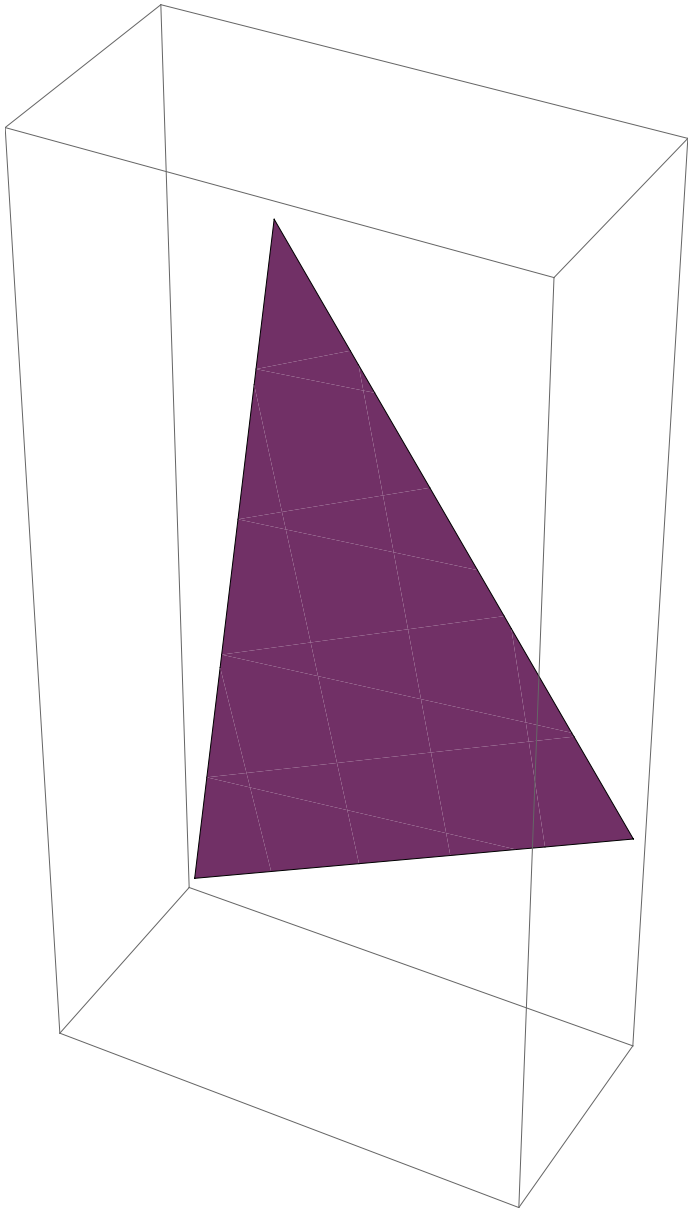


L'istruzione **Polygon** vista nel caso 2D, continua a valere in 3D:

```
triangolo = {{0, 2, 1}, {1, 1, 5}, {2, 2, 2}}
```

```
{{0, 2, 1}, {1, 1, 5}, {2, 2, 2}}
```

```
Polygon[triangolo] // Graphics3D
```



L'esempio dei triangoli random in 2D può essere generalizzato a 3D. Generiamo una lista di coppie ordinate di numeri reali appartenenti all'intervallo [0, 1]

```
Clear[triangolo, listatriangoli, listatriangoli1, listatriangoli2, listatriangoli3, lista]
```

```
RandomReal[1, {3, 3}]
```

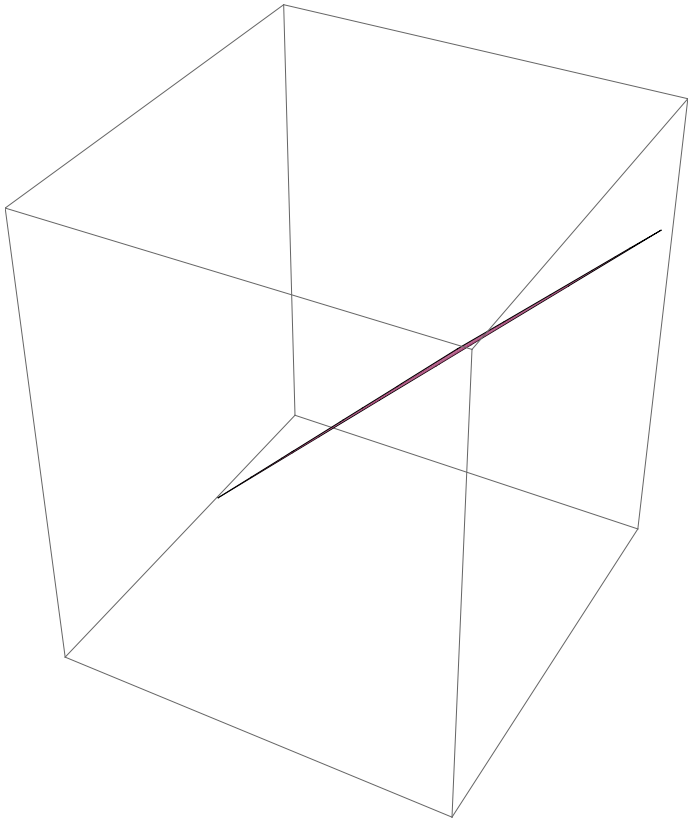
```
{{0.105585, 0.0942737, 0.0660532},
 {0.568096, 0.113087, 0.660577}, {0.480397, 0.272869, 0.397161}}
```

Le predette terne ordinate possono essere le coordinate cartesiane dei vertici di un triangolo

```
triangolo = Polygon[RandomReal[1, {3, 3}]]
```

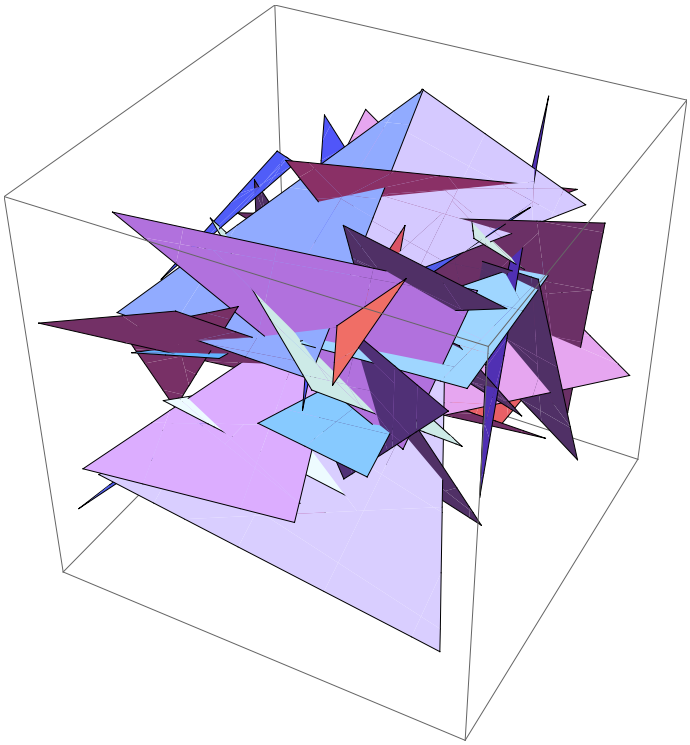
```
Polygon[{{0.094928, 0.439559, 0.156518},
  {0.509198, 0.613276, 0.527427}, {0.501815, 0.173791, 0.655622}}]
```

```
triangolo // Graphics3D
```



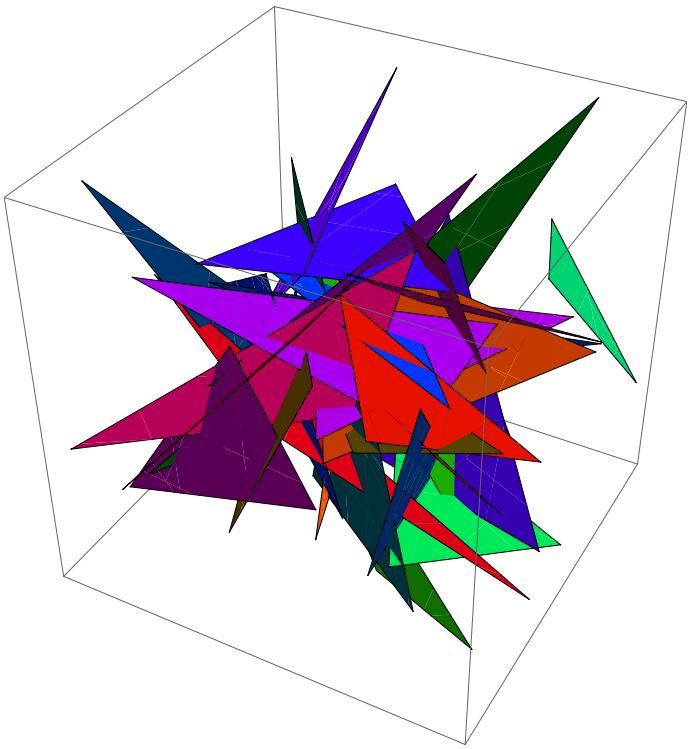
Costruiamo una lista di 30 triangoli random:

```
listatriangoli = Table[
   (*argomento principale*)
   Polygon[RandomReal[1, {3, 3}]],
   (*iterazione*)
   {30}
  ] // Graphics3D
```



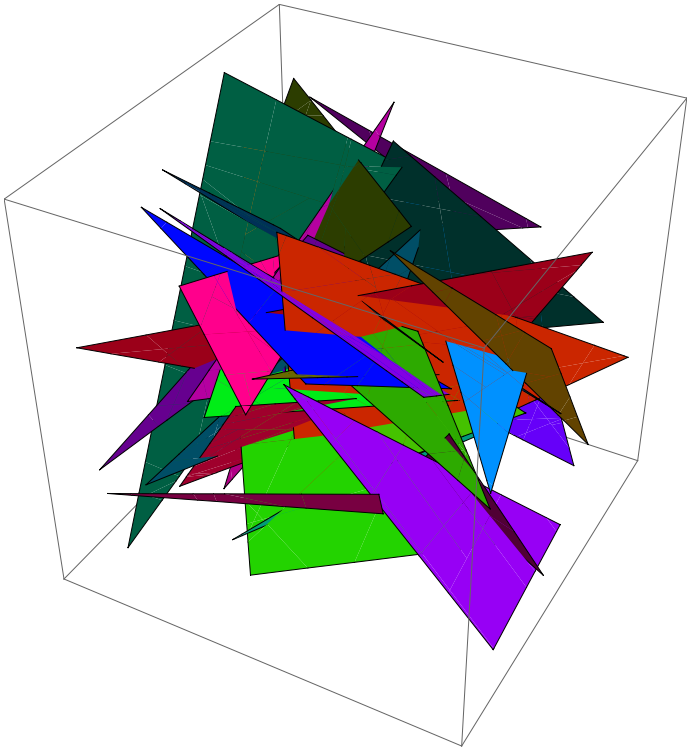
Coloriamo i vari triangoli in modalità random:

```
listatriangoli1 = Table[
   (*argomento principale*)
   {
    Hue[RandomReal[]],
    Polygon[RandomReal[1, {3, 3}]]
   },
   (*iterazione*)
   {30}
  ] // Graphics3D
```



Aggiungiamo un contorno black

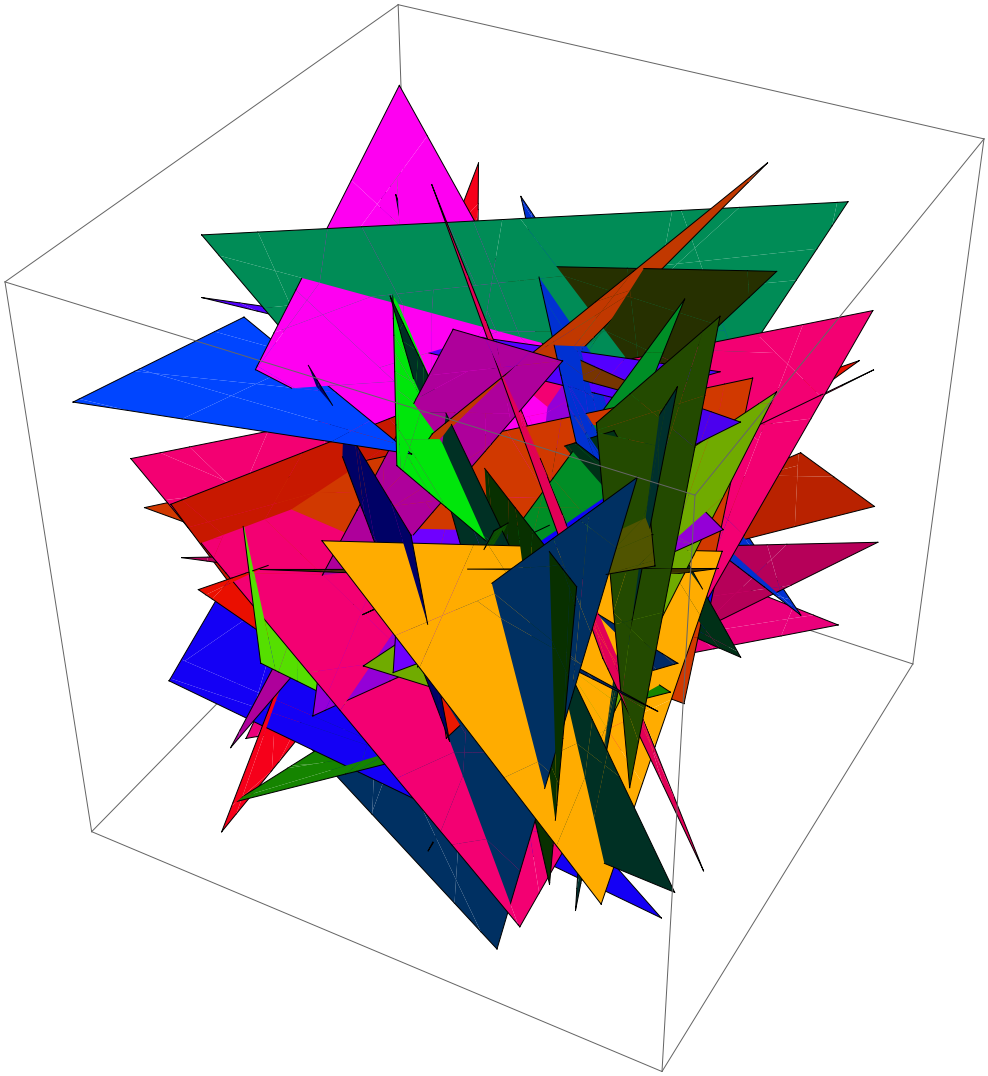
```
listatriangoli2 = Table[
   (*argomento principale*)
   {
    EdgeForm[Black],
    Hue[RandomReal[]],
    Polygon[RandomReal[1, {3, 3}]]
   },
   (*iterazione*)
   {30}
  ] // Graphics3D
```



Generiamo un'altra lista per poi esportarla come gif animata.

```
listatriangoli3[a_] := Graphics3D[
  Table[
   (*argomento principale*)
   {
    EdgeForm[Black],
    Hue[RandomReal[]],
    Polygon[RandomReal[1, {3, 3}]]
   },
   (*iterazione*)
   {n, a}
  ],
  ImageSize → {500, 500}
 ]
```

```
listatriangoli3[50]
```



```
lista = Table[
   listatriangoli3[a],
   {a, 1, 50}
  ];
```

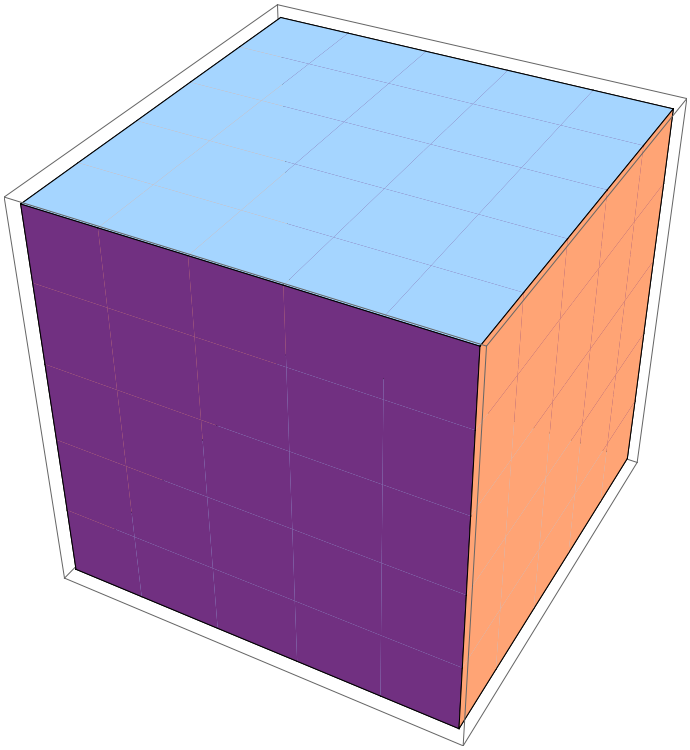
Per disegnare un cubo:

```
? Cuboid
```

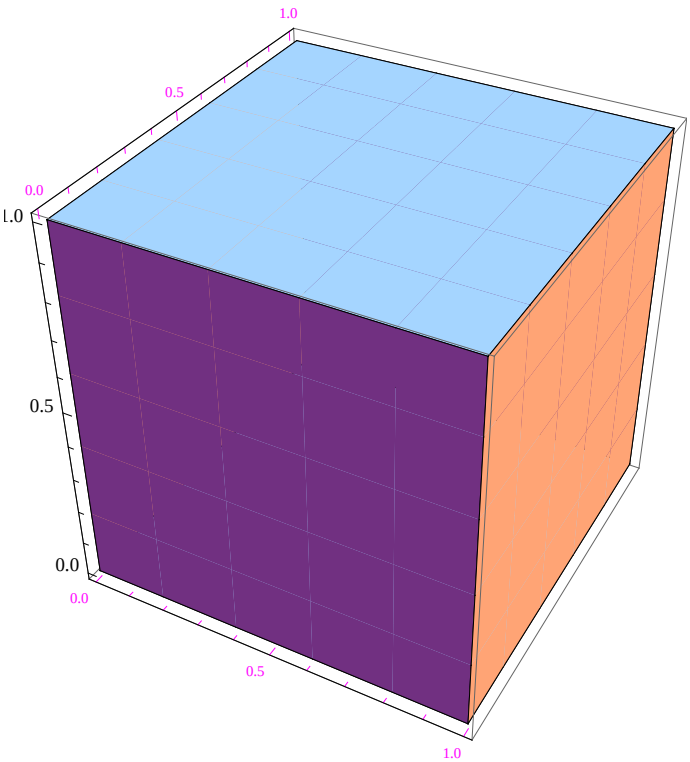
Cuboid[{x_{min}, y_{min}, z_{min}}] is a three-dimensional
graphics primitive that represents a unit cuboid, oriented parallel to the axes.
Cuboid[{x_{min}, y_{min}, z_{min}}, {x_{max}, y_{max}, z_{max}}] specifies a cuboid by giving the coordinates of opposite corners. ≫

