Capitolo 5

■ 5.1 Proprieta' delle funzioni predefinite

Se vogliamo informazioni su una funzione **f** - predefinita o definita da noi - usiamo il comando **?f.** Il comando **??f** descrive piu' in dettaglio la definizione e le proprieta' di **f**. Il comando **Attributes[f]** mostra le proprieta' (o attributi) della funzione **f**. Per esempio per il logaritmo

```
Attributes[Log]
```

```
{Listable, NumericFunction, Protected}
```

Listable vuol dire che Log[{a1,a2,..}] fornisce come risultato {Log[a1],Log[a2],...};

NumericFunction vuol dire che Log[a] e' un valore numerico se a e' un numero;

Protected vuol dire che la definizione di Log non puo' essere modificata.

Il comando **NumericQ[expr]** ha valore *True* se *expr* e' una quantita' numerica; *False* altrimenti.

```
NumericQ[Log[Pi]]
```

Tutte le **Built-in functions** sono protette ovvero hanno la proprieta' (**Attribute**) **Protected.** E' possibile togliere la protezione e "forzare" la definizione. Per esempio la funzione log(5xy) non viene scritta da *Mathematica* come log5 +log x+log y. Possiamo pero' imporre questa regola, insieme a quella $\log x^y = y \log x$

```
Log[5 x y]
Log[(2 x)^(4 y)]
```

```
Unprotect[Log]
Log[a_ b_] := Log[a] + Log[b]
Log[x_^y_] := y Log[x]
Protect[Log]
```

```
Log[5 x y]
Log[(2 x)^(4 y)]
```

Per il seguito, ripuliamo la funzione **Log** da queste nuove definizioni.

```
Unprotect[Log]
Clear[Log]
Protect[Log]
```

Questa versatilita' di *Mathematica* puo' essere pericolosa!. Se definiamo delle regole sbagliate, *Mathematica* fornira' risposte errate.

```
Unprotect[Log]
Log[7] = 2
Log[7] + Log[2]
Clear[Log]
Protect[Log]
```

In alcuni casi potremmo avere bisogno di modificare le regole predefinite in *Mathematica*.

```
Log[Exp[x]]
Unprotect[Log]
Log[Exp[x_]] := x
Protect[Log]
Log[Exp[x]]
```

Per esempio *Mathematica* semplifica

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

Modifichiamo la regola di calcolo nel seguente modo

```
Unprotect[Exp]
Exp[Log[x_]] := explog[x]
Exp[Log[x]]
Clear[Exp]
Protect[Exp]
```

La funzione **ArcSin** assume, per *default*, valori nell'intervallo (-π/2,-π/2).

```
ArcSin[1]
```

Modifichiamo il valore che la funzione **ArcSin** assume in 1

```
Unprotect[ArcSin]
ArcSin[1] = 5 Pi / 2;
ArcSin[1]
Clear[ArcSin]
Protect[ArcSin]
```

Consideriamo ora la funzione predefinita **Plus[x,y,...]**, che somma i suoi argomenti

```
?? Plus
```

```
Attributes[Plus]
```

```
{Flat, Listable, NumericFunction, OneIdentity, Orderless, Protected}
```

Flat vuol dire che l'argomento gode della proprieta' associativa cioe' **Plus[Plus[a,b],c]=Plus[a,b,c]** (altre funzioni **Flat** sono **Times** e **Dot**);

OneIdentity vuol dire che **Plus[a]**, **Plus[Plus[a]]** , **Plus[Plus[Plus[a]]]** etc. sono tutti equivalenti ad a;

Orderless vuol dire che non conta l'ordine in cui compaiono gli argomenti cioe' **Plus[a,b]=Plus[b,a]**, **Plus[a,b,c]=-Plus[b,c,a]**;

Quando definiamo una funzione possiamo assegnargli delle proprieta' con il comando **Attributes[f]={attr1,attr2,...}** oppure aggiungere degli attibuti a quelli che la funzione possiede con il comando **SetAttributes[f,attr].**

Per esempio definiamo la funzione **g** e proteggiamo la sua definizione che non potra' essere modificata:

```
Clear[g]
g[x_] := funz[x]
Attributes[g] = Protected
```

```
g[x_] := x
```

```
?? g
```

Aggiungiamo la proprieta' **Listable**

```
g[{a, b, c}]
SetAttributes[g, Listable]
g[{a, b, c}]
?? g
```

Aggiungo la proprieta' **ReadProtected** cioe' la sua definizione non puo' essere letta

```
SetAttributes[g, ReadProtected]

?? g
```

Per l'elenco completo degli **Attributes** rimandiamo al manuale di *Mathematica.*

Esempio. Consideriamo le funzioni evenq, oddq, primepi, primeq definite precedentemente che ricopiano, rispettivamente, le funzioni predefinite **EvenQ**, **OddQ**, **PrimePi**, **PrimeQ** e assegnamogli le proprieta' delle funzioni che stiamo ricopiando:

```
Attributes[EvenQ]
Attributes[evenq] = %
Attributes[OddQ]
Attributes[oddq] = %
Attributes[PrimePi]
Attributes[primepi] = %
Attributes[PrimeQ]
Attributes[primeq] = %
```

Il comando **ClearAll[f]** rimuove da f sia i valori che gli **Attributes** laddove **Clear[f]** rimuove solo i valori ma non gli **Attributes**. Sia **Clear** che **ClearAll** non ripuliscono le funzioni con l'attributo **Protected.**

```
Unprotect[g]
Clear[g]
?? g

ClearAll[g]
?? g
```

Esempio. Definire una funzione reale - **radice(x)** - che ricopia la funzione predefinita **Sqrt** cioe' che abbia gli stessi attributi e le stesse proprieta'.

```
Attributes[Sqrt]
{Listable, NumericFunction, Protected}

radice[x_ /; x > 0] := N[t /. (Solve[t^2 == x, t])][[2]]
Attributes[radice] = {Listable, NumericFunction, Protected}

radice2[x_ /; x > 0] := t /. (Solve[t^2 == x, t])[[2]]
Attributes[radice2] = {Listable, NumericFunction, Protected}

radice3[x_] := t /. (Solve[t^2 == x, t])[[2]]
Attributes[radice3] = {Listable, NumericFunction, Protected}
```

Esempio. Proviamo a definire una funzione che chiameremo **lg** che ha le stesse proprieta' e gli stessi attributi della funzione **Log.** La nuova funzione **lg** deve soddisfare le seguenti tre regole

lg[x y]=lg[x]+lg[y]

lg[x^y]=ylg[x]

y=lg[x] e' soluzione dell'equazione Exp[y]=x se x e' reale positivo

Per definire quest'ultima regola usiamo il comando **Solve[eq1==eq2,t]** che risolve l'equazione rispetto a t.

```
Clear[x, y, t]
lg[x_ y_] := lg[x] + lg[y]
lg[x_^y_] := y lg[x]
lg[x_ /; (x > 0)] := N[t /. (Solve[Exp[t] == x, t])][[1]]
```

e deve avere i seguenti

```
Attributes[lg] = {Listable, NumericFunction, Protected};
? lg
```

