Come fa il computer a calcolare le derivate ?
manipolando degli alberi

Roberto Innocente

SISSA

È un peccato che l'evoluzione abbia portato gli uomini ad abituarsi ad una scrittura lineare. Molti soggetti avrebbero potuto essere espressi con molta maggiore semplicità e rigore usando una rappresentazione almeno bi-dimensionale. Pensiamo al *linguaggio* od alle *espressioni matematiche*.

In effetti i calcolatori leggono la nostra scrittura lineare delle espressioni matematiche (la nostra abituale notazione delle espressioni e' detta **infissa**, poichè gli operatori vengono infilati tra gli operandi), ma poi le trasformano in **alberi** che descrivono succintamente e con precisione l'ordine delle operazioni ed il loro raggruppamento e sono più facilmente trattabili dai calcolatori.

Per aver subito un'idea di cosa sia un albero in informatica pensate agli *alberi genealogici*. Questi come quelli dell'informatica sono gli unici alberi con la radice posta in alto !

I componenti di un albero sono detti **nodi**. Il nodo in cima a tutti è detto **radice**. Si dice **arco** un collegamento tra due nodi di livelli adiacenti. Un arco collega un nodo **genitore** con un nodo **figlio**. I nodi che non hanno figli si dicono **foglie**. Tutti i nodi della stessa generazione sono solitamente allineati orizzontalmente. Ogni nodo ha un genitore, eccetto il nodo radice. Non è necessario mettere le frecce per dare una direzione alla relazione genitore/figlio poichè il genitore sta sempre al livello superiore ed il figlio al livello inferiore. Un **sottoalbero** è l'insieme di un nodo e tutti i suoi discendenti.

La trasformazione di una espressione in notazione infissa in un albero è un problema su cui in passato si sono scritti libri e che oggi è ben conosciuto ed applicato da tutti i compilatori. Lo tralasceremo poichè non riguarda direttamente la matematica e supporremo di avere già disponibile un albero dell'espressione che vi dovrebbe essere molto facile ricavare mentalmente dall'espressione.

Quando molto tempo fa ho imparato il linguaggio di programmazione LISP(List Processing), uno dei primi programmi che ho fatto in questo linguaggio, di qualche decina di righe , è proprio un programma per il calcolo delle derivate. Questo perchè con questo linguaggio uno può direttamente maneggiare gli alberi delle espressioni (**maxima** è scritto in LISP). (Nel 1958 *John McCarthy* inventò il LISP, un linguaggio molto differente dagli esistenti linguaggi procedurali, proprio per poter facilmente descrivere il calcolo delle derivate.)

Nel seguito sono illustrate alcune delle regole fondamentali per il calcolo delle derivate come manipolazioni di rami di un albero. Vi invito a completare le regole finchè sarete in grado, con queste, di calcolare qualsiasi derivata.

A tal punto potrete ritenervi capaci di scrivere un programma per la derivazione simbolica.

Legenda :

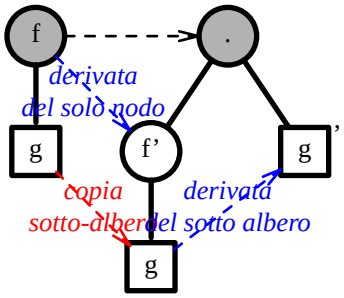
nodo radice nodo sottoalbero derivata sottoalbero

Nelle regole per la derivazione a sinistra viene rappresentata l'espressione da derivare ed a destra la sua derivata. Una linea tratteggiata collega i nodi radice dei due alberi. La derivata del sottoalbero viene quindi sostituita al sottoalbero originario.

Si sono semplificati gli alberi non scrivendo i nodi della variabile, sempre x, che supporremo sempre figlia di nodi funzione senza altri figli.

Derivata funzione composta / Chain rule :

$$[f(g(x))]' = f'(g(x))g'(x)$$

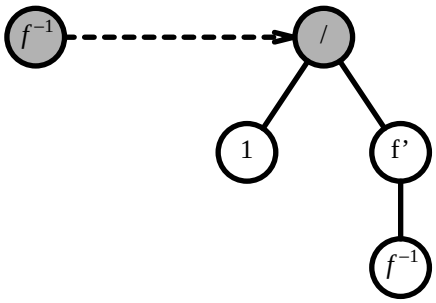


Se volessimo descrivere questa manipolazione in un pseudo-linguaggio di programmazione, potremmo scrivere :

```
derivata(t) :
   Se la radice f di t è una funzione, allora :
    crea_albero(t2,radice='.')
    t2.aggiungi_figlio(derivata( nodo f)).aggiungi_figlio(sottoalbero g)
    t2.aggiungi_figlio(derivata(sottoalbero g))
   ritorna(t2)
```