Un cubo di spigolo unitario

```
Cuboid[] // Graphics3D
```

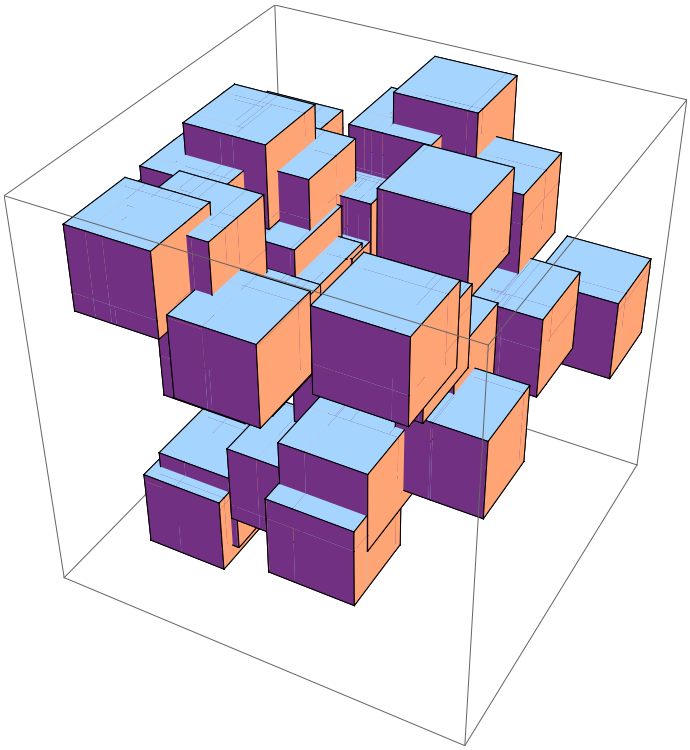


```
Graphics3D[
 Cuboid[],
 Axes → True
]
```



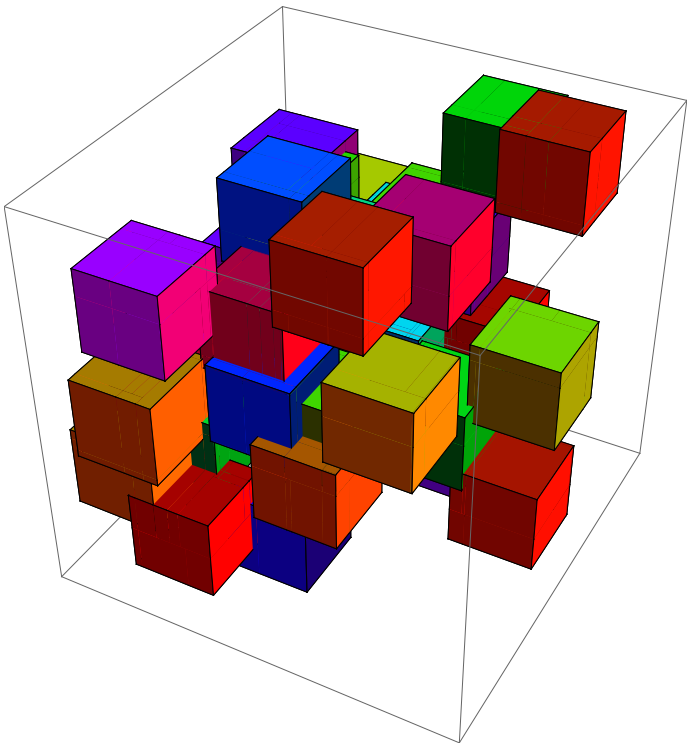
Come nel caso dei triangoli, possiamo generare cubi random. Costruiamo una lista di 30 triangoli random:

```
listacubi = Table[
   (*argomento principale*)
   Cuboid[RandomReal[4, 3]],
   (*iterazione*)
   {40}
  ] // Graphics3D
```



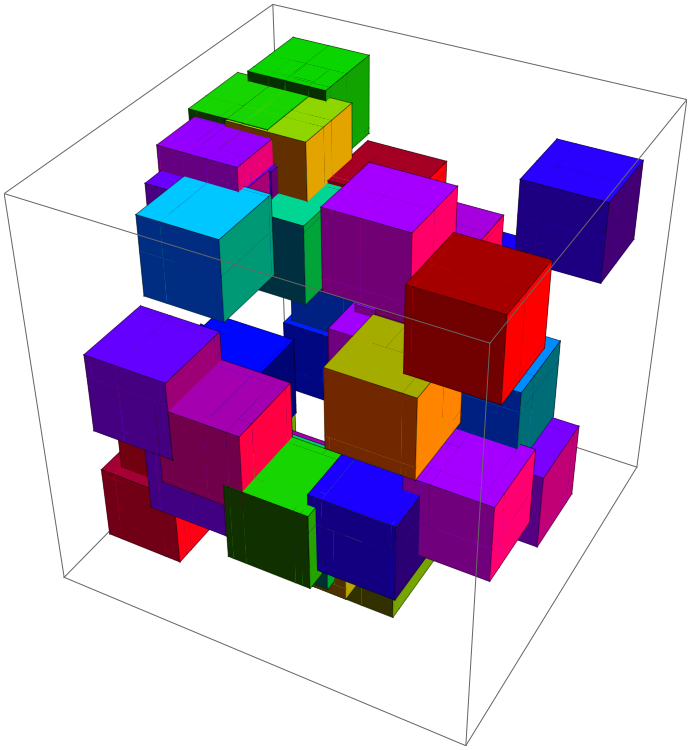
Coloriamo i vari cubi in modalità random:

```
listacubi1 = Table[
   (*argomento principale*)
   {
    Hue[RandomReal[]],
    Cuboid[RandomReal[4, 3]]
   },
   (*iterazione*)
   {40}
  ] // Graphics3D
```



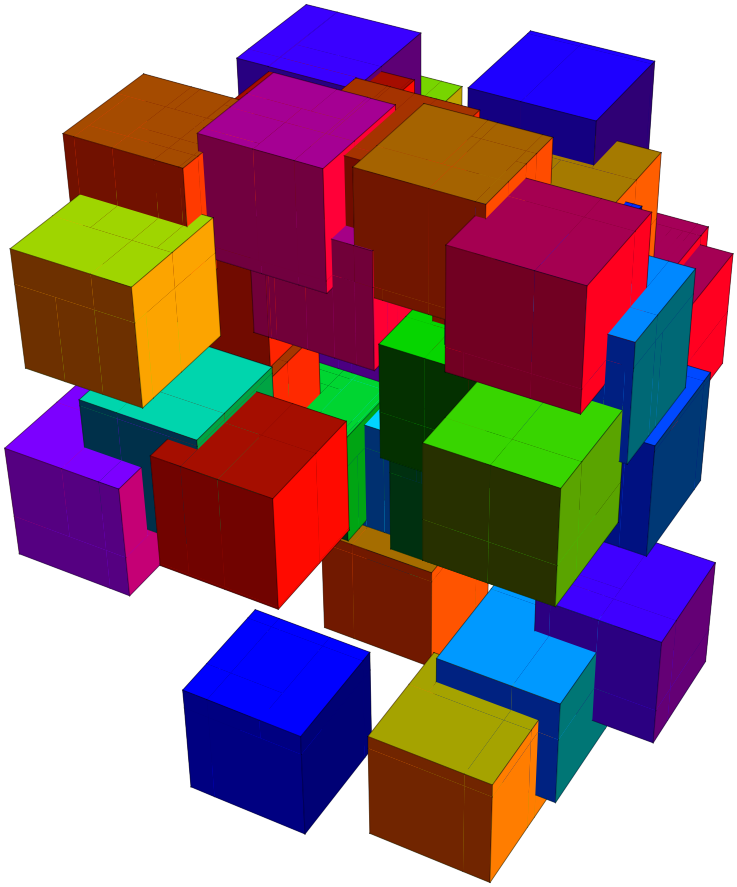
Aggiungiamo un contorno

```
listacubi2 = Table[
   (*argomento principale*)
   {
    EdgeForm[Opacity[.3]],
    Hue[RandomReal[]],
    Cuboid[RandomReal[4, 3]]
   },
   (*iterazione*)
   {40}
  ] // Graphics3D
```



```
listacubi3[a_] := Graphics3D[
  Table[
   (*argomento principale*)
   {
    EdgeForm[Opacity[.3]],
    Hue[RandomReal[]],
    Cuboid[RandomReal[4, 3]]
   },
   (*iterazione*)
   {n, a}
  ],
  ImageSize → {500, 500},
  Boxed → False
 ]
```

```
listacubi3[50]
```



```
lista = Table[
   listacubi3[a],
   {a, 1, 50}
  ];
```

■ L'istruzione Plot3D

Per disegnare il grafico di una funzione di due variabili reali, i.e. la superficie di equazione $z = f(x, y)$ utilizziamo l'istruzione **Plot3D** che è l'equivalente tridimensionale dell'istruzione **Plot**.

```
? Plot3D
```

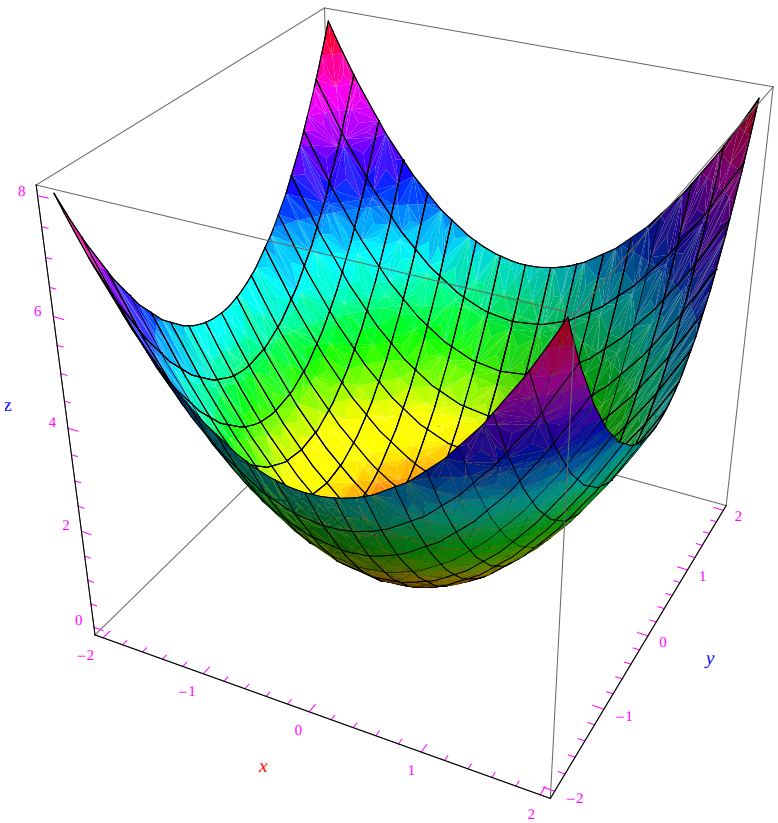
Plot3D[f, {x, x_{min}, x_{max}}, {y, y_{min}, y_{max}}] generates a three-dimensional plot of f as a function of x and y.
Plot3D[{f_1, f_2, …}, {x, x_{min}, x_{max}}, {y, y_{min}, y_{max}}] plots several functions. ≫

```
SetOptions[
  {
   Plot3D,
   ContourPlot
  },
  TicksStyle → Directive[Magenta, 7]
 ];
```

```
Clear[f]
```

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

```
plotf3d = Plot3D[
  f[x, y],
  {x, -2, 2}, {y, -2, 2},
  BoxRatios → {1, 1, 1},
  ColorFunction → Function[{x, y, z}, Hue[z]],
  AxesLabel →
   {
    Style["x", Small, Red],
    Style["y", Small, Blue],
    Style["z", Small, Blue]
   }
 ]
```

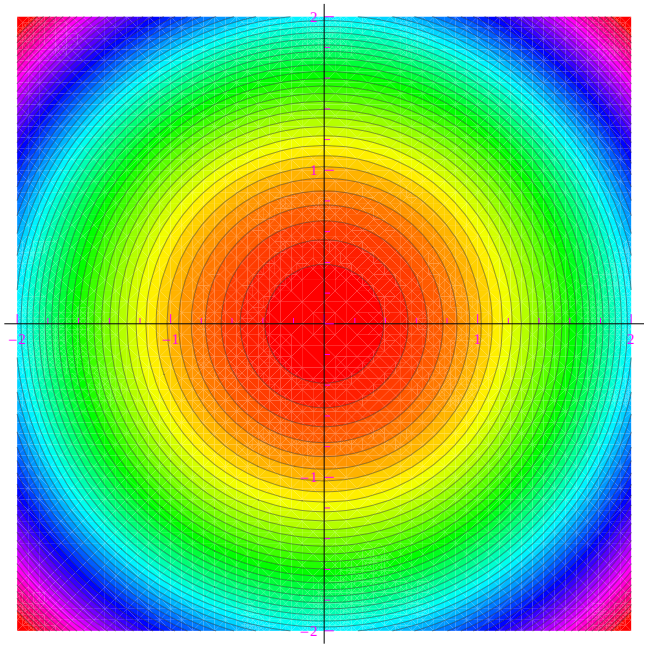



```
SetDirectory["G:\\Siti\\extrabyte2\\Mathematica\\ANALISI2\\data"];
```

```
Export["nls.eps", plotf3d]
```

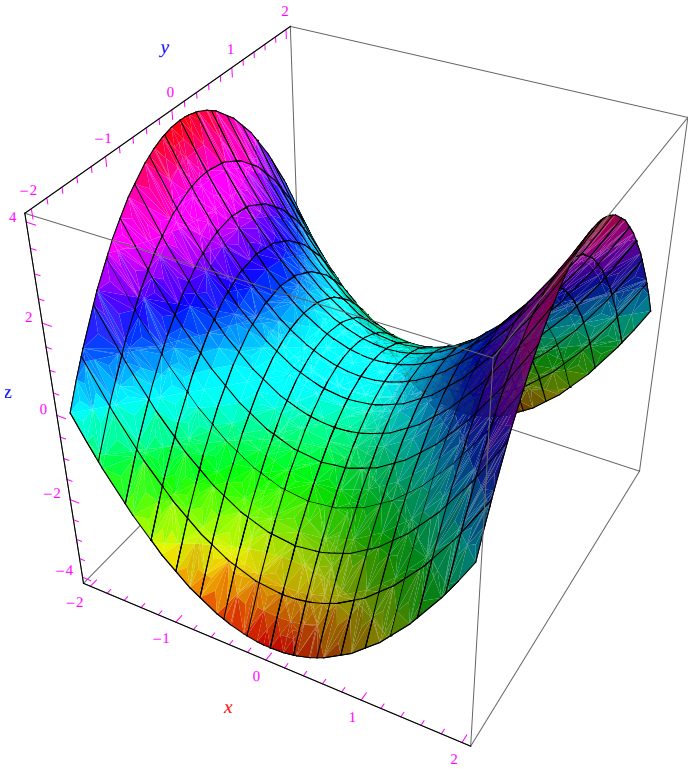
```
nls.eps
```

```
livellof1 = ContourPlot[
  f[x, y],
  {x, -2, 2}, {y, -2, 2},
  Axes → True,
  Frame → False,
  ColorFunction → Hue,
  Contours → 50
 ]
```




```
Export["livellof1.eps", livellof1];

Clear[f, plotf3d, livellof1]

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

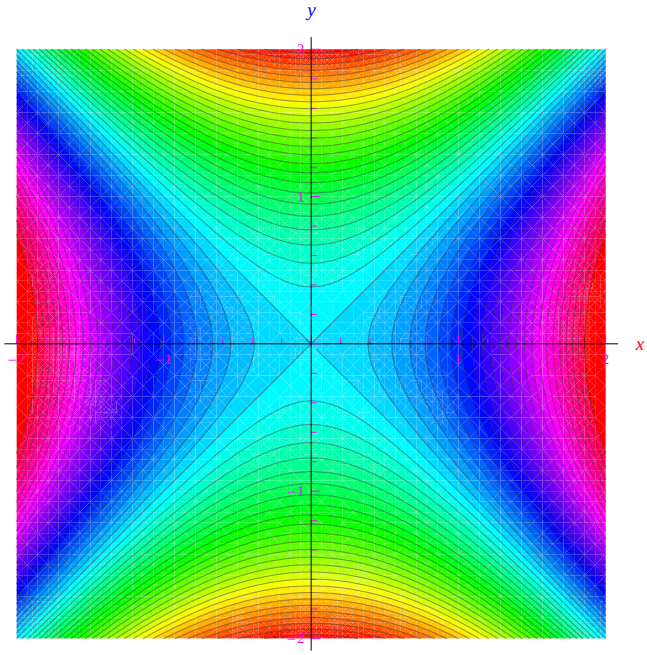
```
plotf3d = Plot3D[
  f[x, y],
  {x, -2, 2}, {y, -2, 2},
  BoxRatios → {1, 1, 1},
  ColorFunction → Function[{x, y, z}, Hue[z]],
  AxesLabel →
   {
    Style["x", Small, Red],
    Style["y", Small, Blue],
    Style["z", Small, Blue]
   }
 ]
```