L'uso di **Solve** genera un messaggio di errore. Per esempio

```
lg[2]
```

Per non visualizzare il messaggio di avvertimento possiamo dare il comando

```
Off[Solve::ifun]

lg[a b]
lg[b a]
lg[a^3]
lg[{a, b, c}]
lg[E]
lg[Pi]
lg[-2]
```

Proviamo ad aggiungere a lg la proprieta' **ReadProtected**

```
SetAttributes[lg, ReadProtected]
? lg
```

Per ripristinare il messaggio di avvertimento

```
On[Solve::ifun]
lg[2]

ClearAttributes[lg, Protected]
ClearAll[lg]
```

■ 5.2 Definire funzioni con argomenti opzionali

Quando definiamo **f[x_,y_]:=expr** la funzione **f** deve sempre essere richiamata con due argomenti. Se richiamiamo f con un solo argomento *Mathematica* mi avverte con un messaggio di errore.

Si possono definire funzioni dipendenti da un certo numero di argomenti che, se non indicati esplicitamente, assumono un prestabilito valore di *default*. La notazione **x_:v** indica una espressione in cui, se non indicato, **x** assume il valore di *default* **v**. Per esempio **f[x_,k_:v]:=expr** e' una tipica funzione in cui il secondo argomento **k** e' opzionale. Si puo' richiamare **f** con uno o due argomenti. Quando il secondo argomento e' omesso *Mathematica* assume **k=v**. Per esempio

```
f1[x_: 1] := 2 x; {f1[], f1[y]}
f2[x_, n_: Infinity] := x^2 - 1 / n
{f2[x], f2[x, y]}
```

Definiamo una funzione con due argomenti opzionali e che quindi puo' essere richiamata con uno, due oppure tre argomer

```
g[x_, n1_: 1, n2_: 2] := (Sin[x] / n1) + Exp[n2]
{g[x], g[x, y], g[x, y, z]}
```

Mathematica assume che gli argomenti mancanti siano contati dalla fine. Quindi in **g[x,y]** e' assunto **n1=y** e **n2=2** .

```
h[x_: x0, y_, z_: z0] := Sin[x] + Cos[y] + z
{h[x], h[x, y], h[x, y, z]}
```

Definiamo una funzione il cui argomento e' una funzione lineare in x

```
Clear[f1, f2]
f1[a_ + b_ x] := {a, b}
{f1[1 + 2 x], f1[1 + x], f1[2 x]}
```

Per calcolare f1[1+x] *Mathematica* non interpreta x=1*x (cioe' b→1) oppure per calcolare f1[2 x] *Mathematica* non interpreta 2 x=0+2 x (cioe' a→0). Se usiamo la definizione con gli argomenti opzionali allora le cose funzionano

```
f2[a_: 0 + (b_: 1) x] := {a, b}
{f2[1 + 2 x], f2[1 + x], f2[2 x]}
```

Le piu' comuni funzioni hanno valori di *default* predefiniti. In questi casi non e' necessario indicare **x_:v** ma semplicemente la notazione **x_.** . Nella somma cioe' **x+a_.** , se omesso **a** assume il valore 0; nel prodotto **b_. x** se omesso **b** assume il valore 1. Nell'elevamento a potenza cioe' **x^c_.** se omesso **c** assume il valore 1. Quindi **a_.** indica un'espressione con il valore di *default* assegnato secondo la regola: **x+a_.** e' equivalente a scrivere **x+a_:0** laddove **b_. x** equivale a scrivere **b_:1 x** mentre **x^c_.** equivale a scrivere **x^c_:1** .

Quindi le funzioni **h1** e **h2** il cui argomento e' una funzione lineare di **x**, sono equivalenti

```
h1[a_: 0 + (b_: 1) x_: 0] := {a, b}
h2[a_. + b_. x] := {a, b}
{h1[1 + x], h1[x], h1[2 x], h1[1 + y]}
{h2[1 + x], h2[x], h2[2 x], h2[1 + y]}
```

Definiamo la seguente funzione

```
g1[x_^n_] := p[x, n]
{g1[a^2], g1[(a + b)^2], g1[a], g1[a + b]}
```

in questo caso g[a] e g[a+b] non vengono interpretati da *Mathematica* come g[a^1] e g[(a+b)^1]. Le cose vanno bene se invece definisco

```
g2[x_^n_.] := p[x, n]
{g2[a^2], g2[(a + b)^2], g2[a], g2[a + b]}
```

Definiamo la funzione **f** il cui argomento e' un polinomio di grado due nella variabile x con termine lineare e quadratico non nullo

```
f[a_. + b_. x + c_. x^2] := {a, b, c}
{f[1 + x + x^2], f[2 x + x^2], f[x + x^2], f[1], f[3 + x]}
```

■ 5.3 Funzioni Pure

Consideriamo la funzione

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

In realta' non e' importante che la funzione si chiami **f** o che l'argomento, che verra' di volta in volta sostituito, si chiami **x**. L'unica cosa che importa e' la regola "eleva al cubo". In alcuni casi puo' essere utile definire funzioni usando una sintassi leggermente diversa da quella usata fino ad ora cioe' usando le *funzioni pure*. Cos'e' una *funzione pura*?

Per esempio **Cos** e' una funzione pura laddove **Cos[a]** calcola la funzione pura in **a** (**Cos[a]** si puo' anche indicare con **Cos@a**).