Derivata funzione inversa :

$$[f^{-1}(x)]' = \frac{1}{f'(f^{-1}(x))}$$

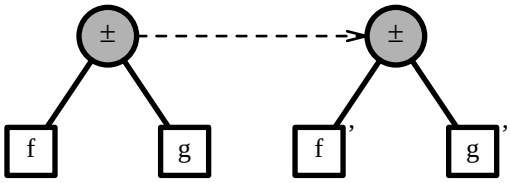


Lo pseudo-codice per questa manipolazione potrebbe essere scritto :

```
derivata(t) :
  Se la radice di t e' l'inversa di una funzione f, allora :
    crea_albero(t2,radice="/")
    t2.aggiungi_figlio("1")
    t2.aggiungi_figlio(derivata ( nodo f)).aggiungi_figlio(inversa di f)
  ritorna(t2)
```

Derivata somma/sottrazione :

$$[f(x)\pm g(x)]' = f'(x)\pm g'(x)$$

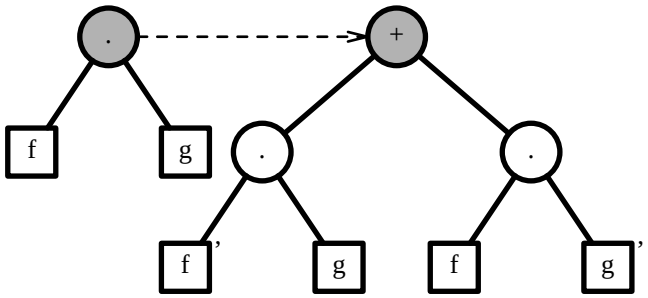


Derivata identità :

$$[x]' = 1$$

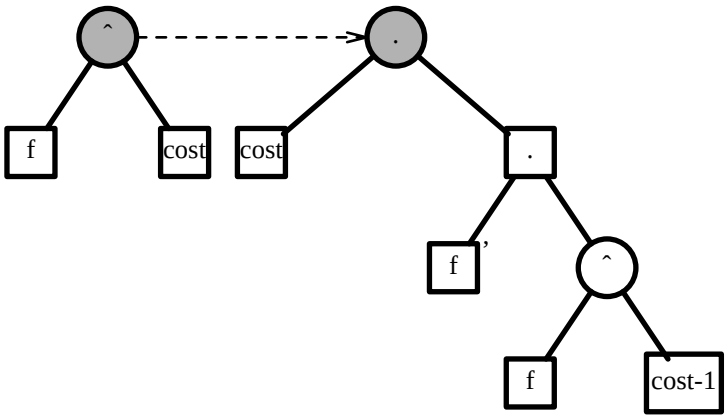
Derivata prodotto :

$$[f(x)g(x)]' = f'(x)g(x) + f(x)g'(x)$$



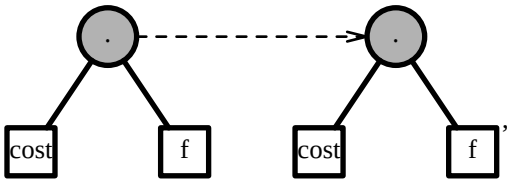
Derivata potenza :

$$[f(x)^{cost}]' = cost * f'(x) * f(x)^{(cost-1)}$$



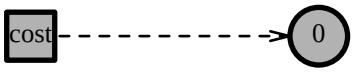
Derivata costante * funzione :

$$[cost * f(x)]' = cost * f'(x)$$



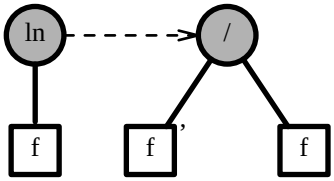
Derivata costante:

$$[cost]' = 0$$



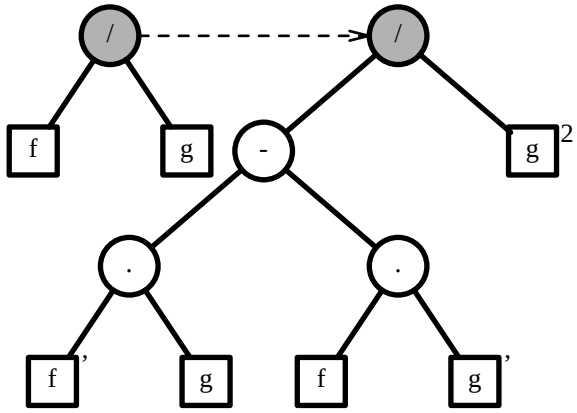
Derivata logaritmo naturale :