```
Export["iperparab.eps", plotf3d]
```

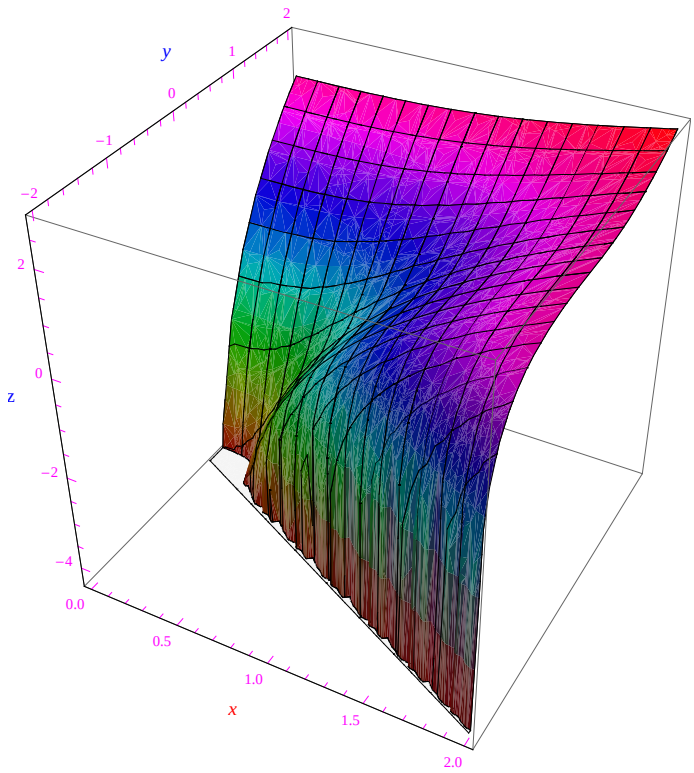
iperparab.eps

```
livellof2 = ContourPlot[
  f[x, y],
  {x, -2, 2}, {y, -2, 2},
  Axes → True,
  Frame → False,
  ColorFunction → Hue,
  Contours → 50,
  AxesLabel →
   {
    Style["x", Small, Red],
    Style["y", Small, Blue]
   }
 ]
```




```
Clear[f, plotf3d, livellof1]
```

```
plotf3d = Plot3D[
  Log[x^3 + y^3],
  {x, 0, 2}, {y, -2, 2},
  BoxRatios → {1, 1, 1},
  ColorFunction → Function[{x, y, z}, Hue[z]],
  AxesLabel →
   {
    Style["x", Small, Red],
    Style["y", Small, Blue],
    Style["z", Small, Blue]
   }
 ]
```



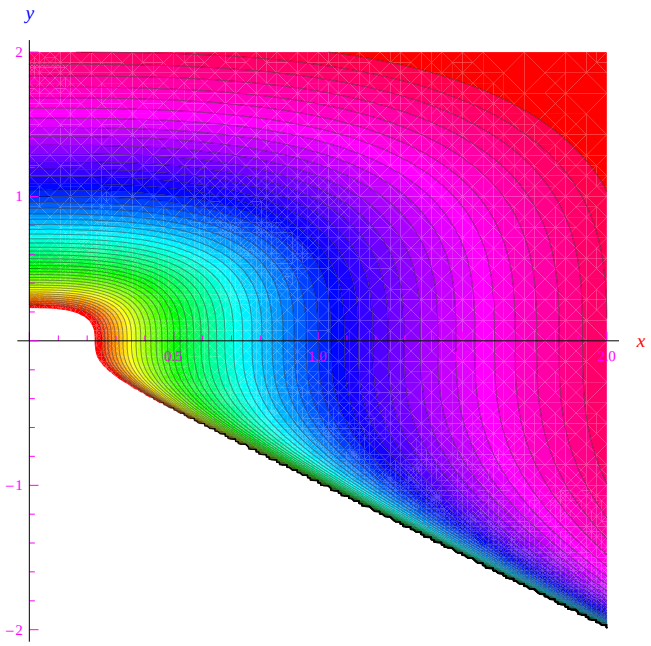

```
Export["lox3d.eps", plotf3d]
```

lox3d.eps

```
Clear[f, plotf3d]
```

```
f[x_, y_] := Log[x^3 + y^3]
```

```
livellof3 = ContourPlot[
  f[x, y],
  {x, 0, 2}, {y, -2, 2},
  Axes → True,
  Frame → False,
  ColorFunction → Hue,
  Contours → 50,
  AxesLabel →
   {
    Style["x", Small, Red],
    Style["y", Small, Blue]
   }
 ]
```




```
Export["livellof3.eps", livellof3]
```

```
livellof3.eps
```

In[1]:=
```
Clear[f, plot]
```

■ L'istruzione RegionPlot

L'istruzione **RegionPlot** permette di visualizzare l'insieme delle soluzioni di una disequazione in due variabili $\phi(x, y) \geq 0$, (o più in generale di un sistema di disequazioni), dove ϕ è una funzione assegnata delle variabili reali x, y. L'argomento di **RegionPlot** è un *predicato* ovvero una combinazione logica di disequazioni, implementata dall'operatore AND.

In[2]:=
```
? RegionPlot
```

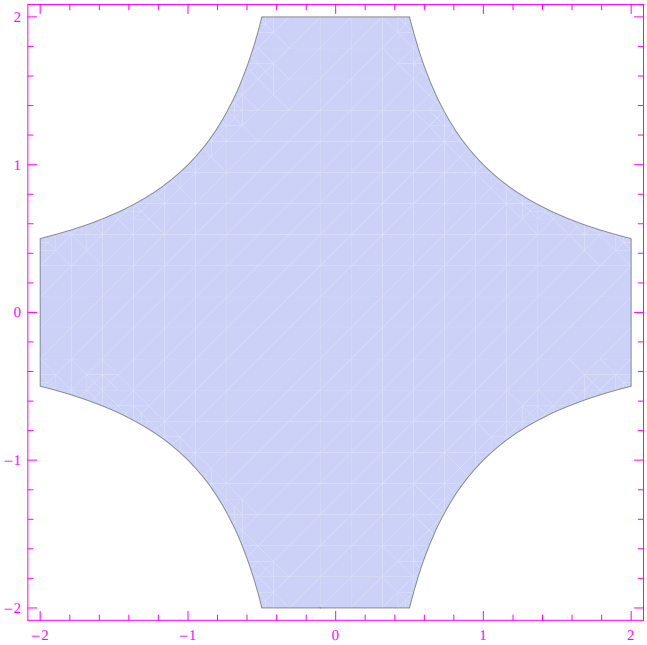
RegionPlot[*pred*, {x, x_{min}, x_{max}}, {y, y_{min}, y_{max}}] makes a plot showing the region in which *pred* is True. ≫

In[7]:=
```
SetOptions[
  RegionPlot,
  FrameStyle → Directive[Hue[5 / 6], 7]
 ];
```

Ad esempio, supponiamo di voler risolvere $x\,y \leq 1$:

In[29]:=
```
RegionPlot[
 Abs[x * y] ≤ 1,
 {x, -2, 2}, {y, -2, 2}
]
```

Out[29]=



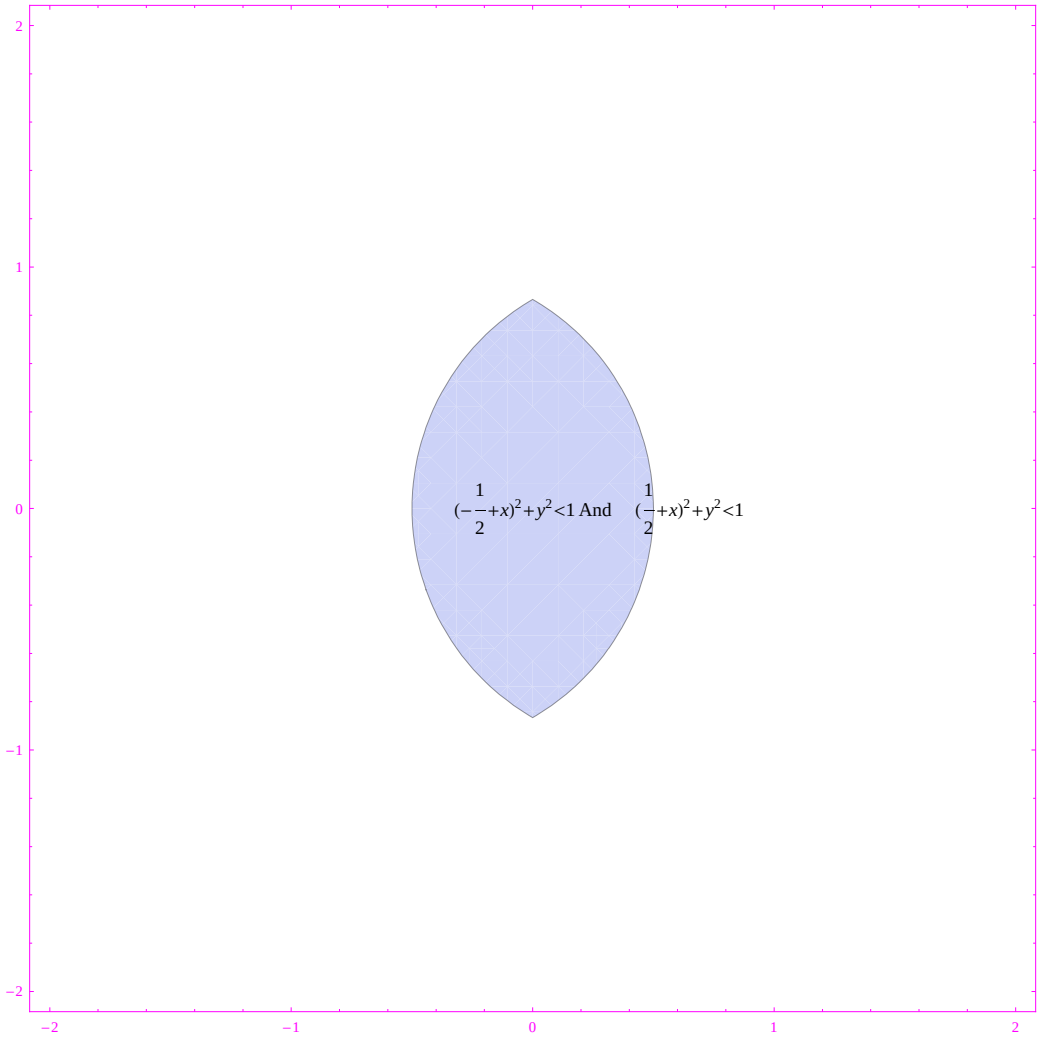
che è il dominio illimitato compreso tra le iperboli $y = \frac{1}{x}$ e $y = -\frac{1}{x}$

In[9]:=
```
a = (-1/2 + x)^2 + y^2 < 1;
b = (1/2 + x)^2 + y^2 < 1;
```

In[39]:=
```
tb = Table[
  RegionPlot[
   f[a, b],
   {x, -2, 2}, {y, -2, 2},
   PlotLabel → f,
   ImageSize → {500, 500},
   Epilog →
    {
     Text["(-1/2+x)^2+y^2<1" f, {0, 0}],
     Text["(1/2+x)^2+y^2<1", {0.65, 0}]
    }
  ],
  {
   f,
   {And, Or, Xor, Implies, Nand, Nor}
  }
 ]
```

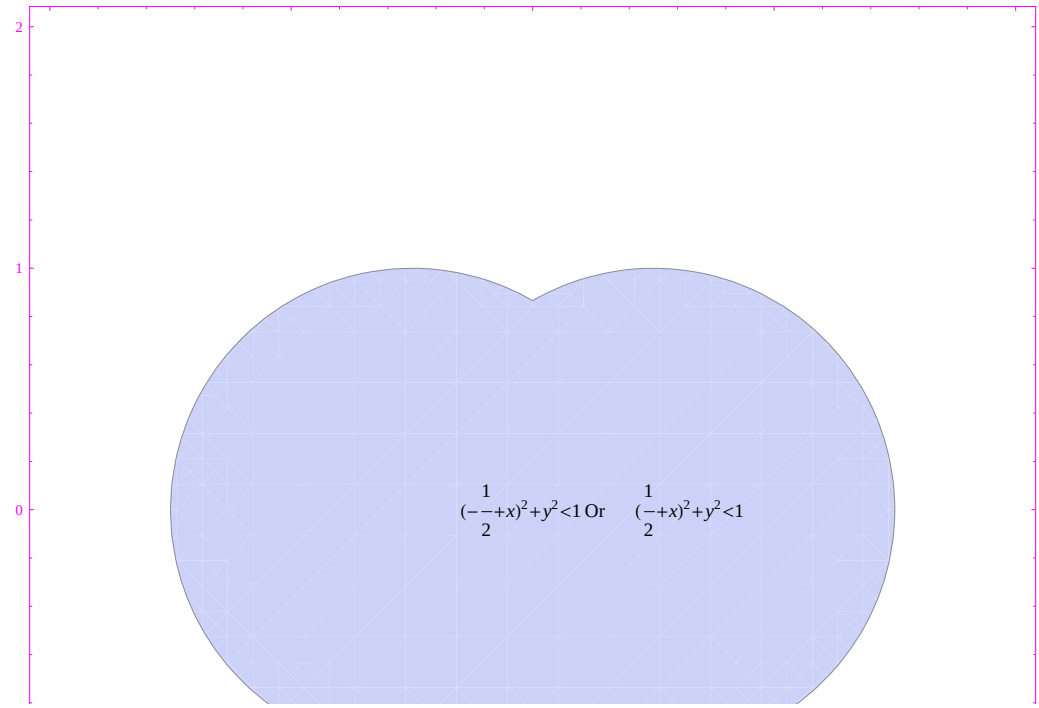
Out[39]= {

And


$(-\frac{1}{2}+x)^2+y^2<1$ And $(\frac{1}{2}+x)^2+y^2<1$
2
1
0
−1
−2
−2
−1
0
1
2

,

Or


$(-\frac{1}{2}+x)^2+y^2<1$ Or $(\frac{1}{2}+x)^2+y^2<1$
2
1
0

,

In[40]:= `SetDirectory["G:\\Siti\\extrabyte2\\Mathematica\\TUTORIAL_Mathematica"];`

In[41]:= `Export["xor.gif", tb];`

L'insieme di Cantor quale Macchina ricorsiva topologica

La possibilità di utilizzare le liste come argomento di una funzione, può essere utilizzata nella costruzione dell'*insieme di Cantor* che, come è noto, è il risultato di un procedimento ricorsivo. Più precisamente, si parte da un segmento di lunghezza unitaria rappresentato dall'intervallo chiuso [0,1], lo si divide in tre segmenti di pari lunghezza (1/3) rimuovendo poi il segmento centrale. Il procedimento viene in seguito applicato ai due segmenti residui, e così via all'infinito. Ciò che rimane è, appunto, l'insieme di Cantor. In generale, comunque prendiamo $a,\ b \in R$ con $b > a$, l'insieme $\{a, b\}$ è interpretato da *Mathematica* con l'head **List**. Infatti:

```
Head[{a, b}]
```

```
List
```

Per quanto visto, il procedimento di Cantor consiste nel prendere un segmento $[a, b]$ dividerlo in tre parti uguali e rimuovere la parte centrale. Definiamo allora la funzione:

```
Clear[f, g, g1]
```

$$f[\{a\_, b\_\}] := \left\{\left\{a, a + \frac{b-a}{3}\right\}, \left\{b - \frac{b-a}{3}, b\right\}\right\}$$

Ad esempio:

```
f[{0, 1}]
```

$$\left\{\left\{0, \frac{1}{3}\right\}, \left\{\frac{2}{3}, 1\right\}\right\}$$

che è proprio quello che volevamo, a patto di osservare che tale funzione restituisce l'insieme degli estremi degli intervalli e non gli intervalli. Riapplichiamo il procedimento i.e. la funzione f al suo output:

$$f\left[\left\{\left\{0, \frac{1}{3}\right\}, \left\{\frac{2}{3}, 1\right\}\right\}\right]$$

$$\left\{\left\{\left\{0, \frac{1}{3}\right\}, \left\{\frac{2}{9}, \frac{5}{9}\right\}\right\}, \left\{\left\{\frac{4}{9}, \frac{7}{9}\right\}, \left\{\frac{2}{3}, 1\right\}\right\}\right\}$$

Riapplichiamo:

$$f\left[\left\{\left\{\left\{0, \frac{1}{3}\right\}, \left\{\frac{2}{9}, \frac{5}{9}\right\}\right\}, \left\{\left\{\frac{4}{9}, \frac{7}{9}\right\}, \left\{\frac{2}{3}, 1\right\}\right\}\right\}\right]$$

$$\left\{\left\{\left\{\left\{0, \frac{1}{3}\right\}, \left\{\frac{2}{9}, \frac{5}{9}\right\}\right\}, \left\{\left\{\frac{4}{27}, \frac{13}{27}\right\}, \left\{\frac{10}{27}, \frac{19}{27}\right\}\right\}\right\}, \left\{\left\{\left\{\frac{8}{27}, \frac{17}{27}\right\}, \left\{\frac{14}{27}, \frac{23}{27}\right\}\right\}, \left\{\left\{\frac{4}{9}, \frac{7}{9}\right\}, \left\{\frac{2}{3}, 1\right\}\right\}\right\}\right\}$$

che è un risultato corretto a meno di livelli di annidamento che possono essere rimossi tramite l'istruzione **Flatten**. Appare chiaro a questo punto che l'insieme di Cantor è una *macchina ricorsiva*, poichè per un assegnato input che denotiamo con x_0, la funzione emette un output $f(x_0)$ che viene poi riprocessato calcolando $f(f(x_0))$ e così via all'infinito. Tutto questo ci rimanda al comando **Nest** visto in precedenza, con la differenza che qui abbiamo una funzione il cui argomento è una lista e non un numero.

Possiamo utilizzare il comando **Map** in modo da forzare **f** ad agire sul proprio output:

```
Map[f, f[{0, 1}]]
```

$$\left\{\left\{\left\{0, \frac{1}{9}\right\}, \left\{\frac{2}{9}, \frac{1}{3}\right\}\right\}, \left\{\left\{\frac{2}{3}, \frac{7}{9}\right\}, \left\{\frac{8}{9}, 1\right\}\right\}\right\}$$

"Flattando":

```
Flatten[Map[f, f[{0, 1}]]]
```

$$\left\{0, \frac{1}{9}, \frac{2}{9}, \frac{1}{3}, \frac{2}{3}, \frac{7}{9}, \frac{8}{9}, 1\right\}$$

Risultato non corretto, in quanto **Flatten** elimina tutti i livelli di annidamento, mentre dobbiamo eliminarne solo uno. Perciò fissiamo l'ordine di annidamento a 1:

```
Flatten[
 (*lista da flattare*)
 Map[f, f[{0, 1}]],
 (*ordine di annidamento*)
 1
]
```

$$\left\{\left\{0, \frac{1}{9}\right\}, \left\{\frac{2}{9}, \frac{1}{3}\right\}, \left\{\frac{2}{3}, \frac{7}{9}\right\}, \left\{\frac{8}{9}, 1\right\}\right\}$$

che è il risultato corretto. Ciò suggerisce di definire la funzione:

```
g[lista_List] := Flatten[
  (*lista da flattare*)
  Map[f, lista],
  (*ordine di annidamento*)
  1
 ]
```

Ad esempio:

```
g[f[{0, 1}]]
```

$$\left\{\left\{0, \frac{1}{9}\right\}, \left\{\frac{2}{9}, \frac{1}{3}\right\}, \left\{\frac{2}{3}, \frac{7}{9}\right\}, \left\{\frac{8}{9}, 1\right\}\right\}$$

```
g[g[f[{0, 1}]]]
```

$$\left\{\left\{0, \frac{1}{27}\right\}, \left\{\frac{2}{27}, \frac{1}{9}\right\}, \left\{\frac{2}{9}, \frac{7}{27}\right\}, \left\{\frac{8}{27}, \frac{1}{3}\right\}, \left\{\frac{2}{3}, \frac{19}{27}\right\}, \left\{\frac{20}{27}, \frac{7}{9}\right\}, \left\{\frac{8}{9}, \frac{25}{27}\right\}, \left\{\frac{26}{27}, 1\right\}\right\}$$

Risultati corretti! Ora siamo in grado di utilizzare **Nest** in modo da implementare la macchina ricorsiva:

```
g1[n_] := Nest[g, f[{0, 1}], n - 1]
```

L'intero naturale *n* definisce la *n* – esima iterazione. Ad esempio:

```
g1[1]
```

$$\left\{\left\{0, \frac{1}{3}\right\}, \left\{\frac{2}{3}, 1\right\}\right\}$$

che è, appunto, la prima iterazione. Siccome non possiamo utilizzare **g1** per $n = 0$, conviene utilizzare l'istruzione **Which**

```
Clear[g1]
```

```
g1[n_Integer] := Which[
  n == 0, {{0, 1}},
  n > 0, Nest[g, f[{0, 1}], n - 1]
 ]
```

Ad esempio, alla terza iterazione ($n = 4$) otteniamo 81 intervalli i cui estremi sono:

```
g1[4]
```

$$\left\{\left\{0, \frac{1}{81}\right\}, \left\{\frac{2}{81}, \frac{1}{27}\right\}, \left\{\frac{2}{27}, \frac{7}{81}\right\}, \left\{\frac{8}{81}, \frac{1}{9}\right\}, \left\{\frac{2}{9}, \frac{19}{81}\right\}, \left\{\frac{20}{81}, \frac{7}{27}\right\}, \left\{\frac{8}{27}, \frac{25}{81}\right\}, \left\{\frac{26}{81}, \frac{1}{3}\right\}, \left\{\frac{2}{3}, \frac{55}{81}\right\}, \left\{\frac{56}{81}, \frac{19}{27}\right\}, \left\{\frac{20}{27}, \frac{61}{81}\right\}, \left\{\frac{62}{81}, \frac{7}{9}\right\}, \left\{\frac{8}{9}, \frac{73}{81}\right\}, \left\{\frac{74}{81}, \frac{25}{27}\right\}, \left\{\frac{26}{27}, \frac{79}{81}\right\}, \left\{\frac{80}{81}, 1\right\}\right\}$$

La lista completa delle iterazioni fino alla terza è:

```
Table[
 g1[n],
 {n, 0, 4}
]
```

$$\left\{\{\{0, 1\}\}, \left\{\left\{0, \frac{1}{3}\right\}, \left\{\frac{2}{3}, 1\right\}\right\}, \left\{\left\{0, \frac{1}{9}\right\}, \left\{\frac{2}{9}, \frac{1}{3}\right\}, \left\{\frac{2}{3}, \frac{7}{9}\right\}, \left\{\frac{8}{9}, 1\right\}\right\}, \left\{\left\{0, \frac{1}{27}\right\}, \left\{\frac{2}{27}, \frac{1}{9}\right\}, \left\{\frac{2}{9}, \frac{7}{27}\right\}, \left\{\frac{8}{27}, \frac{1}{3}\right\}, \left\{\frac{2}{3}, \frac{19}{27}\right\}, \left\{\frac{20}{27}, \frac{7}{9}\right\}, \left\{\frac{8}{9}, \frac{25}{27}\right\}, \left\{\frac{26}{27}, 1\right\}\right\}, \left\{\left\{0, \frac{1}{81}\right\}, \left\{\frac{2}{81}, \frac{1}{27}\right\}, \left\{\frac{2}{27}, \frac{7}{81}\right\}, \left\{\frac{8}{81}, \frac{1}{9}\right\}, \left\{\frac{2}{9}, \frac{19}{81}\right\}, \left\{\frac{20}{81}, \frac{7}{27}\right\}, \left\{\frac{8}{27}, \frac{25}{81}\right\}, \left\{\frac{26}{81}, \frac{1}{3}\right\}, \left\{\frac{2}{3}, \frac{55}{81}\right\}, \left\{\frac{56}{81}, \frac{19}{27}\right\}, \left\{\frac{20}{27}, \frac{61}{81}\right\}, \left\{\frac{62}{81}, \frac{7}{9}\right\}, \left\{\frac{8}{9}, \frac{73}{81}\right\}, \left\{\frac{74}{81}, \frac{25}{27}\right\}, \left\{\frac{26}{27}, \frac{79}{81}\right\}, \left\{\frac{80}{81}, 1\right\}\right\}\right\}$$

Per quanto detto prima, ciò che abbiamo prodotto è l'insieme degli estremi degli intervalli generati a ogni ricorsione, ma a noi interessano gli intervalli per poi essere disegnati utilizzando la primitiva grafica **`Line`**. Ad esempio, per disegnare il segmento di estremi 0 e 1 dobbiamo innanzitutto assegnare un riferimento cartesiano ortogonale del piano *Oxy*, per cui le coordinate cartesiane degli estremi sono (0,0) e (1,0). In questo caso specifico, la sintassi è

```
Line[{{0, 0}, {1, 0}}]
```

```
Line[{{0, 0}, {1, 0}}]
```

Ci sono due modi per graficare il segmento in questione. La prima consiste nel processarlo attraverso l'istruzione **`Graphics`**

```
segmento = Graphics[
  (*primitiva grafica*)
  Line[{{0, 0}, {1, 0}}]
 ]
```

Notiamo che l'istruzione **Graphics** ha l'attributo **Listable:**

```
Graphics[
 {
  Line[{{0, 0}, {1, 0}}]
 }
]
```

E questo è utile perchè permette di inserire le opzioni ed eventualmente altre primitive come ad esempio **Point**:

```
Graphics[
 {
  (*opzioni*)
  Thickness[0.002],
  Blue,
  (*primitive grafiche*)
  Line[{{0, 0}, {1, 0}}],
  Point[{0, 0}],
  Point[{1, 0}]
 }
]
```


Il secondo approccio utilizza il comando **Epilog** nell'istruzione grafica **Plot**. Il fatto di non dover graficare alcuna funzione, equivale per *Mathematica* a graficare la funzione **Null**

```
? Null
```

Null is a symbol used to indicate the absence of an expression or a result. It is not displayed in ordinary output. When Null appears as a complete output expression, no output is printed. ≫

Quindi:

```
Plot[
 Null,
 {x, 0, 2},
 Axes → False,
 Epilog →
  {
   (*opzioni*)
   Thickness[0.002],
   Blue,
   (*primitive grafiche*)
   Line[{{0, 0}, {1, 0}}],
   Point[{0, 0}],
   Point[{1, 0}]
  }
]
```



```
(*usiamo Remove in quanto più potente di Clear*)

Remove[f, g, g1]
```

Prima di implementare l'insieme di Cantor, consideriamo una generica macchina ricorsiva topologica. Ad esempio:

```
f[Line[{{a_, 0}, {b_, 0}}]] := Line[{{a (a + 1), 0}, {b (b^-1 + 1), 0}}]

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

Line[{{2, 0}, {3, 0}}]

Map[f, {f[Line[{{1, 0}, {2, 0}}]]}]

{Line[{{6, 0}, {4, 0}}]}

g[intervallo_] := Flatten[
  (*lista da flattare*)
  Map[f, {intervallo}],
  (*ordine di annidamento*)
  1
 ]
```

```
g[g[f[Line[{{1, 0}, {2, 0}}]]]]
```

```
{f[{Line[{{6, 0}, {4, 0}}]}]}
```

Non restituisce il risultato corretto per via delle parentesi {} che però, sono necessarie. Nel caso contrario, non è possibile utilizzare l'istruzione **Map** che definisce la funzione **g**. Dobbiamo allora dare l'attributo **Listable** alla funzione **f**, in modo da bypassare le parentesi.

```
SetAttributes[f, Listable]
```

```
g[g[g[f[Line[{{1, 0}, {2, 0}}]]]]]
```

```
{Line[{{1806, 0}, {6, 0}}]}
```

```
g1[n_Integer] := Which[
  n == 0, Line[{{1, 0}, {2, 0}}],
  n > 0, Nest[g, f[Line[{{1, 0}, {2, 0}}]], n - 1]
 ]
```

```
g1[2]
```

```
{Line[{{6, 0}, {4, 0}}]}
```

```
SetOptions[
  {
   Plot,
   ListPlot,
   ListLinePlot,
   Plot3D,
   ParametricPlot
  },
  TicksStyle -> Directive[
    Hue[5 / 6],
    9
   ]
 ];
```

Ora siamo in grado di implementare la Macchina di Cantor

```
Remove[f, g, g1]
```

$$\texttt{f[Line[\{\{a\_, 0\}, \{b\_, 0\}\}]] := Line}\left[\left\{\{a, 0\}, \left\{a + \frac{b - a}{3}, 0\right\}\right\}\right]\texttt{; SetAttributes[f, Listable]}$$

```
f[Line[{{0, 0}, {1, 0}}]]
```

$$\text{Line}\left[\left\{\{0, 0\}, \left\{\frac{1}{3}, 0\right\}\right\}\right]$$

Il codice funziona in parte, poiché restituisce solo uno dei segmenti.

$$\texttt{f[Line[\{\{a\_, 0\}, \{b\_, 0\}\}]] := }\left\{\texttt{Line}\left[\left\{\{a, 0\}, \left\{a + \frac{b - a}{3}, 0\right\}\right\}\right], \texttt{Line}\left[\left\{\left\{b - \frac{b - a}{3}, 0\right\}, \{b, 0\}\right\}\right]\right\}\texttt{;}$$

```
SetAttributes[f, Listable]
```

```
f[f[Line[{{0, 0}, {1, 0}}]]] // Graphics
```

che è il risultato corretto. Procediamo quindi come nel caso precedente, defininendo:

```
g[intervallo_] := Flatten[
  (*lista da flattare*)
  Map[f, {intervallo}],
  (*ordine di annidamento*)
  1
 ]
```

```
g[f[Line[{{0, 0}, {1, 0}}]]]
```

$$\left\{\left\{\text{Line}\left[\left\{\{0, 0\}, \left\{\frac{1}{9}, 0\right\}\right\}\right], \text{Line}\left[\left\{\left\{\frac{2}{9}, 0\right\}, \left\{\frac{1}{3}, 0\right\}\right\}\right]\right\}, \left\{\text{Line}\left[\left\{\left\{\frac{2}{3}, 0\right\}, \left\{\frac{7}{9}, 0\right\}\right\}\right], \text{Line}\left[\left\{\left\{\frac{8}{9}, 0\right\}, \{1, 0\}\right\}\right]\right\}\right\}$$

Anche in questo caso non possiamo utilizzare **g1** per $n = 0$, per cui utilizziamo l'istruzione **Which**

```
g1[n_Integer] := Which[
  n == 0, Line[{{0, 0}, {1, 0}}],
  n > 0, Nest[g, f[Line[{{0, 0}, {1, 0}}]], n - 1]
 ]
```

```
g1[10] // Graphics
```

```
cantor[n_] := Plot[
  Null,
  {x, 0, 1},
  ImageSize → {500, 500},
  Axes → False,
  Epilog → {
    Red,
    Thickness[0.003],
    g1[n],
    Point[{0, 0}],
    Point[{1, 0}],
    Text[Style["1", Medium, Red], {1, 0.05}],
    Text[Style["0", Medium, Red], {0, 0.05}]
   }
 ]
```

```
cantor[4]
```


```
moviecantor = Table[cantor[n], {n, 0, 20}];
```

```
$Aborted
```

```
Export["moviecantor.gif", moviecantor];
```

```
Remove[f, segmento]
```

```
segmento[a_, b_] := Line[{{a, 0}, {b, 0}}]
```

```
f[a_, b_] := {segmento[a + (b - a)/3, a], segmento[b - (b - a)/3, b]}; SetAttributes[f, Listable]
```

```
f[0, 1]
```

$$\left\{\text{Line}\left[\left\{\left\{\frac{1}{3}, 0\right\}, \{0, 0\}\right\}\right], \text{Line}\left[\left\{\left\{\frac{2}{3}, 0\right\}, \{1, 0\}\right\}\right]\right\}$$

che è il risultato corretto.

```
f[0, 1]
```

$$\left\{\text{Line}\left[\left\{\left\{\frac{1}{3}, 0\right\}, \{0, 0\}\right\}\right], \text{Line}\left[\left\{\left\{\frac{2}{3}, 0\right\}, \{1, 0\}\right\}\right]\right\}$$

A questo punto definiamo la funzione

```
g[intervallo_] := Flatten[
  (*lista da flattare*)
  Map[f, {intervallo}],
  (*ordine di annidamento*)
  1
 ]
```

```
g[g[f[0, 1]]]
```

$$\left\{f\left[f\left[\text{Line}\left[\left\{\left\{\frac{1}{3}, 0\right\}, \{0, 0\}\right\}\right]\right]\right], f\left[f\left[\text{Line}\left[\left\{\left\{\frac{2}{3}, 0\right\}, \{1, 0\}\right\}\right]\right]\right]\right\}$$

```
f[0, 1]
```

$$\left\{\text{Line}\left[\left\{\left\{\frac{1}{3}, 0\right\}, \{0, 0\}\right\}\right], \text{Line}\left[\left\{\left\{\frac{2}{3}, 0\right\}, \{1, 0\}\right\}\right]\right\}$$

```
f[0, 10]
```

$$\left\{\text{Line}\left[\left\{\left\{\frac{10}{3}, 0\right\}, \{0, 0\}\right\}\right], \text{Line}\left[\left\{\left\{\frac{20}{3}, 0\right\}, \{10, 0\}\right\}\right]\right\}$$

```
g[f[0, 1]]
```

$$\left\{f\left[\text{Line}\left[\left\{\left\{\frac{1}{3}, 0\right\}, \{0, 0\}\right\}\right]\right], f\left[\text{Line}\left[\left\{\left\{\frac{2}{3}, 0\right\}, \{1, 0\}\right\}\right]\right]\right\}$$

Funzione di Cantor (incompleto)

La funzione zeta di Riemann

Funzioni non elementarmente esprimibili

Argomenti avanzati

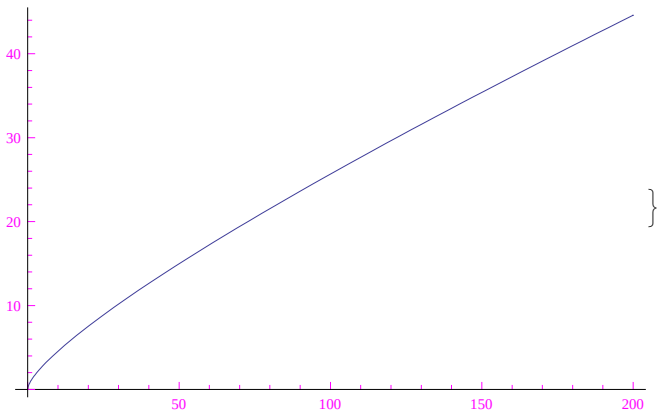
L'analisi di Riesel-Gohl sulla Congettura di Riemann

```
SetOptions[
  {
   Plot,
   ListLinePlot
  },
  TicksStyle -> Directive[
    Hue[5 / 6],
    7
   ]
 ];
```

Approssimazione di Riemann

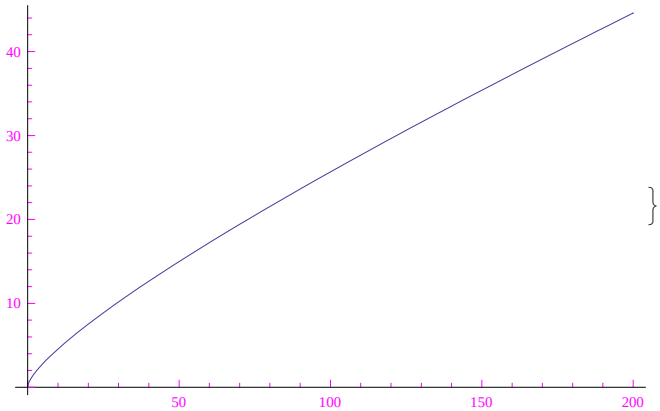
```
R[x_, n_] := 1 + Sum[(Log[x])^k / (k! k * Zeta[k + 1]), {k, n}]

(*begin routine Ilan Vardi*)

lista1[x_, n_] := N[Log[x]]^Range[n]

lista2[n_] := Map[
  (1 / (#! # N[Zeta[# + 1]])) &,
  Range[n]
 ]

Rvardi[x_, n_] := 1 + lista1[x, n].lista2[n]

(*end routine Ilan Vardi*)

Plot[
  {Rvardi[x, 100]},
  {x, 0, 200}
 ] // Timing
```

{0.328,



}

```
Plot[
  {R[x, 100]},
  {x, 0, 200}
 ] // Timing
```

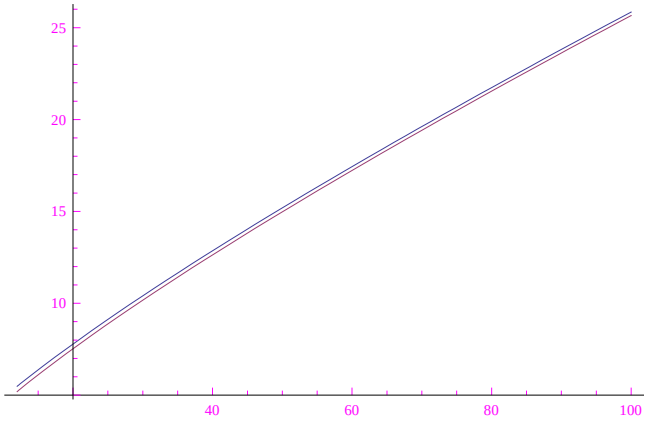
{0.453, 

}

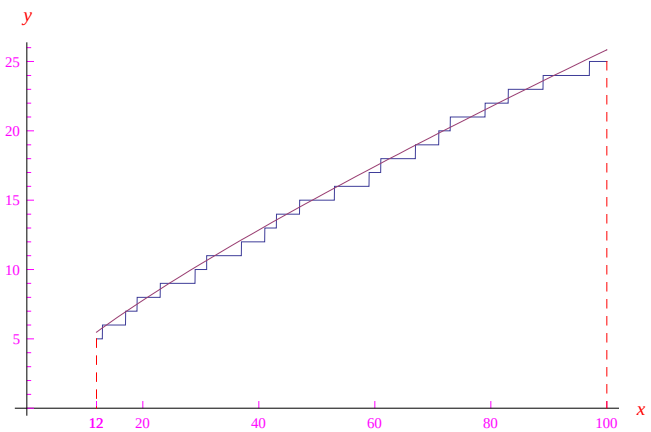
Calcolo del termine $G_{N_0}(x)$