Un primo modo e' indicare la funzione pura con **Function[x,body]** se funzione di una variabile e **Function[{x,y,...},body]** se funzione di piu' variabili. Il *body* e' analogo a quello nell'usuale definizione di funzione. Quindi la funzione

```
Cos[x] + Sin[x]
```

diventa

```
Function[x,Cos[x]+Sin[x]]
```

la calcolo in t , poi in $\sqrt{\pi}$ e poi in a+b

```
Function[x,Cos[x]+Sin[x]][t]
Function[x,Cos[x]+Sin[x]][Sqrt[Pi]]
Function[x,Cos[x]+Sin[x]][a+b]
```

Questo modo di definire funzioni puo' essere utile se, ad esempio, la funzione deve essere usata solo una volta e non e' necessario darle un nome. Se serve richiamarla piu' volte si puo' definire nel modo usuale

```
Clear[f]
f := Function[x, Cos[x] + Sin[x]]
? f
```

e richiamarla poi con **f[x]**

```
f[5]
f[x]
```

Definiamo la funzione pura di due variabili cos(x)+cos(y)

```
g := Function[{x, y}, Cos[x] + Cos[y]]

g[x, y]
```

Un altro modo di definire *funzioni pure*, piu' semplice sia da scrivere che da leggere, e' quello di definire la funzione dove l'argomento e' sostituito dal simbolo #. Se ci sono piu' variabili , queste sono indicate con #1,#2 (**Slot[1]**, **Slot[2]**) e cosi' via. La fine della definizione e' seguito da **&** cioe' *body&*.Per esempio la funzione x^2 e' definita come

```
#^2&
```

la calcolo in 5

```
#^2&[5]
```

e la disegno

```
Plot[#^2 &[x], {x, -2, 2}]
```

la funzione $x^2 + y^2$ e' cosi' definita

```
#1^2+#2^2 &

%[x,y]
```

La funzione di tre variabili $\cos^2(x) + \sin^2(y) + z$ si definisce come

```
f := Cos[#1]^2 + Sin[#2]^2 + #3 &

f[x, y, z]
```

e la funzione $x - 1 + y^4$

```
g := #1 - 1 + #2^4 &

g[x, y]
```

Vediamo diversi modi equivalenti di definire la funzione cos(x) sin(x)

```
f1[x_] := Cos[x] Sin[x]
f2 := Cos[#] Sin[#] &
f3 := Function[x, Cos[x] Sin[x]]

{f1[x], f2[x], f3[x]}

Plot[f2[x], {x, 0, Pi}]
```

e diversi modi di definire la funzione cos(x)sin(y)

```
g1[x_, y_] := Cos[x] Sin[y]
g2 := Cos[#1] Sin[#2] &
g3 := Function[{x, y}, Cos[x] Sin[y]]

{g1[x, y], g2[x, y], g3[x, y]}
```

Vedremo che le *funzioni pure* si possono comporre, derivare, integrare etc.

La funzione **Composition[f,g,...]** rappresenta la composizione delle funzioni f ,g... cioe' f∘g∘...

```
Composition[f, g, h][x]

new := Composition[Sin, Cos, Log](*e' una funzione pura*)
new[x]

new2 := Composition[Sin[#] &, Cos[#] &, Log[#] &]
new2[x]

Clear[f, g, k]
f := #^2 - 3 + Exp[-#] &
g := Sin[#]^2 &
k := Cos[#]^2 &

h := Composition[f, g]
h
h[x]
```

```
Composition[g, f, k][x]
```

Consideriamo il comando **Map[f,{a,b,...}],** che applica la *funzione pura* f separatamente ad ogni elemento della lista **{a,b,...}** e fornisce come output la lista **{f[a],f[b],...}**. **Map** ha anche una forma abbreviata: invece di scrivere **Map[f, {a,b,...}]** si puo' scrivere **f/@{a,b,...}**

```
Clear[f]
Map[f,{a,b,c}]
Map[f,{{a,b},{c,d}}]
Map[Sqrt,{a,b,c}]
Map[Cos,{a,b,c}]
Map[Function[x,Cos[x]],{a,b,c}]
Map[Cos[#]&,{a,b,c}]
```

Il comando **Nest[f,x,n]** applica la *funzione pura* f n volte su x cioe'

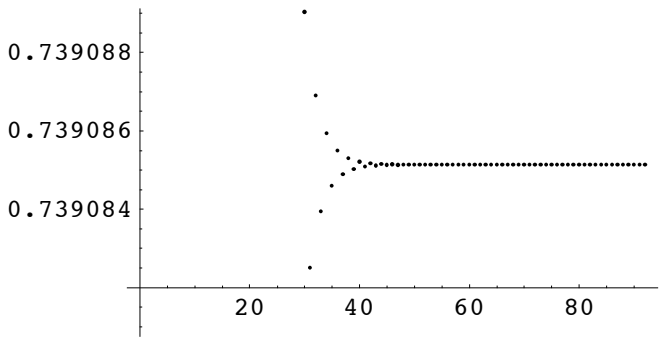
```
Nest[f,x,4]
Nest[Cos,x,4]


f[x_]:=1/(x+1)
Nest[f,x,3]
Nest[Function[x,1/(x+1)],x,3]
Nest[1/(1+#)&,x,3]
```

Il comando **FixedPoint[f,x0]**, parte da x0, applica ripetutamente **f** fino a che il risultato tra due calcoli successivi non cambia. **FixedPointList[f,x0]** visualizza la successione dei valori {x0,f[x0],f[f[x0]],...}

```
FixedPointList[Cos, 0.2]
FixedPoint[Cos, 0.2]
```

che e' un punto fisso del coseno. In pratica *Mathematica* cerca la soluzione dell'equazione f(x)=x basandosi sul teorema del punto fisso. Non e' sempre detto che il procedimento converga e che quindi **FixedPoint** termini. **FixedPoint[f,x0, n]** applica ripetutamente la funzione **f** n volte (per evitare loop!)

```
FixedPoint[Cos, 0.2, 10]
Table[FixedPoint[Cos, 0.2, n], {n, 1, 40}]
ListPlot[FixedPointList[Cos, 0.2]]
```




```
- Graphics -
```

Esercizio. Usando i comandi **Map** e **Composition**, definire una funzione sulle liste di lunghezza 4 tale che f({x,y,z,w})={cos(sin(x)),cos(sin(y)),cos(sin(z)),cos(sin(t))}.

```
f[x_List /; Length[x] == 4] := Map[Composition[Cos, Sin], x]
```

■ 5.4 Limiti per funzione di una variabile

Mathematica e' in grado di calcolare alcuni limiti con il comando **Limit[expr,x->x0]**. Per esempio

```
ClearAll["Global`*"]
Limit[Sin[x], x → Pi / 2]
Limit[(2 x + 1) / (x - 3), x → Infinity]
Limit[1 / x, x → Infinity]
Limit[(1 + x) ^ (1 / x), x → 0]
```

Mathematica per *default* calcola il limite da destra $x \to x_0^+$. Per esempio

```
Limit[Exp[1 / x], x → 0]
Limit[x Log[x], x → 0]
Limit[Sign[x], x → 0]
```

Per richiedere esplicitamente che calcoli il limite da destra o da sinistra bisogna specificare, rispettivamente, l'opzione **Direction→-1** o **Direction→1**

```
Limit[Exp[1 / x], x → 0, Direction -> 1]
Limit[Sign[x], x → 0, Direction → 1]
```

Nel prossimo esempio *Mathematica* ci dice che il limite e' compreso tra -1 e 1...anche se in realta' il limite non esiste.

```
Limit[Sin[1 / x], x → 0]
Plot[Sin[1 / x], {x, -1, 1}]

Limit[x Cos[x], x → Infinity]
```

Esercizio 1. Calcolare i limiti per x $\to 0$, delle funzioni $\frac{\sin x}{x}$, $\frac{1-\cos x}{x^2}$, $\frac{e^x-1}{x}$, $\frac{\log(1+x)}{x}$.