$$[ln(f(x))]' = \frac{f'(x)}{f(x)}$$



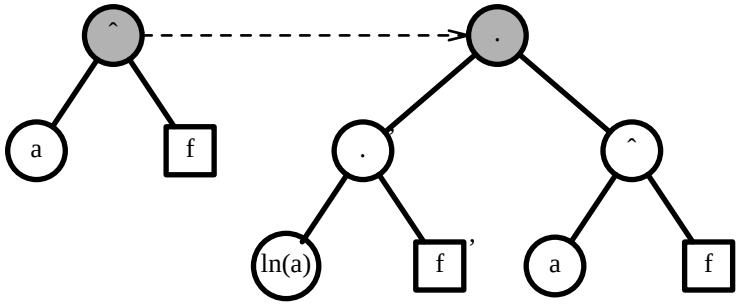
Derivata quoziente:

$$[f(x)/g(x)]' = \frac{f'(x)g(x) - f(x)g'(x)}{g^2(x)}$$



Derivata esponenziale:

$$[a^{f(x)}]' = [e^{ln(a)f(x)}]' = ln(a) * f'(x) * a^{f(x)}$$



Derivata seno :

$[sin(f(x))]' = f'(x) * cos(f(x))$

Derivata coseno :

$[cos(f(x))]' = -f'(x) * sin(f(x))$

Derivata tangente :

$[tan(f(x)]' = f'(x) * sec(f(x))^2$

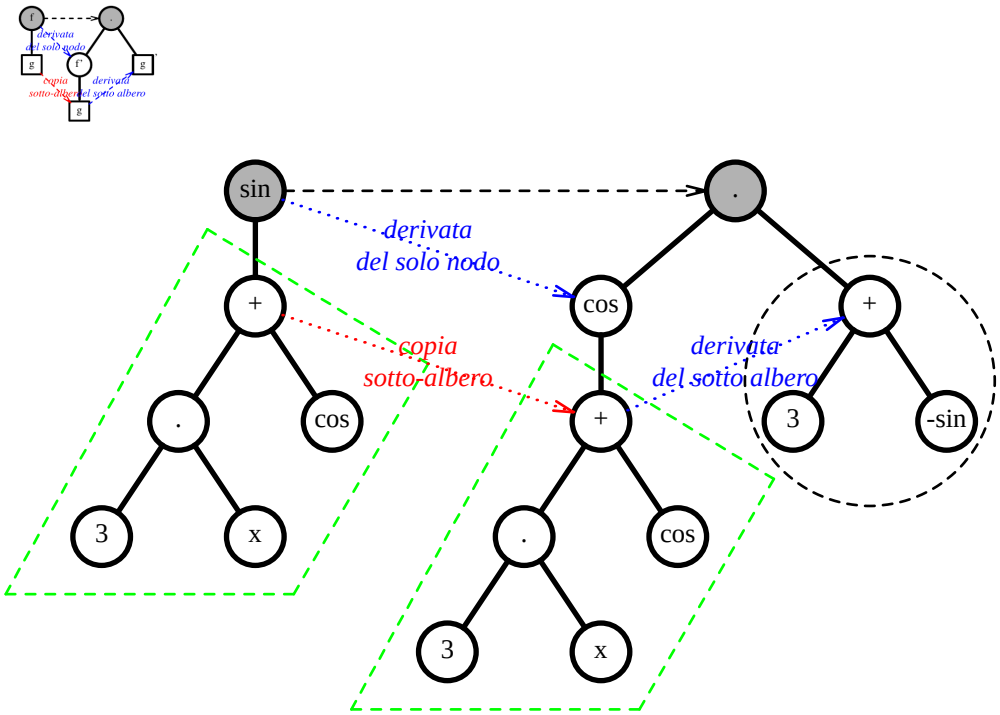
Derivata radice quadrata :

$$[\sqrt{(f(x))}]' = \frac{1}{2}\frac{f'(x)}{\sqrt{(f(x))}}$$

Esempio :

$$[sin(3x + cos(x))]'?$$

Alcuni passaggi sono omessi sulla carta (non siamo così veloci come un calcolatore). Essenzialmente abbiamo applicato la trasformazione relativa alle funzioni composte sull'albero dell'espressione originale..



Poniamo quindi che t sia l'albero (useremo la notazione infissa per rappresentare gli alberi qui, perchè altrimenti dovremmo disegnarli):

```
sin( 3 x + cos(x))
```

i passi che il nostro programma per le derivate farebbe sono :

```
derivata(sin(3 x + cos(x)))
cos(3 x + cos(x)) * derivata(3 x + cos(x))  #derivata funzione composta/chain rule
cos(3 x + cos(x)) * ( derivata( 3 x) + derivata(cos x)) #derivata somma
cos(3 x + cos(x)) * ( 3 + derivata(cos(x)))  # derivata prodotto e semplificato
cos(3 x + cos(x)) * ( 3 - sin(x))  # derivata cos(x)
```

Il meno unario (il meno di -sin(x) non e' l'operatore binario della sottrazione)
della derivata del *cos(x)* in effetti darebbe come risultato un albero con genitore il meno unario e figlio il *sin(x)* , ma per semplificare l'ho scritto come singolo atomo.