```
g[t_] := 1/(t (t^2 - 1) Log[t])

J[x_] := NIntegrate[
  g[t],
  {t, x, +∞}
 ]

dataJ = Table[
   J[x],
   {x, 2, 200, 0.5}
  ];

Iapp = Interpolation[dataJ]

InterpolatingFunction[{{1., 397.}}, <>]

G[x_, N0_] := Sum[
  (*argomento della sommatoria*)
  MoebiusMu[k]/k * Iapp[x^(1/k)],
  (*indice della sommatoria*)
  {k, N0}]

Clear[G]

(*approssimazione di riesel gohl*)

G[x_] := 1/π ArcTan[π/Log[x]]

R0[x_] := R[x, 100] + G[x] // N

ticks0 = Table[k, {k, 20, 100, 20}];
```

```
Plot[
 {
  R0[x],
  R[x, 100]
 },
 {x, 12, 100}
]
```



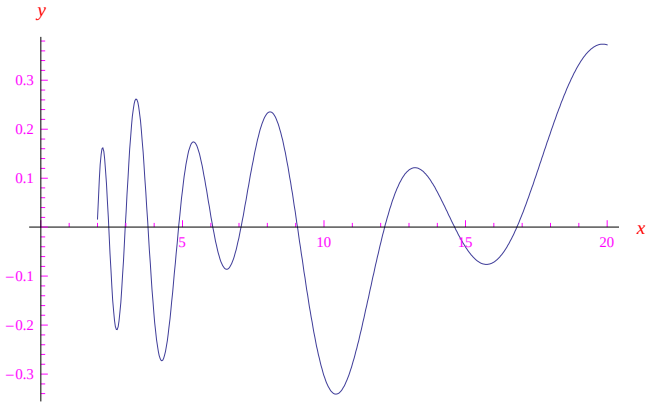
```
Plot[
 {
  PrimePi[x],
  R0[x]
 },
 {x, 12, 100},
 AxesOrigin → {0, 0},
 AxesLabel →
  {
   Style["x", Small, Red],
   Style["y", Small, Red]
  },
 Ticks →
  {
   PrependTo[ticks0, 12]
  },
 Epilog →
  {
   Red,
   Dashed,
   Line[{{12, PrimePi[12]}, {12, 0}}],
   Line[{{100, PrimePi[100]}, {100, 0}}]
  }
]
```




```
(*zeri non banali della zeta di Riemann*)

ρ[n_] := N[ZetaZero[n]]

ψ[x_, k_, n_] := Re[ExpIntegralEi[ρ[n]/k * Log[N[x]]]]

(*coefficienti di Moebius*)

c[k_] := (-2 N[MoebiusMu[k]])/k

T[x_, n_, N01_] := Sum[c[k] * ψ[x, k, n], {k, 1, N01}]

Clear[T]

(*genera la lista dei coefficienti di Möbius*)
```

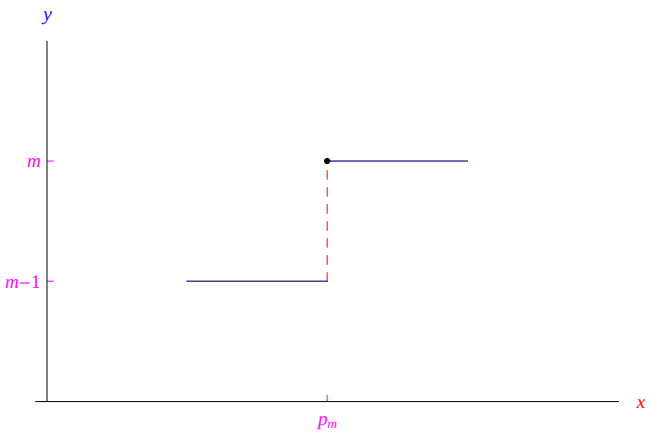
```
listaMoebius = Table[
   -2 MoebiusMu[k] // N / k,
   {k, 154}
  ];

(*ridefinisce gli indici della somma sui coefficienti non nulli*)

IndiciMoebius = Select[
   Range[154],
   listaMoebius[[#]] ≠ 0 &
  ];

(*calcola i coefficienti di Möbius non nulli*)

CoefficientiMoebius = listaMoebius[[IndiciMoebius]];

psi[x_, n_] = Re[ExpIntegralEi[ρ[n] / IndiciMoebius * Log[N[x]]]];

T[x_, n_] := CoefficientiMoebius . psi[x, n]

Dec[b_] := Range[
   (*punto iniziale*)
   2.,
   (*punto finale*)
   N[b],
   (*step*)
   N[(b - 2) / (b + 550)]
  ];

Tlist[b_, n_] := T[Dec[b], n];

M[b_, n_] := {
  (*prima riga*)
  Dec[b],
  (*seconda riga*)
  Tlist[b, n]
 }

Clear[ticksx]

ticksx[b_] := Table[k, {k, 12, b, 20}];

appendto[b_] := AppendTo[ticksx[b], 12];

plotT[b_, n_] := ListLinePlot[
  Transpose[M[b, n]],
  AxesOrigin → {0, 0},
  AxesLabel →
   {
    Style["x", Small, Red],
    Style["y", Small, Red]
   }
 ]
```

```
plotT[20, 1]
```



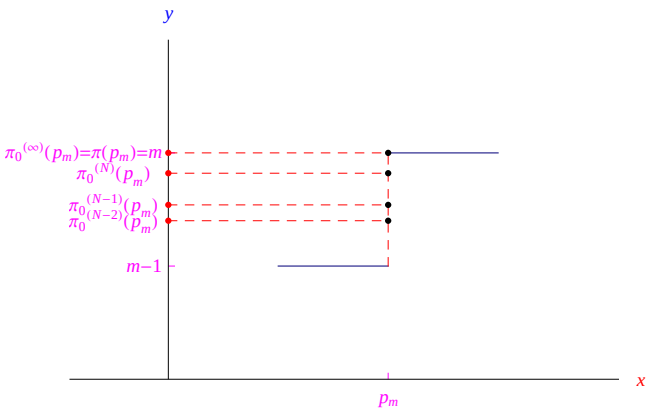

```
R0List[b_] := R0[Dec[b]];

T1[b_] := Tlist[b, 1];

T2[b_] := Tlist[b, 2]; T3[b_] := Tlist[b, 3]; T4[b_] := Tlist[b, 4]; T4[b_] := Tlist[b, 4];

Table[Prime[k], {k, 8}]

{2, 3, 5, 7, 11, 13, 17, 19}

H[b_, p_] := Sum[Tlist[b, n], {n, p}]

RpList[b_, p_] := R0List[b] + H[b, p]

primiList[b_] := PrimePi[Dec[b]];

listlineplot[b_, p_] := ListLinePlot[
  {
   Transpose[{Dec[b], primiList[b]}],
   Transpose[{Dec[b], RpList[b, p]}]
  },
  AxesOrigin → {0, 0},
  AxesLabel →
   {
    Style["x", Small, Red],
    Style["y", Small, Red]
   },
  Ticks →
   {
    {2, 3, 5, 7, 11, 13, 17, 19}
   }
 ]
```

```
loc0 = Plot[
  Which[
   x < 1, Null,
   x ≥ 1 && x ≤ 2, 1,
   x ≥ 2 && x ≤ 3, 2
  ],
  {x, 0, 4},
  PlotRange → {0, 3},
  Exclusions → x == 2,
  PlotStyle → Thickness[0.00255],
  AxesLabel →
   {
    Style["x", Small, Red],
    Style["y", Small, Blue]
   },
  Ticks →
   {
    {
     {2, "pm"}
    },
    {
     {1, "m-1"}, {2, "m"}
    }
   },
  Epilog → {
    {
     Red,
     Dashed,
     Line[{{2, 1}, {2, 2}}]
    },
    Point[{2, 2}]
   }
 ]
```



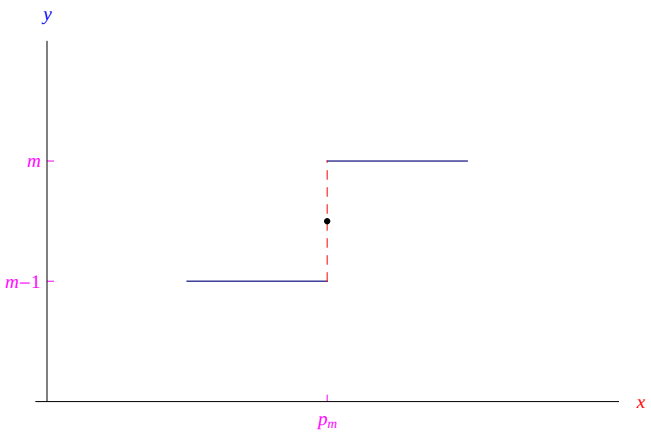

```
Export["loc0.eps", loc0];
```

```
loc0a = Plot[
  Which[
   x < 1, Null,
   x ≥ 1 && x ≤ 2, 1,
   x ≥ 2 && x ≤ 3, 2
  ],
  {x, -0.8, 4},
  PlotRange → {0, 3},
  Exclusions → x == 2,
  PlotStyle → Thickness[0.00255],
  AxesLabel →
   {
    Style["x", Small, Red],
    Style["y", Small, Blue]
   },
  Ticks →
   {
    {
     {2, "pm"}
    },
    {
     {1, "m-1"}, {2, "π0^(∞) (pm) = π(pm) = m"}
    }
   },
  Epilog → {
    {
     Red,
     Dashed,
     Line[{{2, 1}, {2, 2}}],
     Line[{{0, 1.4}, {2, 1.4}}],
     Point[{0, 1.4}],
     Line[{{0, 1.54}, {2, 1.54}}],
     Point[{0, 1.54}],
     Line[{{0, 1.82}, {2, 1.82}}],
     Point[{0, 1.82}],
     Line[{{0, 2}, {2, 2}}],
     Point[{0, 2}],
     Text[Style["π0^(N-2) (pm)", Small, Magenta], {-0.5, 1.4}],
     Text[Style["π0^(N-1) (pm)", Small, Magenta], {-0.5, 1.54}],
     Text[Style["π0^(N) (pm)", Small, Magenta], {-0.5, 1.82}]
    },
    Point[{2, 2}],
    Point[{2, 1.82}],
    Point[{2, 1.54}],
    Point[{2, 1.4}]
   }
 ]
```



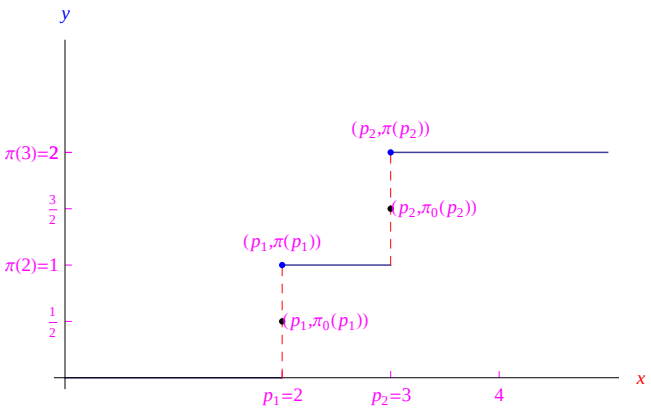

```
Export["loc0a.eps", loc0a];
```

```
loc1 = Plot[
  Which[
   x < 1, Null,
   x ≥ 1 && x ≤ 2, 1,
   x ≥ 2 && x ≤ 3, 2
  ],
  {x, 0, 4},
  PlotRange → {0, 3},
  Exclusions → x == 2,
  PlotStyle → Thickness[0.00255],
  AxesLabel →
   {
    Style["x", Small, Red],
    Style["y", Small, Blue]
   },
  Ticks →
   {
    {
     {2, "pm"}
    },
    {
     {1, "m-1"}, {2, "m"}
    }
   },
  Epilog → {
    {
     Red,
     Dashed,
     Line[{{2, 1}, {2, 2}}]
    },
    Point[{2, 3 / 2}]
   }
 ]
```




```
Export["loc1.eps", loc1];
```

```
locgraph = Plot[
  PrimePi[x],
  {x, 0, 5},
  Exclusions → {x == 2, x == 3},
  PlotRange → {-0.1, 3},
  PlotStyle → {
    Thickness[0.00255]
   },
  AxesLabel →
   {
    Style["x", Small, Red],
    Style["y", Small, Blue]
   },
  Ticks → {
    {
     {2, "p1=2"}, {3, "p2=3"}, 4
    },
    {
     {1, "π(2)=1"},
     {2, "π(3)=2"},
     2, 1 / 2, 3 / 2
    }
   },
  Epilog → {
    {
     Red,
     Dashed,
     Line[{{2, 0}, {2, 1}}],
     Line[{{3, 1}, {3, 2}}]
    },
    {
     Blue,
     Point[{2, 1}],
     Point[{3, 2}]
    },
    {
     Point[{2, 1 / 2}],
     Point[{3, 3 / 2}]
    },
    Text[Style["(p1,π(p1))", Small, Magenta], {2, 1.2}],
    Text[Style["(p1,π0(p1))", Small, Magenta], {2.4, 0.5}],
    Text[Style["(p2,π(p2))", Small, Magenta], {3, 2.2}],
    Text[Style["(p2,π0(p2))", Small, Magenta], {3.4, 1.5}]

   }
 ]
```




```
Export["locgraph.eps", locgraph];

Clear[Dec, Tlist, M, R0List, T1, T2, T3, H, Rplist, primiList, listlineplot]

Table[Prime[k], {k, 20}]
```

{2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71}

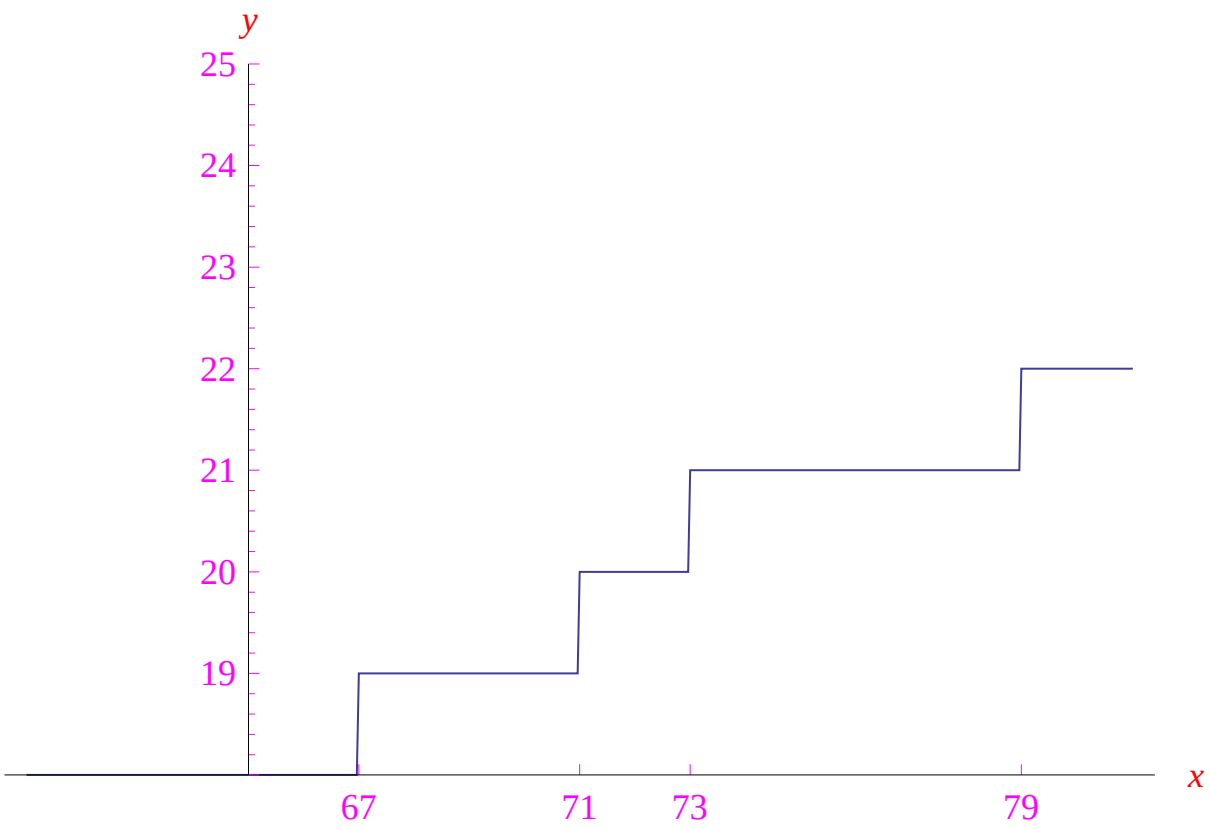
```
Dec[δ_] := Range[
  (*punto iniziale*)
  N[71 - δ],
  (*punto finale*)
  N[71 + δ],
  (*step*)
  N[2 δ / 550]
 ]

Tlist[δ_, n_] := T[Dec[δ], n];

M[δ_, n_] := {
  (*prima riga*)
  Dec[δ],
  (*seconda riga*)
  Tlist[δ, n]
 }

R0List[δ_] := R0[Dec[δ]];

T1[δ_] := Tlist[δ, 1];

H[δ_, p_] := Sum[Tlist[δ, n], {n, p}]