Esercizo 2. Calcolare il limite per x $\to 0$ da sinistra e da destra delle funzioni $\frac{1}{x}$ e $\frac{e^{1/x}\,\mathrm{tg}(x)}{\sin(2x)}$.

Esempio 1. Costruire una funzione test che calcoli limite sinistro e limite destro di una funzione in un punto e fornisca come risultato il valore del limite, se questo esiste, altrimenti mandi un messaggio di avvertimento .

```
test[f_, x_, x0_] := Module[{l1, l2},
  l1 = Limit[f, x → x0];
  l2 = Limit[f, x → x0, Direction → 1];
  If[l1 == l2,
   Print["Il limite in ", x0, " e' ", l1],
   Print["I limiti da destra e da sinistra in ", x0, " sono diversi"]]
 ]

test[Sign[x], x, 0]
test[Exp[1 / x], x, 0]
test[Sin[x] / x, x, 0]
test[Exp[x] Tan[x] / Sin[2 x], x, 0]
test[(x^2 - 16) / (2 - Sqrt[x]), x, 4]
test[((a + #)^2 - a^2) / # &[x], x, 0]
```

■ 5.5 Derivate

Il comando per calcolare la derivata prima di f(x) rispetto a x e' **D[f[x],x]** oppure **f'[x]**. Per calcolare la derivata di ordine n il comando e' **D[f[x],{x,n}]** oppure **D[f[x],{x,...,x}]**. Se f e' una funzione di piu' variabili, f(x_1 ,...,x_n), la derivata parziale si calcola con il comando **D[f[x_1 ,...,x_n],{x_1 ,n1},..,{x_n ,nn}].**

```
ClearAll["Global`*"]
f[x_] := Cos[x]
D[f[x], {x, 1}]
D[f[x], {x, 2}]
D[f[x], {x, 3}]

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

g[x_, y_] := Sin[x^3 + 3 y]
D[g[x, y], {x, 2}, {y, 1}]
∂x,x,y g[x, y]
```

Mathematica deriva anche espressioni con parametro oppure funzioni composte:

```
D[x^n, x]
D[g[x]^2, x]
D[x^2 + y[x]^3, x]
D[h[x^2, y^2], x]
```

Le funzioni pure si derivano nello stesso modo. Per esempio

```
Clear[f]
f := #^2 + Log[#] - Sin[#] &

f' (*e' una funzione pura*)
f'[x]
D[f[x], x]
```

Si possono definire le derivate di una funzione di una variabile come si definiscono usualmente le funzioni. Definiamo la derivata di f(x) uguale alla funzione fprime(x) cioe'

```
Clear[f]
f'[x_] := fprime[x]
```

Mathematica usa questa regola, per esempio, per calcolare

```
D[f[x^3], x]
```

Deriviamo ancora

```
D[%, x]
```

• Le funzioni pure possono essere utili per definire operatori differenziali. Per esempio l'operatore $\frac{\partial}{\partial x}$ si indica con

```
operator=(D[#,x]&)
operator[g[x]]
operator[Sin[x]]
```

Definiamo l'operatore I + $\frac{\partial}{\partial x}$ si indica con

```
operator2=Identity+(D[#,x]&)
```

dove **Identity** e' l'operatore identita'. Applichiamo l'operatore a x^3

```
operator2[g[x]]
operator2[x^3]
```

```
(Identity + (∂x #1 &))[g[x]]
```

```
(Identity + (∂x #1 &))[x^3]
```

Mathematica non applica automaticamente all'espressione le due parti separate dell'operatore . Per fare questo si usa **Through[expr,Plus]**

```
Through[operator2[g[x]], Plus]
Through[operator2[x^3], Plus]
```

Definiamo l'operatore di Laplace $\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}$

```
laplacian=D[#,{x,2}]+D[#,{y,2}]&
```

e applichiamolo a qualche funzione f(x,y)

```
laplacian[g[x, y]]
laplacian[x^2 + y^2]
laplacian[Cos[x + y]]
laplacian[Log[Sqrt[x^2 + y^2]]] // Simplify
```

Definiamo l'operatore $\frac{\partial^2}{\partial t^2} - \frac{\partial^2}{\partial x^2}$ e verifichiamo che la funzione $e^{-(x-t)^2}$ verifica l'*equazione delle onde*

```
wave = D[#, {t, 2}] - D[#, {x, 2}] &
wave[Exp[-(x - t)^2]]
```

Definiamo l'operatore $\frac{\partial}{\partial t} - \frac{\partial^2}{\partial x^2}$ e verifichiamo che la funzione $\frac{e^{-x^2/4t}}{\sqrt{t}}$ verifica l'*equazione del calore*

```
heat = D[#, t] - D[#, {x, 2}] &
heat[Exp[-x^2 / (4 t)] / Sqrt[t]] // Simplify
```

Esempio. Definire una funzione **derivata[f,x]** che ricopia la funzione predefinita **D** (**Derivative**) cioe' che abbia gli stessi attributi e sia tale che

- la derivata di una costante e' zero;

- valgano le proprieta' di linearita' e additivita';

- calcoli la derivata del prodotto e del quoziente di funzioni;

- calcoli la derivata di f(x)=x^a ;

- calcoli la derivata del seno, coseno, logaritmo, esponenziale.

Per verificare che c sia costante rispetto a x usiamo il test **/;** e la funzione **FreeQ[expr,form]** che verifica se *form* compare in *expr*: vale *True* se compare e *False* altrimenti

```
derivata[c_,x_]:=0/;FreeQ[c,x]
derivata[f_+g_,x_]:=derivata[f,x]+derivata[g,x]
derivata[c_ f_,x_]:=c derivata[f,x]/;FreeQ[c,x]
derivata[f_ g_,x_]:=f derivata[g,x]+g derivata[f,x]
derivata[ f_/g_ ,x_]:=(g derivata[f,x]-derivata[g,x] f)/g^2
derivata[x_^n_.,x_]:=n x^(n-1)/;FreeQ[n,x]
```

```
derivata[Sin[f_], x_] := Cos[f] derivata[f, x]
derivata[Cos[f_], x_] := -Sin[f] derivata[f, x]
derivata[Log[f_], x_] := (1 / f) derivata[f, x]
derivata[Exp[f_], x_] := Exp[f] derivata[f, x]
```

```
{derivata[y, x], D[y, x]}
{derivata[x^2 + y, x], D[x^2 + y, x]}
{derivata[Sin[x^2], x], D[Sin[x^2], x]}
{derivata[Cos[Cos[x]], x], D[Cos[Cos[x]], x]}
{derivata[Log[a x^2 + b], x], D[Log[a x^2 + b], x]}
{derivata[Exp[x y], y], D[Exp[x y], y]}
{derivata[Sin[x] Cos[x], x], D[Sin[x] Cos[x], x]}
{derivata[Exp[x^2] / Log[x], x], D[Exp[x^2] / Log[x], x]}
```

```
Attributes[derivata] = Attributes[D]
```

■ 5.6 Studio di funzione

Esempio 1. Studiamo la funzione razionale

```
ClearAll["Global`*"]
f[x_] := (1 - 2 x - x^2 + 2 x^3) / (-6 + x + x^2)
```

Vediamo come *Mathematica* disegna la funzione

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

Cominciamo uno studio piu' dettagliato.

Insieme di definizione:

```
Solve[-6 + x + x^2 == 0, x]
x /. %
xm = %[[1]]
xp = %%[[2]]
```

Studio del segno:

```
Needs["Algebra`InequalitySolve`"]
InequalitySolve[(1 - 2 x - x^2 + 2 x^3) / (-6 + x + x^2) ≥ 0, x]
```

Limiti alla frontiera del dominio e ricerca di asintoti:

```
Limit[f[x], x → xp, Direction → 1](*da sinistra*)
Limit[f[x], x → xp, Direction → -1](*da destra*)
Limit[f[x], x → xm, Direction → 1](*da sinistra*)
Limit[f[x], x → xm, Direction → -1](*da destra*)
```

Quindi x=xm e x=xp sono asintoti verticali.

```
Limit[f[x], x → Infinity]
Limit[f[x], x → -Infinity]
```

Potrebbero esserci asintoti obliqui:

```
m1 = Limit[f[x] / x, x → Infinity]
q1 = Limit[f[x] - m1 x, x → Infinity]
m2 = Limit[f[x] / x, x → -Infinity]
q2 = Limit[f[x] - m2 x, x → -Infinity]

asintoto = m1 x + q1

Plot[{f[x], asintoto}, {x, -7, 7},
 PlotStyle → {{}, {RGBColor[0, 1, 1 / 2], Dashing[{.01, .01}]}}]
```

Studio della derivata prima: determiniamo gli intervalli di crescenza e decrescenza e gli eventuali punti di massimo o minimo relativi

```
d1 = Simplify[D[f[x], x]];
num = Numerator[d1];
sol = NSolve[num == 0, x]
```

vogliamo una lista delle soluzioni

```
x /. sol
```

ovvero

```
x1 = x /. sol[[1]]
x2 = x /. sol[[2]]
x3 = x /. sol[[3]]
x4 = x /. sol[[4]]

InequalitySolve[d1 ≥ 0, x] // N
```

da cui deduciamo i seguenti intervalli in cui la funzione e' crescente: (-∞,x1), (x2,x3), (x4,+∞) e quindi i seguenti intervalli in cui la funzione e' decrescente: (x1,xm), (xm,x2), (x3,xp), (xp,x4).

Studio della derivata seconda:

```
d2 = Simplify[D[f[x], {x, 2}]];
Positive[d2 /. x → x1]
Positive[d2 /. x → x2]
Positive[d2 /. x → x3]
Positive[d2 /. x → x4]
```

x1 e x3 sono punti di massimo relativo; x2 e x4 sono punti di minimo relativo. La funzione assume in questi punti i seguenti valori:

```
{f[x1], f[x2], f[x3], f[x4]}

NSolve[d2 == 0, x](*cerchiamo gli eventuali punti di flesso*)
x /. %
```

Scartiamo le soluzioni complesse

```
xf = %[[3]]
InequalitySolve[d2 ≥ 0, x] // N
```

xf e' un punto di flesso. Dalla diseguaglianza precedente determiniamo gli intervalli di concavita' e convessita'. Determini

```
y = Simplify[(d1 /. x → xf) * (x - xf) + f[xf]]
Plot[{f[x], Last[%]}, {x, -1, 1}, Ticks → {{0, 0.2, 0.4, 0.6}, {-.1, 0, .1}},
 PlotStyle → {{}, {RGBColor[1, 0, 0]}}]
```

Esempio 2. Studiamo la funzione

```
ClearAll["Global`*"]
f[x_] := x + ArcTan[(x + 1) / x]
```

Per avere un'idea dell'andamento della funzione

```
Plot[f[x], {x, -2, 2}]
```

La funzione non e' definita in 0: calcoliamo i limiti da destra e da sinistra e i limiti a + ∞ e - ∞

```
Limit[f[x], x → 0, Direction → 1]
Limit[f[x], x → 0, Direction → -1]
Limit[f[x], x → Infinity]
Limit[f[x], x → -Infinity]
```

Verifichiamo se ci sono asintoti

```
m1 = Limit[f[x] / x, x → Infinity]
m2 = Limit[f[x] / x, x → Infinity]
q1 = Limit[f[x] - m1 x, x → Infinity]
q2 = Limit[f[x] - m2 x, x → -Infinity]

asintoto = m1 x + q1

Plot[{f[x], asintoto}, {x, -7, 7},
 PlotStyle → {{GrayLevel[.2]}, {RGBColor[1, 0, 1 / 2], Dashing[{.01, .01}]}}]

d1 = Simplify[D[f[x], x]]
Solve[d1 == 0, x]
xs = x /. %[[1]]
d2 = Simplify[D[f[x], {x, 2}]]
d2 /. x -> xs
```

Quindi xs e' un punto di massimo relativo.

```
Solve[d2 == 0, x]
xf = x /. %[[1]]
InequalitySolve[d2 ≥ 0, x]
```

xf e' un punto di flesso.

Esercizio 1. Sia f(x)=$\frac{x^2-3x}{x+1}$. Determinare l'insieme di definizione di f, studiare il segno di f, calcolare i limiti agli estremi dell'intervallo di definizione, gli eventuali asintoti, calcolare la derivata prima, gli intervalli di crescenza e decrescenza, gli eventuali punti di massimo o minimo relativi, studiare la derivata seconda, determinare gli eventuali punti di flesso. Tracciare il grafico di f.

Esercizio 2. Sia f(x)=$(x^2-8)e^x$. determinare l'insieme di definizione di f, studiare il segno di f, calcolare i limiti agli estremi dell'intervallo di definizione, gli eventuali asintoti, calcolare la derivata prima, gli intervalli di crescenza e decrescenza, gli eventuali punti di massimo o minimo relativi, studiare la derivata seconda, determinare gli eventuali punti di flesso. Tracciare il grafico di f.