RpList[δ_, p_] := R0List[δ] + H[δ, p]

primiList[δ_] := PrimePi[Dec[δ]];
```

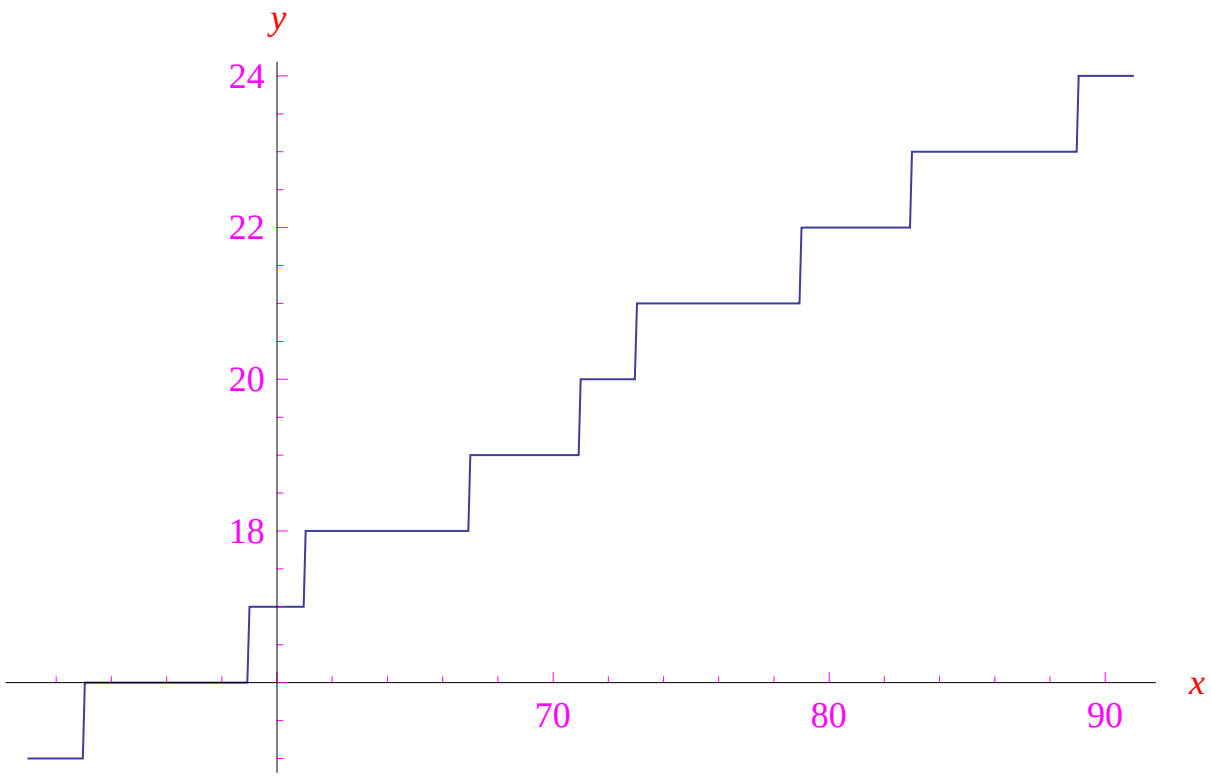
```
listlineplot0[δ_] := ListLinePlot[
  {
   Transpose[{Dec[δ], primiList[δ]}]
  },
  AxesOrigin → Automatic,
  AxesLabel →
   {
    Style["x", Small, Red],
    Style["y", Small, Red]
   },
  Ticks → {
    {
     67, 71, 73, 79, 83
    }
   },
  PlotRange → {18, 25}
 ]
```

```
Table[Prime[k], {k, 30}]
```

```
{2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47,
 53, 59, 61, 67, 71, 73, 79, 83, 89, 97, 101, 103, 107, 109, 113}
```

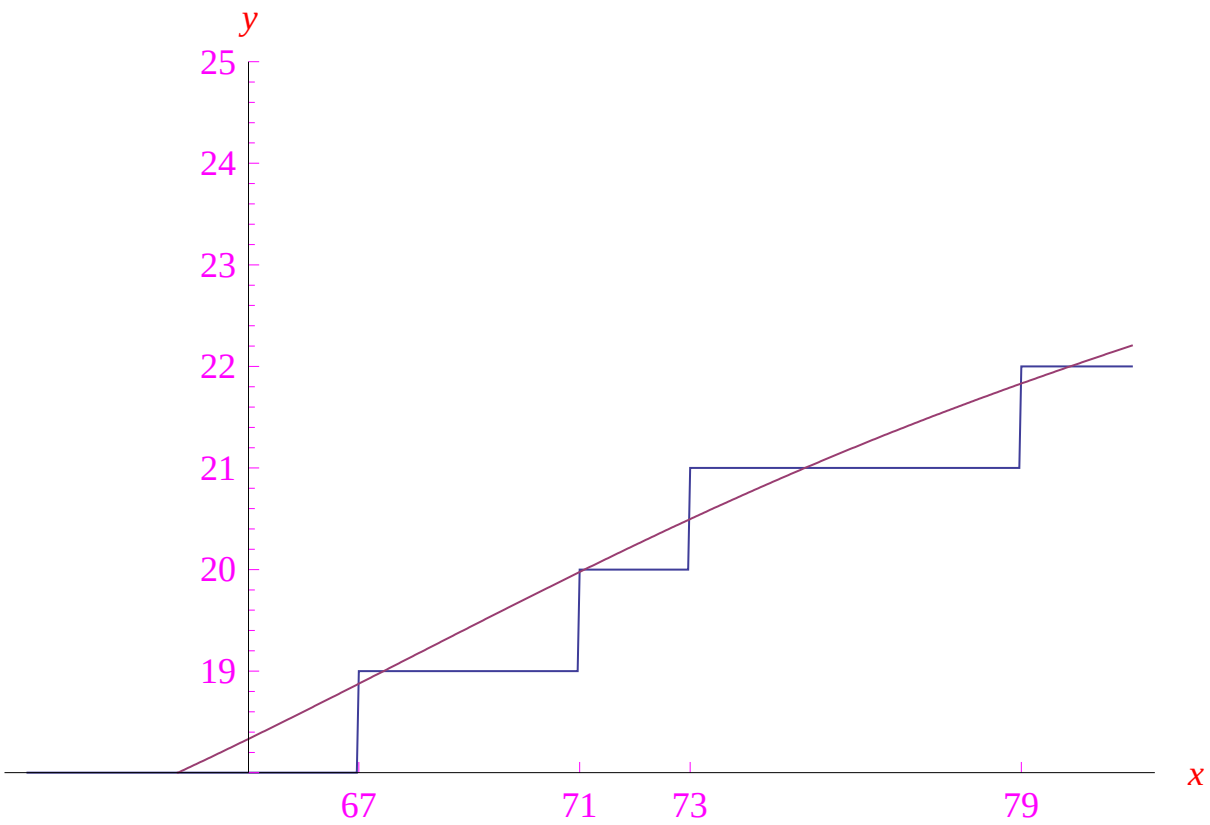
```
listlineplot0[10]
```



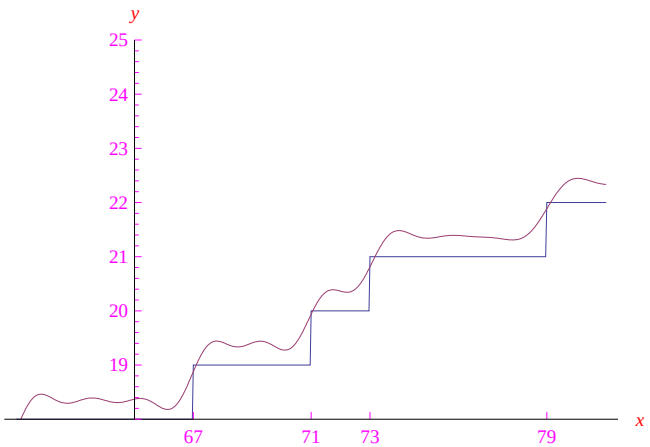



```
listlineplot[δ_, p_] := ListLinePlot[
  {
   Transpose[{Dec[δ], primiList[δ]}],
   Transpose[{Dec[δ], RpList[δ, p]}]
  },
  AxesOrigin → Automatic,
  AxesLabel →
   {
    Style["x", Small, Red],
    Style["y", Small, Red]
   },
  Ticks → {
    {
     67, 71, 73, 79, 83
    }
   },
  PlotRange → {18, 25}
 ]
```

```
listlineplot[10, 1]
```




```
apploc1 = listlineplot[10, 100]
```




```
Prime[20]
```

71

```
(*Export["apploc1.eps",apploc1];*)
```

```
Clear[ρ, Dec, Tlist, M, R0List, T1, T2, T3, H, Rplist, primiList, listlineplot, psi]
```

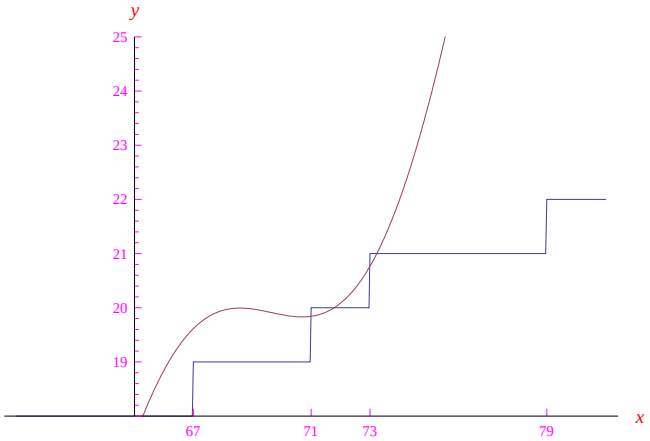
$$\rho[n\_] := \frac{1}{2} + \text{RandomReal}[] + i * \text{Im}[N[\text{ZetaZero}[n]]]$$

$$\text{psi}[x\_, n\_] = \text{Re}\left[\text{ExpIntegralEi}\left[\frac{\rho[n]}{\text{IndiciMoebius}} * \text{Log}[N[x]]\right]\right];$$

$$\tau[x\_, n\_] := \text{CoefficientiMoebius}.\text{psi}[x, n]$$

```
Dec[δ_] := Range[
  (*punto iniziale*)
  N[71 - δ],
  (*punto finale*)
  N[71 + δ],
  (*step*)
  N[2 δ / 550]
 ]

τlist[δ_, n_] := τ[Dec[δ], n];

M[δ_, n_] := {
  (*prima riga*)
  Dec[δ],
  (*seconda riga*)
  τlist[δ, n]
 }

R0List[δ_] := R0[Dec[δ]];

H[δ_, p_] := Sum[τlist[δ, n], {n, p}]

RpList[δ_, p_] := R0List[δ] + H[δ, p]

primiList[δ_] := PrimePi[Dec[δ]];

listlineplot[δ_, p_] := ListLinePlot[
  {
   Transpose[{Dec[δ], primiList[δ]}],
   Transpose[{Dec[δ], RpList[δ, p]}]
  },
  AxesOrigin → Automatic,
  AxesLabel →
   {
    Style["x", Small, Red],
    Style["y", Small, Red]
   },
  Ticks → {
    {
     67, 71, 73, 79, 83
    }
   },
  PlotRange → All
 ]
```

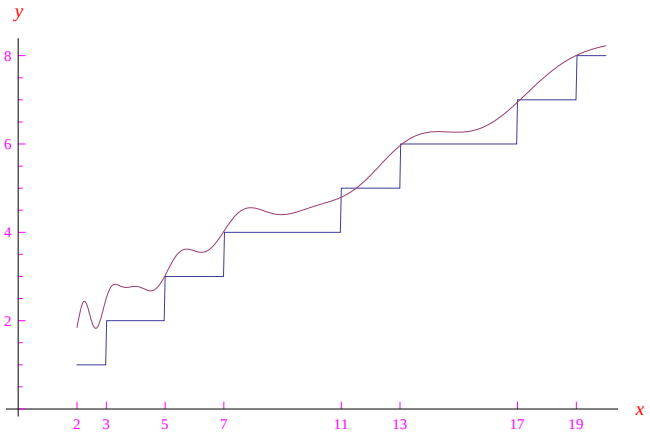
```
test01 = listlineplot[10, 3]
```




```
Export["test01.eps", test01];
```

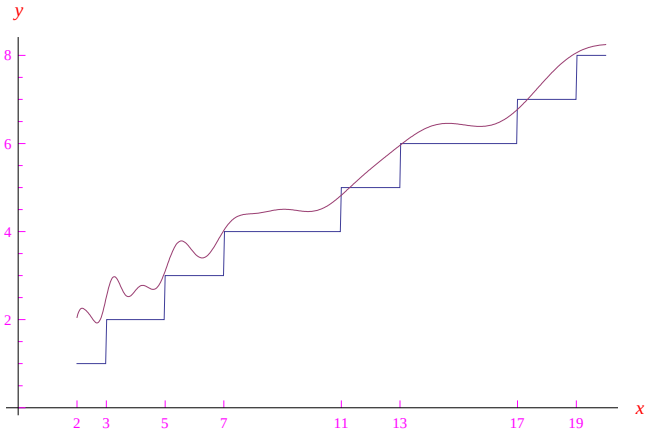
.

```
app2 = listlineplot[20, 2]
```



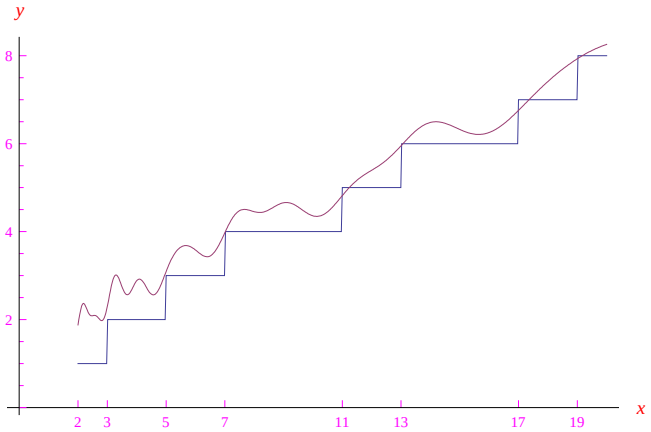

```
Export["app2.eps", app2];
```

```
app3 = listlineplot[20, 3]
```

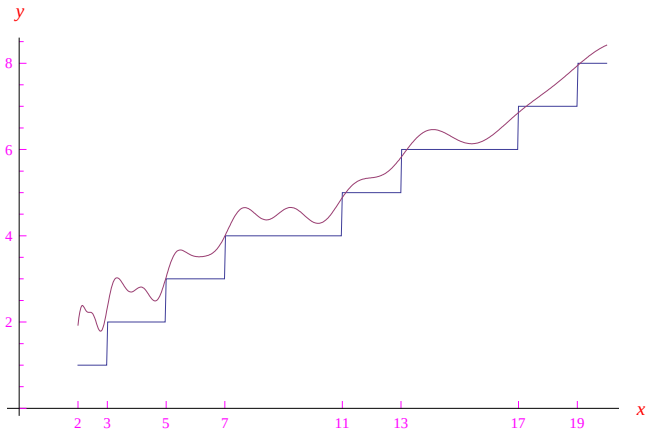



```
Export["app3.eps", app3];
```

```
app4 = listlineplot[20, 4]
```

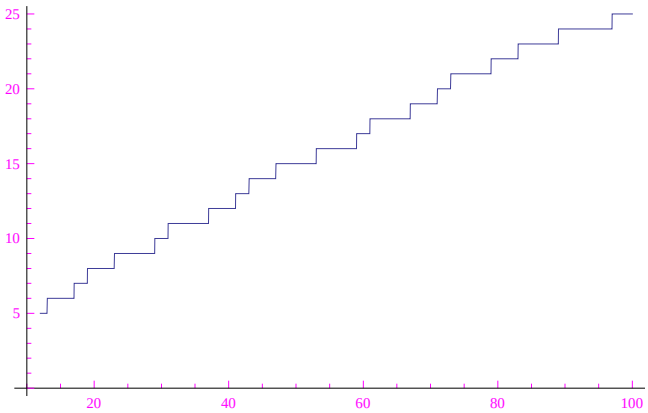



```
Export["app4.eps", app4];
app5 = listlineplot[20, 5]
```



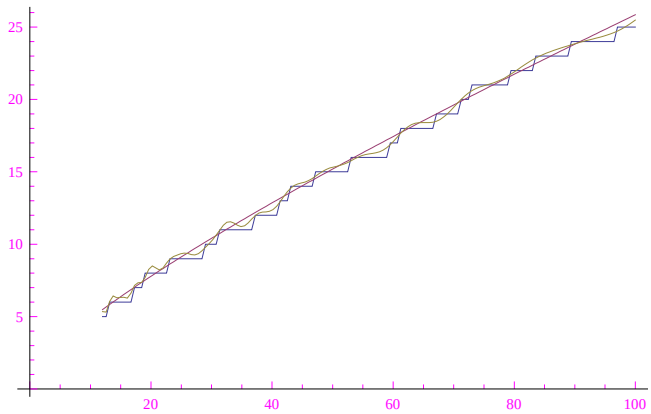

```
Export["app5.eps", app5];
app6 = listlineplot[20, 6];
Export["app6.eps", app6];
app7 = listlineplot[20, 7];
Export["app7.eps", app7];
app8 = listlineplot[20, 8];
Export["app8.eps", app8];
app9 = listlineplot[20, 9];
Export["app9.eps", app9];
app10 = listlineplot[20, 10]; Export["app10.eps", app10];
app11 = listlineplot[20, 11]; Export["app11.eps", app11];
app15 = listlineplot[20, 15]; Export["app15.eps", app15];
app18 = listlineplot[20, 18]; Export["app18.eps", app18];
app19 = listlineplot[20, 19]; Export["app19.eps", app19];
```

```
app20 = listlineplot[20, 20]; Export["app20.eps", app20];

app25 = listlineplot[20, 25]; Export["app25.eps", app25];

app30 = listlineplot[20, 30]; Export["app30.eps", app30];

app35 = listlineplot[20, 35]; Export["app35.eps", app35];

app40 = listlineplot[20, 40]; Export["app40.eps", app40];

app45 = listlineplot[20, 45]; Export["app45.eps", app45];

app48 = listlineplot[20, 48]; Export["app48.eps", app48];

app50 = listlineplot[20, 50]; Export["app50.eps", app50];

lista = {app1, app2, app3, app4, app5, app7, app8, app9, app10, app11,
   app15, app18, app19, app20, app25, app30, app35, app40, app45, app48, app50};

Export["primi.gif", lista];

Plot[PrimePi[x], {x, 12, 100}, AxesOrigin → {10, 0}]
```




```
T4 = Tlist[4];

T5 = Tlist[5]; T6 = Tlist[6]; T7 = Tlist[7];

Clear[T8, T9, T10]

T8 = Tlist[8]; T9 = Tlist[9]; T10 = Tlist[10];

Clear[R1List]

R1List = R0List + T1 + T2 + T3 + T4 + T5 + T6 + T7 + T8 + T9 + T10;
```

```
ListLinePlot[
 {
  Transpose[{Dec, primiList}],
  Transpose[{Dec, R0List}],
  Transpose[{Dec, R1List}]
 },
 AxesOrigin → {0, 0}
]
```