Esercizio 3. Sia f(x)=x $e^{x/(x-2)}$. determinare l'insieme di definizione di f, studiare il segno di f, calcolare i limiti agli estremi dell'intervallo di definizione, gli eventuali asintoti, calcolare la derivata prima, gli intervalli di crescenza e decrescenza, gli eventuali punti di massimo o minimo relativi, studiare la derivata seconda, determinare gli eventuali punti di flesso. Tracciare il grafico di f.

Esempio. Definiamo una funzione che costruisca la retta tangente al grafico di una funzione f in un punto di ascissa x0

```
ClearAll["Global`*"]
tangente[f_, x_, x0_] :=
 Module[{f0, f10, t},
  f0 = f /. x → x0;
  f10 = D[f, x] /. x → x0;
  t = Simplify[f0 + f10 (x - x0)]
 ]

f[x_] := (1 + x) ^ 5
tangente[f[x], x, 1]
```

Disegnamo ora il grafico di f e della retta tangente intorno al punto x0 cioe' in (x0-h,x0+h), sempre rimanendo all'interno di **Module**. Il grafico di f viene tracciato in grigio e la retta tangente in rosso

```
grafico[f_, x_, x0_, h_] :=
 Module[{f0, f10, t},
  f0 = f /. x → x0;
  f10 = D[f, x] /. x → x0;
  t = Simplify[f0 + f10 (x - x0)];
  Plot[{f, t}, {x, x0 - h, x0 + h},
   AspectRatio → Automatic,
   PlotRange → {f0 - h, f0 + h},
   PlotStyle -> {{Thickness[0.005], GrayLevel[0.7]}, {RGBColor[1, 0, 0]}}]
 ]

grafico[ Exp[x], x, 0, 1]
grafico[Sin[x^2], x, Pi / 2, 1]
```

■ 5.7 Integrali

Il comando **Integrate[f[x],x]** calcola $\int f(x)\,dx$ cioe' calcola una primitiva della funzione f. L'integrale indefinito, cioe' l'insieme di tutte le primitive, si ottiene aggiungendo una generica costante c.

```
ClearAll["Global`*"]
Integrate[x^2 Sin[x], x]
Integrate[1 / (x^2 - 1), x]
Integrate[Sin[x]^3 Cos[x]^2, x]
Integrate[Sin[x^2], x]
```

Ogni simbolo diverso dalla variabile di integrazione viene trattato come costante

```
Integrate[Exp[-x^a], x]
Integrate[x^n, x]
```

Mathematica conosce molte funzioni e quindi e' in grado di calcolare un gran numero di integrali...ma non sempre si e' fortunati. Il comando **Integrate** calcola una primitiva esattamente, esprimendola eventualmente in termini di funzioni speciali

```
Integrate[Sqrt[Cos[x]], x]
```

Se questa primitiva non e' conosciuta allora *Mathematica* non calcola nulla

```
Integrate[Cos[Sin[x]], x]
```

Per il calcolo dell'integrale definito $\int_a^b f(x)\,dx$ si usa il comando **Integrate[f[x],{x,a,b}]**

```
Integrate[x^3 Cos[x], {x, 0, 4 Pi}]
```

oppure per il calcolo di integrali doppi estesi a domini normali si usa il comando **Integrate[f[x], {x,a,b}, {y,c,d}].** Attenzione all'ordine con cui scrivete {x,a,b}, {y,c,d}. Calcoliamo $\int_a^b dx \int_c^d f(x, y)\,dy$

```
Integrate[f[x, y], {x, a, b}, {y, c, d}]
```

e l'integrale $\int_c^d dy \int_a^b f(x, y)\,dx$

```
Integrate[f[x, y], {y, c, d}, {x, a, b}]
```

Calcoliamo $\int_0^{\pi} dx \int_{-\pi}^{2\pi} (x^3 \cos y + y \sin x)\,dy$

```
Integrate[x^3 Cos[y] + Sin[x] y, {x, 0, Pi}, {y, -Pi, 2 Pi}]
```

e l' integrale di $f(x, y) = (x^3 \cos y + y \sin x)$ esteso al dominio normale $\{0 \le x \le \pi, 0 \le y \le x^2\}$

```
Integrate[x^3 Cos[y] + Sin[x] y, {x, 0, Pi}, {y, 0, x^2}]
```

Visualizziamo il dominio di integrazione caricando il *package*

```
Needs["Graphics`FilledPlot`"]
FilledPlot[x^2, {x, 0, Pi}, Fills → RGBColor[1, 0, 0]]
```

Mathematica conosce le funzioni speciali ed e' in grado di calcolarne i valori numericamente. Per esempio

```
Integrate[Cos[Cos[x]], {x, 0, Pi}]

N[%]
```

In alcuni casi non riesce a calcolare il valore esatto dell'integrale

```
Integrate[1 / (1 + x + Exp[x^2]), {x, 0, 2}]
```

ma puo' calcolare il valore approssimato dell'integrale definito, con i metodi dell'analisi numerica usando il comando **NIntegrate**

```
NIntegrate[1 / (1 + x + Exp[x^2]), {x, 0, 2}]
```

Si possono anche calcolare integrali dipendenti da parametri

```
Integrate[1 / (1 + a Sin[x]), x] // Simplify
```

e integrali generalizzati

```
Integrate[1 / x, {x, 0, 1}](*l' integrale non converge*)

Integrate[Log[x] Exp[-x^2], {x, 0, Infinity}]
```

Con l'uso di **If** *Mathematica* esprime le condizioni affinche' l'integrale sia convergente

```
Integrate[1 / x^a, {x, 0, 1}]
Integrate[1 / x^a, {x, 1, Infinity}]
Integrate[Sin[a x] / x, {x, 0, Infinity}]
Integrate[x^n, {x, 0, 1}]
```

Assumptions e' un'opzione di **Integrate** che permette di specificare delle relazioni sui parametri che compaiono nell'integrale

```
Integrate[1 / x^a, {x, 0, 1}]
Integrate[1 / x^a, {x, 0, 1}, Assumptions → a < 1 ]

Integrate[1 / x^a, {x, 1, Infinity}]
Integrate[1 / x^a, {x, 1, Infinity}, Assumptions → a > 1]

Integrate[Sin[a x] / x, {x, 0, Infinity}]
Integrate[Sin[a x] / x, {x, 0, Infinity}, Assumptions → Im[a] == 0]
```

• Possiamo istruire *Mathematica* sul calcolo di integrali di funzioni che *Mathematica* non e' capace di calcolare definendo nuove regole di integrazione. Per esempio *Mathematica* non calcola il seguente integrale

```
Integrate[Sin[Sin[x]], x]
```

Prima di aggiungere nuove regole di integrazione, dobbiamo "rimuovere" la protezione dal comando **Integrate** (che e' una funzione con l'attributo **Protected**)

```
Unprotect[Integrate]
```