```
a = 1;

a >> "test1.dat"

a = 2;

a >>> "test1.dat"

FilePrint["test1.dat"]
```

```
1
2
```

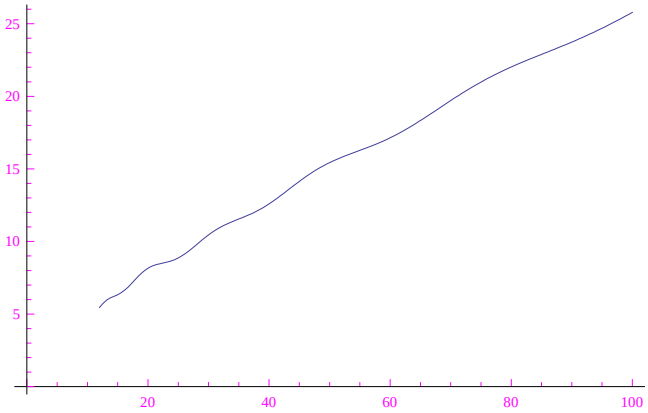
```
Do[
 a = a / 2 >>> "test1.dat",
 {5}
]

FilePrint["test1.dat"]
```

```
1
2
1
1/2
1/4
1/8
1/16
```

```
(*n=1,10*)

Do[
 Tlist[n] >>> "termini.dat",
 {n, 1, 10}
]

Clear[T10]

R1List = R0List + T10;
```

Thread::tdlen : Objects of unequal length in {5.48413, 5.66195, ≪8≫, ≪141≫} + {≪1≫} cannot be combined. ≫

```
ListLinePlot[
 Transpose[{Dec, R1List}],
 AxesOrigin → {0, 0}
]
```




```
Do[
 Tlist[n] >>> "T.data",
 {n, 11, 20}
]
```

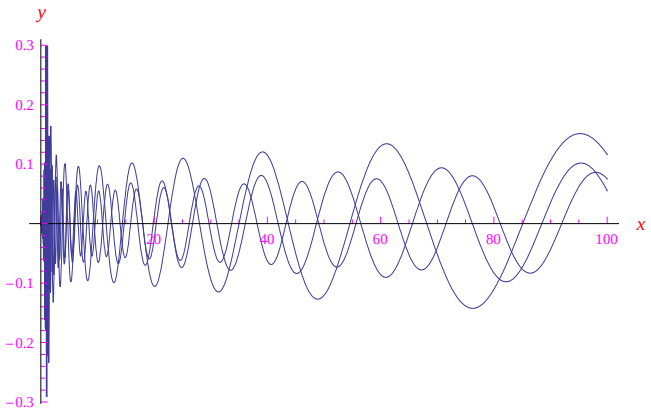
```
Do[
 Tlist[n] >>> "T.data",
 {n, 21, 30}
]
```

```
Do[
 Tlist[n] >>> "T.data",
 {n, 31, 40}
]
```

```
Do[
 Tlist[n] >>> "T.data",
 {n, 41, 50}
]
```

```
correzioni = Import["T.data"];
RplusList = Rplus[partizione];
```

```
Plot[
 Table[
  ψ[x, 1, n],
  {n, 1, 3}
 ],
 {x, 0, 100},
 AxesLabel →
  {
   Style["x", Small, Red],
   Style["y", Small, Red]
  }
]
```

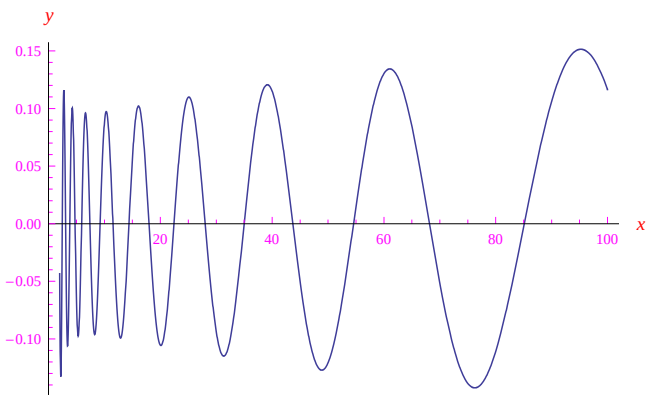



```
Export["multipsi.eps", %]
```

multipsi.eps

```
plotpsi[k_, n_] := Plot[
  ψ[x, k, n],
  {x, 2, 100},
  PlotStyle → Thickness[0.00255],
  AxesLabel →
   {
    Style["x", Small, Red],
    Style["y", Small, Red]
   }
 ]
```

```
plotpsi[1, 1]
```




```
Export["psi5.eps", %]
```

psi5.eps

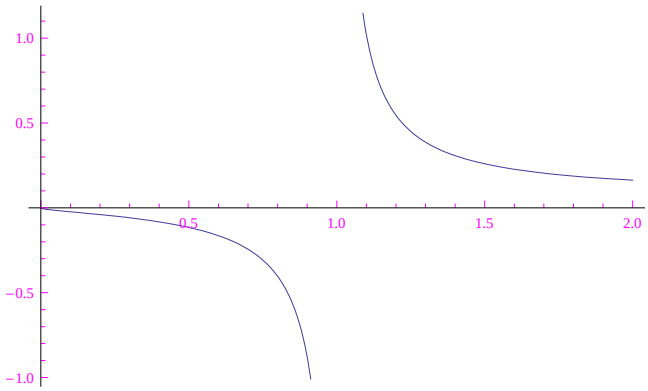
```
A[x_, k_, n_] := k * x^(1/(2*k)) / (Abs[ρ[n]] * Log[x])

lim1[k_, n_] := Limit[
  A[x, k, n],
  x → 0,
  Direction → -1
 ]

lim1[1, 1]
```

0

```
Limit[
 A[x, 1, 1],
 x → 1,
 Direction → 1
]
```

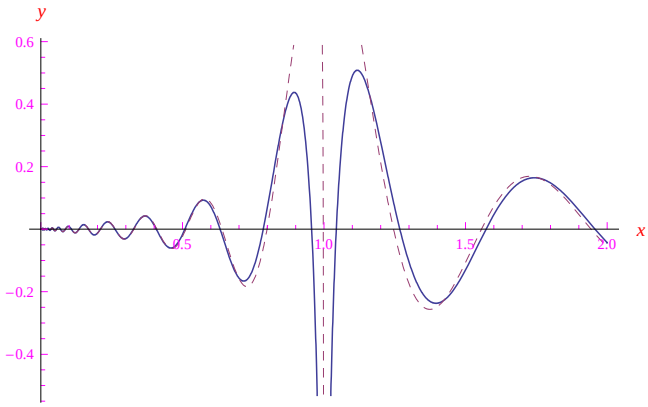
$-\infty$

```
Plot[
 A[x, 2, 2],
 {x, 0, 2},
 Exclusions → x == 1
]
```




```
ψa[x_, k_, n_] := A[x, k, n] * Cos[Im[ρ[n]]/k * Log[x] - Arg[ρ[n]]]
```

```
Plot[
 {
  ψ[x, 1, 1],
  ψa[x, 1, 1]
 },
 {x, 0, 2},
 Exclusions → x == 1,
 PlotStyle → {
   Thickness[0.00255],
   {
    Dashed
   }
  },
 AxesLabel →
  {
   Style["x", Small, Red],
   Style["y", Small, Red]
  }
]
```




```
Export["exapp.eps", %];
```

```
Limit[
 ψ[x, 1, 1],
 x → 1,
 Direction → 1
]
```

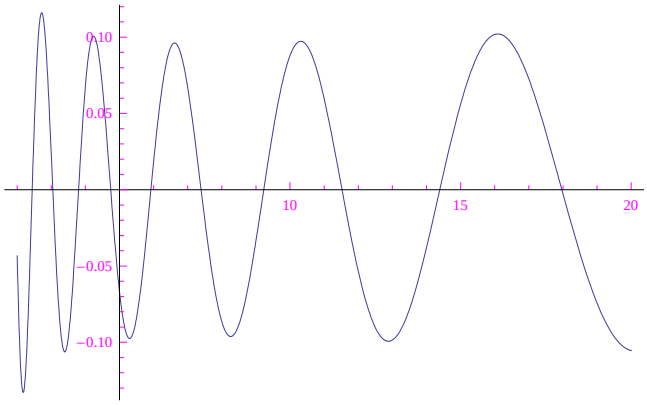
$-\infty$

```
ff[x_] = ψ[x, 1, 1];
```

```
Clear[plotf]
```

```
plotf[a_] := Plot[
  ff[x],
  {x, 2, a}
 ]
```

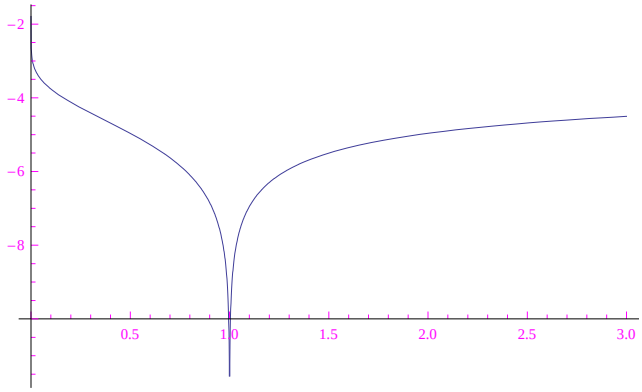
```
plotf[20]
```




```
Limit[
 ψ[x, 1, 1],
 x → 1
]
```

$-\infty$

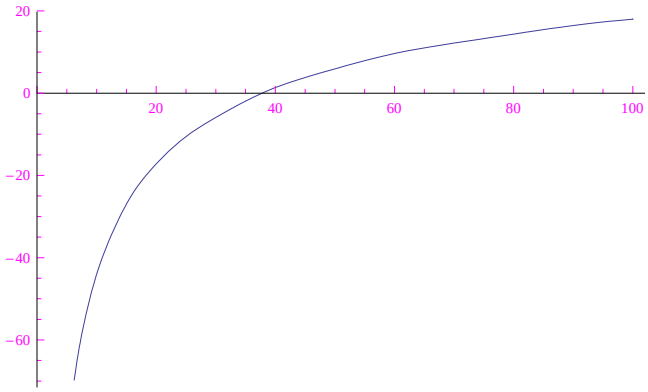
```
Plot[
 ψ[x, 2500, 1],
 {x, 0, 3},
 PlotRange → All
]
```




```
T[x_, n_, N1_] := Sum[
  -2 * MoebiusMu[k] / k * ψ[x, k, n],
  {k, 1, N1}
 ]
```

```
T1[x_, n_, N1_] := Sum[
  ψ[x, k, n],
  {k, 1, N1}
 ]
```

```
Plot[
 T1[x, 1, 154],
 {x, 2, 100}
]
```




```
(AccuracyGoal → 20) [FindRoot[x^2 == x + 1, x]]
```

```
Clear[rho]
```

```
rho = Table[
   N[ZetaZero[k]],
   {k, 50}
  ];
```

```
Select[{1, 2, 4, 7, 6, 2}, # > 2 &]
```

```
{4, 7, 6}
```

```
Select[Range[154], # > 10 &]
```

```
{11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34,
 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56,
 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78,
 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100,
 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116, 117, 118,
 119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129, 130, 131, 132, 133, 134, 135, 136,
 137, 138, 139, 140, 141, 142, 143, 144, 145, 146, 147, 148, 149, 150, 151, 152, 153, 154}
```

```
listaMoebius = Table[
   N[MoebiusMu[k]/k],
   {k, 20}
  ];
```

```
listaMoebius
```

```
{1., -0.5, -0.333333, 0., -0.2, 0.166667, -0.142857, 0., 0., 0.1, -0.0909091,
 0., -0.0769231, 0.0714286, 0.0666667, 0., -0.0588235, 0., -0.0526316, 0.}
```

```
coefficientiMoebius = Select[
  lista, # != 0 &
 ]
```

```
{1., -0.5, -0.333333, -0.2, 0.166667, -0.142857, 0.1,
 -0.0909091, -0.0769231, 0.0714286, 0.0666667, -0.0588235, -0.0526316}
```

```
MoebiusData = -2 * coefficientiMoebius
```

```
{-2., 1., 0.666667, 0.4, -0.333333, 0.285714, -0.2,
 0.181818, 0.153846, -0.142857, -0.133333, 0.117647, 0.105263}
```

$$c[k\_] := \frac{-2 * \text{MoebiusMu}[k]}{k}$$

```
MoebiusMatrix = Table[
   c[k],
   {k, 154}
  ] // N
```

```
{-2., 1., 0.666667, 0., 0.4, -0.333333, 0.285714, 0., 0., -0.2, 0.181818, 0., 0.153846,
 -0.142857, -0.133333, 0., 0.117647, 0., 0.105263, 0., -0.0952381, -0.0909091,
 0.0869565, 0., 0., -0.0769231, 0., 0., 0.0689655, 0.0666667, 0.0645161, 0., -0.0606061,
 -0.0588235, -0.0571429, 0., 0.0540541, -0.0526316, -0.0512821, 0., 0.0487805,
 0.047619, 0.0465116, 0., 0., -0.0434783, 0.0425532, 0., 0., 0., -0.0392157, 0.,
 0.0377358, 0., -0.0363636, 0., -0.0350877, -0.0344828, 0.0338983, 0., 0.0327869,
 -0.0322581, 0., 0., -0.0307692, 0.030303, 0.0298507, 0., -0.0289855, 0.0285714,
 0.028169, 0., 0.0273973, -0.027027, 0., 0., -0.025974, 0.025641, 0.0253165, 0.,
 0., -0.0243902, 0.0240964, 0., -0.0235294, -0.0232558, -0.0229885, 0., 0.0224719,
 0., -0.021978, 0., -0.0215054, -0.0212766, -0.0210526, 0., 0.0206186, 0., 0.,
 0., 0.019802, 0.0196078, 0.0194175, 0., 0.0190476, -0.0188679, 0.0186916, 0.,
 0.0183486, 0.0181818, -0.018018, 0., 0.0176991, 0.0175439, -0.0173913, 0., 0.,
 -0.0169492, -0.0168067, 0., 0., -0.0163934, -0.0162602, 0., 0., 0., 0.015748,
 0., -0.0155039, 0.0153846, 0.0152672, 0., -0.0150376, -0.0149254, 0., 0.,
 0.0145985, 0.0144928, 0.0143885, 0., -0.0141844, -0.0140845, -0.013986, 0.,
 -0.0137931, -0.0136986, 0., 0., 0.0134228, 0., 0.013245, 0., 0., 0.012987}
```

```
Ψ[x_, n_] = Table[
   {ψ[x, k, n]},
   {k, 154}
  ];
```

```
T[x_, n_] = MoebiusMatrix.Ψ[x, n];
```

```
SetAttributes[T, Listable];
```

```
Plot[
 T[x, 1],
 {x, 2, 100}
]
```

```
$Aborted
```

```
A = Table[k, {k, 10}]
```

```
{1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
```

```
B = Table[{k}, {k, 10}]
```

```
{{1}, {2}, {3}, {4}, {5}, {6}, {7}, {8}, {9}, {10}}
```

```
B1 = {{1}, {2}}
```

```
{{1}, {2}}
```

```
A.B
```

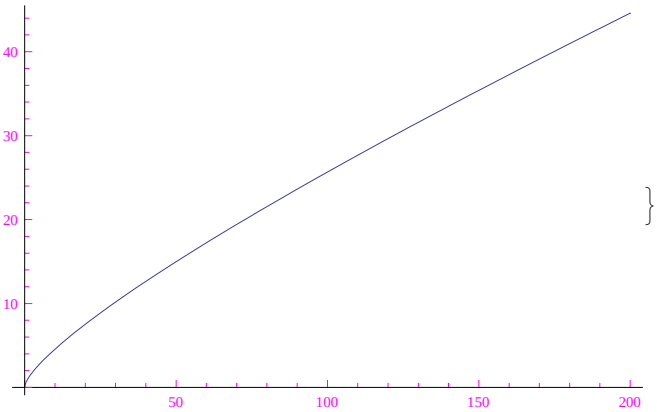
```
{385}
```

```
(*reset: cancellare tutte le variabili*)
```

```
SetOptions[
  {
   Plot,
   ListLinePlot
  },
  TicksStyle -> Directive[
    Hue[5 / 6],
    7
   ]
 ];
```

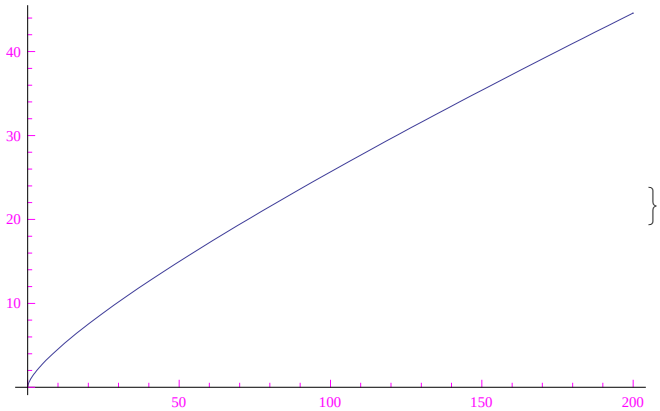
Approssimazione di Riemann

```
R[x_, n_] := 1 + Sum[(Log[x])^k / (k! k * Zeta[k + 1]), {k, n}]

(*begin routine Ilan Vardi*)

lista1[x_, n_] := N[Log[x]]^Range[n]

lista2[n_] := Map[
  (1 / (#! # N[Zeta[# + 1]])) &,
  Range[n]
 ]

Rvardi[x_, n_] := 1 + lista1[x, n].lista2[n]

Plot[
  {Rvardi[x, 100]},
  {x, 0, 200}
 ] // Timing
```

{0.703, 

}

```
Plot[
  {R[x, 100]},
  {x, 0, 200}
 ] // Timing
```

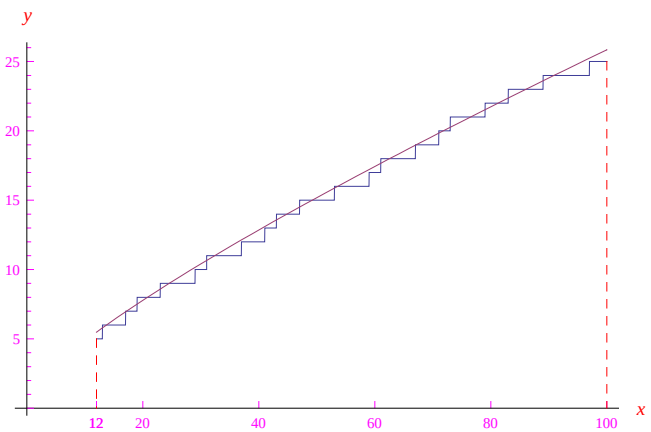
{0.906, 

}

Calcolo del termine $G_{N_0}(x)$