Definiamo la seguente regola di integrazione

```
Integrate[Sin[Sin[ x_]], x_] := sindue[x]
Protect[Integrate]
```

Ora *Mathematica* calcola l'integrale secondo la regola che abbiamo appena definito

```
Integrate[Sin[Sin[x]], x]
```

Con questa definizione *Mathematica* non riesce a calcolare **Integrate[Sin[Sin[3 x]],x]** oppure **Integrate[Sin[Sin[2 x+4]],x]** perche' cambi di variabili e altre trasformazioni non sono automaticamente eseguite da *Mathematica***.** Definisco la seguente nuova regola

```
Unprotect[Integrate]
ClearAll[Integrate]
Integrate[Sin[Sin[a_]], x_] := Sin[Sin[a]] x /; FreeQ[a, x]
Integrate[Sin[Sin[a_. + b_. x_]], x_] := sindue[a + b x] / b
Protect[Integrate]
```

Ora *Mathematica* calcola i seguenti integrali secondo la regola che abbiamo appena definito

```
Integrate[Sin[Sin[c]], x]
Integrate[Sin[Sin[x]], x]
Integrate[Sin[Sin[3 x]], x]
Integrate[Sin[Sin[2 x + 4]], x]
```

In questo modo possiamo definire altre regole. Per esempio

```
Unprotect[Integrate]
Integrate[x_ Sin[Sin[x_^2]], x_] := sindue[x^2] / 2
Protect[Integrate]
Integrate[x Sin[Sin[x^2]], x]
```

Esempio. Definiamo la funzione **integrate** che "ricopia" la funzione predefinita **Integrate**. Imponiamo cioe' le seguenti regole

1. Proprieta' di linearita' $\int (f+g)\,dx = \int f\,dx + \int g\,dy$

$\int c\,f\,dx = c\int f\,dx$, c non dipendente da x

```
integrate[f_ + g_, x_] := integrate[f, x] + integrate[g, x]
integrate[c_ f_, x_] := c integrate[f, x] /; FreeQ[c, x]
```

Verifichiamo

```
integrate[a x + b x^2 + c, x]
```

Definiamo alcune regole di integrazione. Per esempio $\int c\,dx = cx$; $\int x^n\,dx = \frac{x^{n+1}}{n+1}$, $n \neq -1$.

Usando nella definizione **x_^n_.** piuttosto che **x_^n_**
includiamo anche il caso di **x** che viene interpretato da *Mathematica* come **x^1**

```
integrate[c_, x_] := c x /; FreeQ[c, x]
integrate[x_^n_., x_] := x^(n + 1) / (n + 1) /; (FreeQ[n, x] && n ≠ -1)
```

Ora l' integrale $\int (ax + bx^2 + c)\,dx$ puo' essere calcolato completamente

```
integrate[a x + b x^2 + c, x]
```

Definiamo ora la regola per integrare $\frac{1}{ax+b}$ e l' esponenziale e^{ax+b}

```
integrate[1 / (a_. x_ + b_.), x_] := Log[a x + b] / a /; FreeQ[{a, b}, x]
integrate[Exp[a_. x_ + b_.], x_] := Exp[a x + b] / a /; FreeQ[{a, b}, x]
```

e verifichiamo

```
integrate[1 / x, x]
integrate[1 / (2 x + 1), x]
integrate[Exp[x], x]
integrate[Exp[2 x + 1], x]

Attributes[integrate] = Attributes[Integrate]
```

Si puo' naturalmente andare avanti e definire altre regole di integrazione.

■ 5.8 Equazioni Differenziali Ordinarie

Mathematica riesce anche a risolvere alcune equazioni differenziali (praticamente quelle che riusciamo a risolvere a mano...ma non fa errori). Il comando e' **DSolve[eqn,y[x],x]** dove y[x] e' la funzione incognita e x la variabile indipendente oppure **DSolve[eqn,y,x]** la cui soluzione y e' una *funzione pura*. Per esempio, consideriamo una equazione del secondo ordine a coefficienti costanti

```
DSolve[y''[x] + y[x] == 1, y[x] , x] // Flatten
```

isoliamo la soluzione y[x] con la regola di sostituzione immediata **/.**

```
y[x] /. %

DSolve[y''[x] + y[x] == 1, y , x] // Flatten
y /. %

DSolve[y''[x] + 3 y[x] == 1, y[x] , x] // Flatten
sol = y[x] /. %
```

Con il comando **Flatten** abbiamo rimosso le parentesi inutili. *Mathematica* fornisce l'integrale generale chiamando C[1] e C[2] le costanti arbitrarie. Una soluzione particolare e' data da

```
sol /. {C[1] → 1, C[2] → 3}
```

Con l'opzione **GeneratedParameters→K** si possono chiamare le costanti arbitrarie K[1],... Risolviamo un'equazione lineare del primo ordine

```
ClearAll["Global`*"]
DSolve[y'[x] + Sin[x] y[x] == 0, y[x], x, GeneratedParameters → K] // Flatten
```

Risolviamo una equazione del primo ordine a variabili separabili

```
DSolve[y[x] y'[x] == x^4, y[x], x] // Flatten
```

oppure una equazione con parametro

```
DSolve[y''[x] - a y[x] == 0, y[x], x] // Flatten
DSolve[y''[x] + a y[x] == 0, y[x], x] // Flatten
```

Per risolvere il problema di Cauchy la sintassi e' **DSolve[{eqn,listacond},y[x],x]** (o **DSolve[{eqn,listacond}, y, x]**).

Consideriamo i seguenti esempi

```
DSolve[{y'[x] == y[x], y[0] == 1}, y[x], x] // Flatten
y[x] /. %
Plot[%, {x, -1, 1}]

DSolve[{y'[x] == y[x], y[0] == 1}, y, x] // Flatten
y /. %[[1]]
Plot[%[x], {x, -1, 1}]

DSolve[{y''[x] + y[x] == 1, y'[0] == 1.5, y[0] == 2}, y[x], x] // Flatten

DSolve[{y''[x] + 3 y'[x] - 2 y[x] == 0, y'[0] == 1.5, y[0] == 2}, y[x], x] //
 Flatten
Plot[y[x] /. %, {x, -1, 1}]
```

Mathematica non e' invece capace di risolvere l'equazione

```
DSolve[y''[x] + Sin[y[x]] == 1, y[x], x]
```

E' comunque possibile costruire una soluzione, in un intervallo fissato, con i metodi dell'Analisi Numerica, assegnando opportune condizioni iniziali e specificando il **Range** in cui far variare la variabile indipendente x. Il comando e' **NDSolve[{eqn,listacond}, y, {x,xmin,xmax}].** La soluzione viene fornita sotto forma di funzione di interpolazione che contiene una tabella dei valori $y(x_i)$ che la soluzione assume in punti x_i dell' intervallo , cioe' una funzione il cui valore "approssimato" in un punto e' trovato tramite interpolazione. Questa funzione puo' essere rappresentata graficamente come una qualsiasi altra funzione.