```
g[t_] := 1/(t (t^2 - 1) Log[t])

J[x_] := NIntegrate[
  g[t],
  {t, x, +∞}
 ]

dataJ = Table[
   J[x],
   {x, 2, 200, 0.5}
  ];

Iapp = Interpolation[dataJ]

InterpolatingFunction[{{1., 397.}}, <>]

G[x_, N0_] := Sum[
  (*argomento della sommatoria*)
  MoebiusMu[k]/k * Iapp[x^(1/k)],
  (*indice della sommatoria*)
  {k, N0}]

Clear[G]

(*approssimazione di riesel gohl*)

G[x_] := 1/π ArcTan[π/Log[x]]

Rplus[x_] := R[x, 100] + G[x]

ticks0 = Table[k, {k, 20, 100, 20}];
```

```
Plot[
 {
  PrimePi[x],
  Rplus[x]
 },
 {x, 12, 100},
 AxesOrigin → {0, 0},
 AxesLabel →
  {
   Style["x", Small, Red],
   Style["y", Small, Red]
  },
 Ticks →
  {
   PrependTo[ticks0, 12]
  },
 Epilog →
  {
   Red,
   Dashed,
   Line[{{12, PrimePi[12]}, {12, 0}}],
   Line[{{100, PrimePi[100]}, {100, 0}}]
  }
]
```

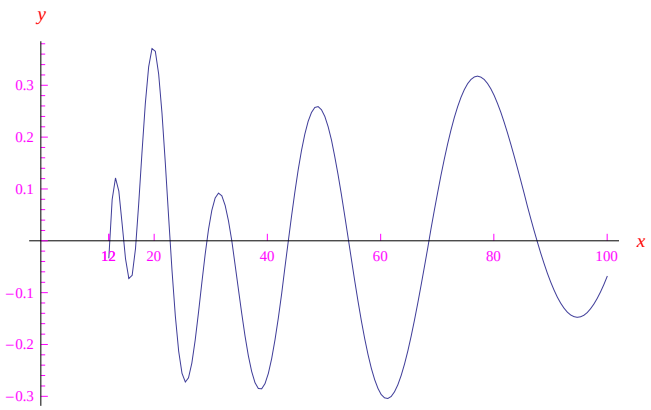



```
(*zeri non banali della zeta di Riemann*)

ρ[n_] := N[ZetaZero[n]]

ψ[x_, k_, n_] := Re[ExpIntegralEi[ρ[n]/k * Log[N[x]]]]

(*coefficienti di Moebius*)

c[k_] := -2 N[MoebiusMu[k]]/k

T[x_, n_, N01_] := Sum[c[k] * ψ[x, k, n], {k, 1, N01}]

Clear[T]

(*genera la lista dei coefficienti di Möbius*)
```

```
listaMoebius = Table[
   -2 MoebiusMu[k] // N / k,
   {k, 154}
  ];

(*ridefinisce gli indici della somma sui coefficienti non nulli*)

IndiciMoebius = Select[
   Range[154],
   listaMoebius[[#]] ≠ 0 &
  ];

(*calcola i coefficienti di Möbius non nulli*)

CoefficientiMoebius = listaMoebius[[IndiciMoebius]];

psi[x_, n_] = Re[ExpIntegralEi[ρ[n] / IndiciMoebius * Log[N[x]]]];

T[x_, n_] := CoefficientiMoebius . psi[x, n]

Dec = Range[
   (*punto iniziale*)
   12.,
   (*punto finale*)
   100,
   (*step*)
   88. / 150
  ];

Tlist[n_] := T[Dec, n];

M[n_] := {
  (*prima riga*)
  Dec,
  (*seconda riga*)
  Tlist[n]
 }

Clear[ticksx]

ticksx = Table[k, {k, 20, 100, 20}];

ticksx

{20, 40, 60, 80, 100}

AppendTo[ticksx, 12]

{20, 40, 60, 80, 100, 12}
```

```
plotT[n_] := ListLinePlot[
  Transpose[M[n]],
  AxesOrigin → {0, 0},
  Ticks →
   {
    AppendTo[ticksx, 12]
   },
  AxesLabel →
   {
    Style["x", Small, Red],
    Style["y", Small, Red]
   }
 ]
```

```
plotT[1]
```




```
Export["T1plot.eps", %];
```

```
a = 1;
```

```
a >> "test1.dat"
```

```
a = 2;
```

```
a >>> "test1.dat"
```

```
FilePrint["test1.dat"]
```

```
1
2
```

```
Do[
 a = a / 2 >>> "test1.dat",
 {5}
]
```

```
FilePrint["test1.dat"]
```

```
1
2
1
1/2
1/4
1/8
1/16
```

```
Do[
 Tlist[n] >>> "T.data",
 {n, 1, 10}
]
```

```
Do[
 Tlist[n] >>> "T.data",
 {n, 11, 20}
]
```

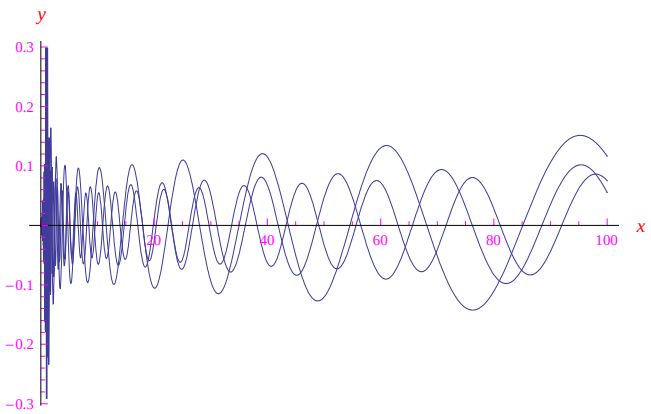
```
Do[
 Tlist[n] >>> "T.data",
 {n, 21, 30}
]
```

```
Do[
 Tlist[n] >>> "T.data",
 {n, 31, 40}
]
```

```
Do[
 Tlist[n] >>> "T.data",
 {n, 41, 50}
]
```

```
correzioni = Import["T.data"];
RplusList = Rplus[partizione];
```

```
Plot[
 Table[
  ψ[x, 1, n],
  {n, 1, 3}
 ],
 {x, 0, 100},
 AxesLabel →
  {
   Style["x", Small, Red],
   Style["y", Small, Red]
  }
]
```

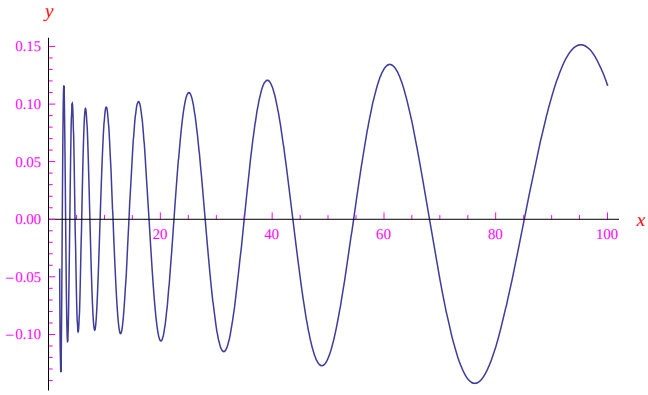



```
Export["multipsi.eps", %]
```

multipsi.eps

```
plotpsi[k_, n_] := Plot[
  ψ[x, k, n],
  {x, 2, 100},
  PlotStyle → Thickness[0.00255],
  AxesLabel →
   {
    Style["x", Small, Red],
    Style["y", Small, Red]
   }
 ]
```

```
plotpsi[1, 1]
```




```
Export["psi5.eps", %]
```

psi5.eps

$$A[x\_, k\_, n\_] := \frac{k * x^{\frac{1}{2*k}}}{\text{Abs}[\rho[n]] * \text{Log}[x]}$$

```
lim1[k_, n_] := Limit[
  A[x, k, n],
  x → 0,
  Direction → -1
 ]
```

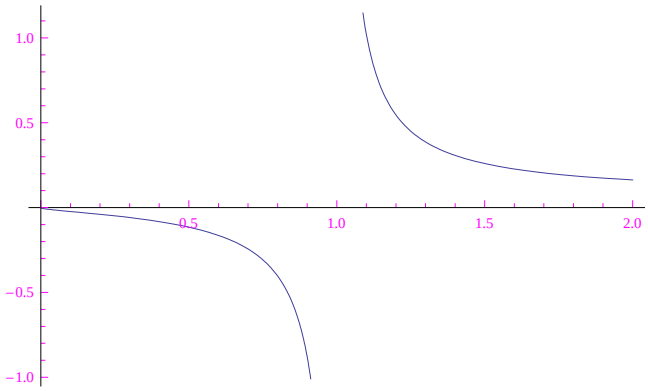
```
lim1[1, 1]
```

0

```
Limit[
 A[x, 1, 1],
 x → 1,
 Direction → 1
]
```

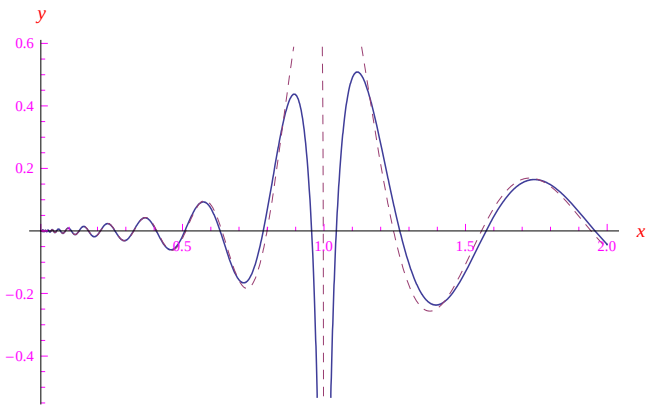
$-\infty$

```
Plot[
 A[x, 2, 2],
 {x, 0, 2},
 Exclusions → x == 1
]
```




```
ψa[x_, k_, n_] := A[x, k, n] * Cos[Im[ρ[n]]/k * Log[x] - Arg[ρ[n]]]
```

```
Plot[
 {
  ψ[x, 1, 1],
  ψa[x, 1, 1]
 },
 {x, 0, 2},
 Exclusions → x == 1,
 PlotStyle → {
   Thickness[0.00255],
   {
    Dashed
   }
  },
 AxesLabel →
  {
   Style["x", Small, Red],
   Style["y", Small, Red]
  }
]
```




```
Export["exapp.eps", %];
```

```
Limit[
 ψ[x, 1, 1],
 x → 1,
 Direction → 1
]
```

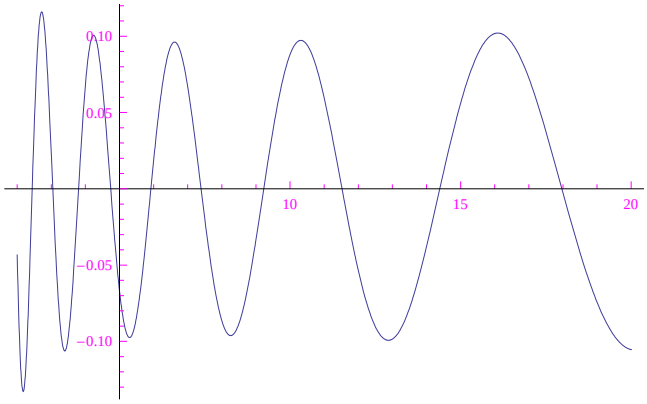
$-\infty$

```
ff[x_] = ψ[x, 1, 1];
```

```
Clear[plotf]
```

```
plotf[a_] := Plot[
  ff[x],
  {x, 2, a}
 ]
```

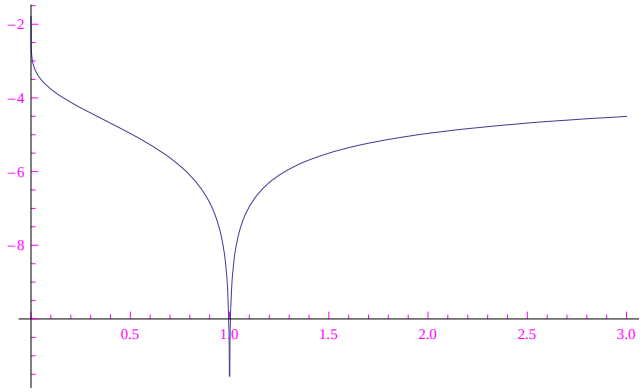
```
plotf[20]
```




```
Limit[
 ψ[x, 1, 1],
 x → 1
]
```

$-\infty$

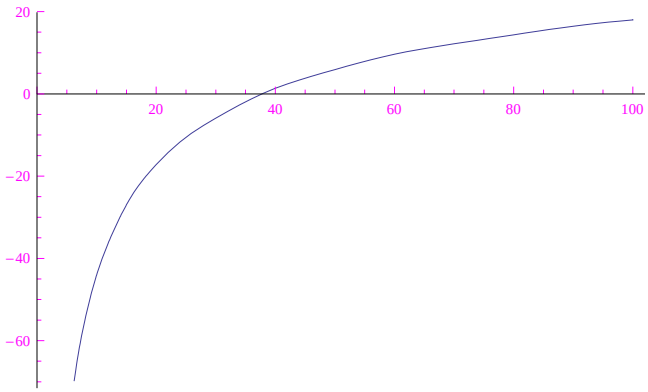
```
Plot[
 ψ[x, 2500, 1],
 {x, 0, 3},
 PlotRange → All
]
```



```
T[x_, n_, N1_] := Sum[
  -2 * MoebiusMu[k]/k * ψ[x, k, n],
  {k, 1, N1}
 ]
```

```
T1[x_, n_, N1_] := Sum[
  ψ[x, k, n],
  {k, 1, N1}
 ]
```

```
Plot[
 T1[x, 1, 154],
 {x, 2, 100}
]
```




```
(AccuracyGoal → 20) [FindRoot[x^2 == x + 1, x]]
```

```
Clear[rho]
```

```
rho = Table[
   N[ZetaZero[k]],
   {k, 50}
  ];
```

```
Select[{1, 2, 4, 7, 6, 2}, # > 2 &]
```

```
{4, 7, 6}
```

```
Select[Range[154], # > 10 &]
```

```
{11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34,
 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56,
 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78,
 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100,
 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116, 117, 118,
 119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129, 130, 131, 132, 133, 134, 135, 136,
 137, 138, 139, 140, 141, 142, 143, 144, 145, 146, 147, 148, 149, 150, 151, 152, 153, 154}
```

```
listaMoebius = Table[
   N[MoebiusMu[k]/k],
   {k, 20}
  ];
```

```
listaMoebius
```

```
{1., -0.5, -0.333333, 0., -0.2, 0.166667, -0.142857, 0., 0., 0.1, -0.0909091,
 0., -0.0769231, 0.0714286, 0.0666667, 0., -0.0588235, 0., -0.0526316, 0.}
```

```
coefficientiMoebius = Select[
  lista, # != 0 &
 ]
```

```
{1., -0.5, -0.333333, -0.2, 0.166667, -0.142857, 0.1,
 -0.0909091, -0.0769231, 0.0714286, 0.0666667, -0.0588235, -0.0526316}
```

```
MoebiusData = -2 * coefficientiMoebius
```

```
{-2., 1., 0.666667, 0.4, -0.333333, 0.285714, -0.2,
 0.181818, 0.153846, -0.142857, -0.133333, 0.117647, 0.105263}
```

$$c[k\_] := \frac{-2 * \text{MoebiusMu}[k]}{k}$$

```
MoebiusMatrix = Table[
   c[k],
   {k, 154}
  ] // N
```

```
{-2., 1., 0.666667, 0., 0.4, -0.333333, 0.285714, 0., 0., -0.2, 0.181818, 0., 0.153846,
 -0.142857, -0.133333, 0., 0.117647, 0., 0.105263, 0., -0.0952381, -0.0909091,
 0.0869565, 0., 0., -0.0769231, 0., 0., 0.0689655, 0.0666667, 0.0645161, 0., -0.0606061,
 -0.0588235, -0.0571429, 0., 0.0540541, -0.0526316, -0.0512821, 0., 0.0487805,
 0.047619, 0.0465116, 0., 0., -0.0434783, 0.0425532, 0., 0., 0., -0.0392157, 0.,
 0.0377358, 0., -0.0363636, 0., -0.0350877, -0.0344828, 0.0338983, 0., 0.0327869,
 -0.0322581, 0., 0., -0.0307692, 0.030303, 0.0298507, 0., -0.0289855, 0.0285714,
 0.028169, 0., 0.0273973, -0.027027, 0., 0., -0.025974, 0.025641, 0.0253165, 0.,
 0., -0.0243902, 0.0240964, 0., -0.0235294, -0.0232558, -0.0229885, 0., 0.0224719,
 0., -0.021978, 0., -0.0215054, -0.0212766, -0.0210526, 0., 0.0206186, 0., 0.,
 0., 0.019802, 0.0196078, 0.0194175, 0., 0.0190476, -0.0188679, 0.0186916, 0.,
 0.0183486, 0.0181818, -0.018018, 0., 0.0176991, 0.0175439, -0.0173913, 0., 0.,
 -0.0169492, -0.0168067, 0., 0., -0.0163934, -0.0162602, 0., 0., 0., 0.015748,
 0., -0.0155039, 0.0153846, 0.0152672, 0., -0.0150376, -0.0149254, 0., 0.,
 0.0145985, 0.0144928, 0.0143885, 0., -0.0141844, -0.0140845, -0.013986, 0.,
 -0.0137931, -0.0136986, 0., 0., 0.0134228, 0., 0.013245, 0., 0., 0.012987}
```

```
Ψ[x_, n_] = Table[
   {ψ[x, k, n]},
   {k, 154}
  ];
```

```
T[x_, n_] = MoebiusMatrix.Ψ[x, n];
```

```
SetAttributes[T, Listable];
```

```
Plot[
 T[x, 1],
 {x, 2, 100}
]
```

```
$Aborted
```

```
A = Table[k, {k, 10}]
```

```
{1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
```

```
B = Table[{k}, {k, 10}]
```

```
{{1}, {2}, {3}, {4}, {5}, {6}, {7}, {8}, {9}, {10}}
```

```
B1 = {{1}, {2}}
```

```
{{1}, {2}}
```

```
A.B
```

```
{385}
```