In generale *Mathematica*, con il comando **Interpolation[{{x1,f1},{x2,f2}...}]** costruisce una funzione di interpolazione (**InterpolatingFunction,** che e' una *funzione pura*), per interpolazione nella successione di punti assegnati, assumendo che la funzione sia "regolare" . Quindi quando chiediamo a *Mathematica* di calcolare il valore che la funzione assume in un punto, la funzione **Interpolation** trova questo valore per interpolazione. Per esempio costruiamo una tabella dei valori che cos(x) assume nell'intervallo $(0,\pi)$, distanziati di $\frac{\pi}{20}$ e costruiamo la funzione di interpolazione che chiamiamo **cos**

```
Table[{x, Cos[x]}, {x, 0, Pi, Pi / 20}];
cos = Interpolation[%]
{cos[0.3], Cos[0.3]}

{NIntegrate[cos[x]^2, {x, 0, Pi / 2}], NIntegrate[Cos[x]^2, {x, 0, Pi / 2}]}
```

Non distinguiamo i grafici delle due funzioni in $(0,\pi)$

```
Plot[{Cos[x], cos[x]}, {x, 0, Pi},
 PlotStyle → {{RGBColor[1, 0, 0]}, {RGBColor[0, 1, 0]}}]
```

pero' se ingrandiamo il grafico notiamo la differenza:

```
Plot[{Cos[x], cos[x]}, {x, 0, Pi / 200},
 PlotStyle → {{RGBColor[1, 0, 0]}, {RGBColor[0, 1, 0]}}]
```

Esempio. Risolviamo numericamente l'equazione differenziale y''(x)+sin(y(x))=1 con le condizioni di Cauchy y'(0)=0, y(0)=1, nell'intervallo (-1,1)

```
soluzione = NDSolve[
    {y''[x] + Sin[y[x]] == 1, y'[0] == 0, y[0] == 1}, y, {x, -1, 1}] // Flatten
y[x] /. soluzione (*funzione di interpolazione*)
```

<> rappresenta una lunga lista di output che non viene visualizzata. Puo' essere visualizzata con il comando **FullForm.**

Ho generato la soluzione nell'intervallo (-1,1). Calcoliamo la soluzione nei punti x=0,1/2,1

```
% /. x → 0
%% /. x → 1 / 2
%%% /. x → 1
```

Con il comando **Evaluate** chiedo a *Mathematica* di costruire una tabella della funzione di interpolazione. Successivamente, con **Plot**, calcolo la funzione per diversi valori della variabile indipendente.

```
Evaluate[y[x] /. soluzione]
FullForm[%]
```

Visualizziamo la tabella della soluzione numerica, con intervalli tra due punti di 0.25.

```
Table[Evaluate[y[x] /. soluzione], {x, -1, 1, 0.25}]
```

e disegnamo la soluzione

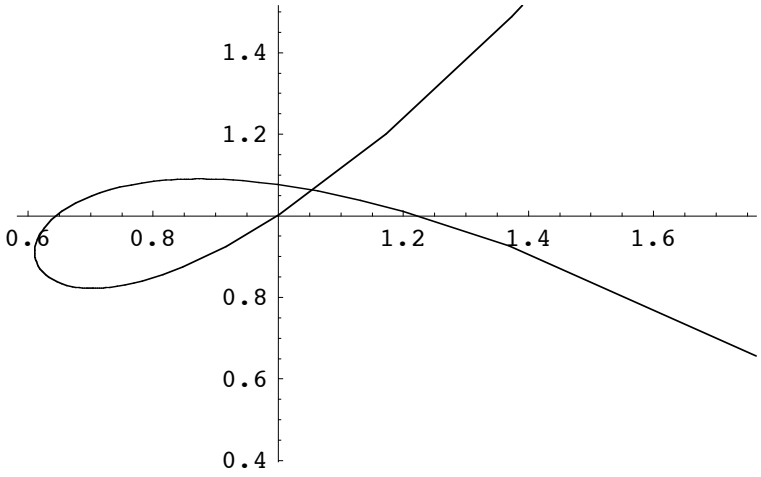
```
Plot[Evaluate[y[x] /. soluzione], {x, -1, 1}]
```

Si possono anche assegnare arbitrariamente le condizioni iniziali. In questo caso non e' detto che la soluzione esista e sia unica. Nel prossimo esempio abbiamo quattro soluzioni, che possiamo disegnare insieme.

```
NDSolve[{y'[x]^2 == y[x]^2, y[0]^2 == 4}, y, {x, -1, 1}]
Plot[Evaluate[y[x] /. %], {x, -1, 1},
 PlotStyle →
  {{RGBColor[1, 0, 0]}, {RGBColor[0, 1, 0]}, {RGBColor[0, 0, 1]}, {}}]
```

Si possono usare i comandi **DSolve** e **NDSolve** per risolvere sistemi di equazioni differenziali. Per esempio

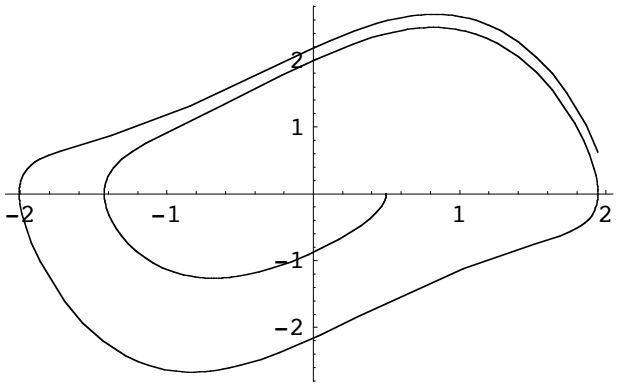
```
DSolve[{x'[t] + y[t] == 2 t Sin[t], y'[t] + x[t] == 2 t Cos[t], x[0] == 1, y[0] == 1},
  {x, y}, t] // Flatten
sol = {x[t], y[t]} /. %;


ParametricPlot[sol, {t, -2, 2}]
```




```
- Graphics -
```

L'oscillatore di Van der Pol:

```
eq1 = x'[t] == y[t];
eq2 = y'[t] == (1 - x[t]^2) y[t] - x[t];
soluzione = NDSolve[{eq1, eq2, x[0] == 1 / 2, y[0] == 0}, {x, y}, {t, 0, 12}]

ParametricPlot[Evaluate[{x[t], y[t]} /. soluzione], {t, 0, 12}]
```




```
- Graphics -
```