

Calcolo della derivata di a^x

un esempio di uso di bc

Come noto la derivata rispetto a x di a^x , con $a > 0$, è semplicemente $e^x \ln a$. Ci possiamo convincere di ciò molto facilmente. Infatti $\ln a^x = x \ln a$, e dunque $a^x = e^{x \ln a}$, ed essendo $\ln a$ una costante sappiamo ben calcolare la derivata che sarà dunque $e^{x \ln a} \ln a = a^x \ln a$.

Ora supponiamo di voler calcolare tale derivata come limite del rapporto incrementale. Essendo $f(x) = a^x$ vogliamo calcolare

$$\lim_{\varepsilon \rightarrow 0} \frac{f(x + \varepsilon) - f(x)}{\varepsilon}$$

cioè

$$\lim_{\varepsilon \rightarrow 0} \frac{a^{x+\varepsilon} - a^x}{\varepsilon} = \lim_{\varepsilon \rightarrow 0} \frac{a^x(a^\varepsilon - 1)}{\varepsilon} = a^x \lim_{\varepsilon \rightarrow 0} \frac{a^\varepsilon - 1}{\varepsilon}$$

Ci concentriamo sul solo calcolo del limite (che è noto ma noi lo calcoleremo numericamente). Per far ciò usiamo bc. Per prima cosa scriviamo il codice (lo vediamo e lo commentiamo a pag. 3) che ci permette di calcolare il limite. Facciamo poi girare questo codice con diversi $a > 0$, in modo da poter costruire per punti una funzione. Ci accorgiamo poi di che funzione si tratta e lo verificiamo.

Una semplice generalizzazione del codice presentato ci porta una tabella di valori che possiamo graficare per esempio con gnuplot. Il grafico che osserviamo è in effetti il grafico di un logaritmo.

Facciamo girare il codice del programma “completo” (il codice sorgente senza commenti è dato in modo tipografico a pag. 5) per ottenere la tabella di cui potete vedere un estratto in tab. 1

Per verificare che questa sia proprio la funzione logaritmo, basta calcolare il logaritmo di a su tutti i valori e confrontarli con $f(a)$. Tuttavia in questo documento vogliamo fingere semplicemente di non saperlo; il metodo in linea di principio

a	$f(a)$
.25000	-1.3862943611198906188
.50000	-.6931471805599453094
.75000	-.2876820724517809274
1.00000	0
1.25000	.2231435513142097557
1.50000	.4054651081081643819
1.75000	.5596157879354226862
2.00000	.6931471805599453094
2.25000	.8109302162163287639
2.50000	.9162907318741550651
2.75000	1.0116009116784799252
3.00000	1.0986122886681096914

Tabella 1: I punti generati dal codice di pag. 5; siamo partiti dal valore $a = 0.25$ fino al valore 8, a incrementi di .25. Si noti che, come atteso, la funzione f è negativa quando $a < 1$.

può essere esteso a funzioni di cui non sia nota veramente (*sic*) la derivata! Per questo usiamo il nostro programma preferito per poter intanto vedere ad “occhio” la funzione. Abbiamo usato `gnuplot` per generare il grafico di fig. 1.

Ciò che abbiamo fatto è stato impartire i semplici comandi

```
plot [x=.25:8] log(x), 'tab.dat'
```

dove `tab.dat` è il nome del file che contiene le coppie di valori $(a, f(a))$, come generate dal programma per `bc`. *Casualmente* i punti cadono proprio sulla curva $\log(x)$ (`gnuplot` indica con `log` il logaritmo naturale).

Un semplice `plot 'tab.dat'` ci avrebbe comunque mostrato i dati e avremmo riconosciuto ad occhio la possibilità che questi punti fossero punti della funzione logaritmo. In casi più complessi (magari per l'integrazione piuttosto che per la derivazione) sarà certamente necessario pensare a un'interpolazione e dei test con delle funzioni di prova parametrizzate (cosa che `gnuplot` permette di fare).

Per concludere, a parte il caso specifico semplice, il metodo può essere affinato e usato per situazioni più spinose (derivate discrete, integrali definiti etc.)

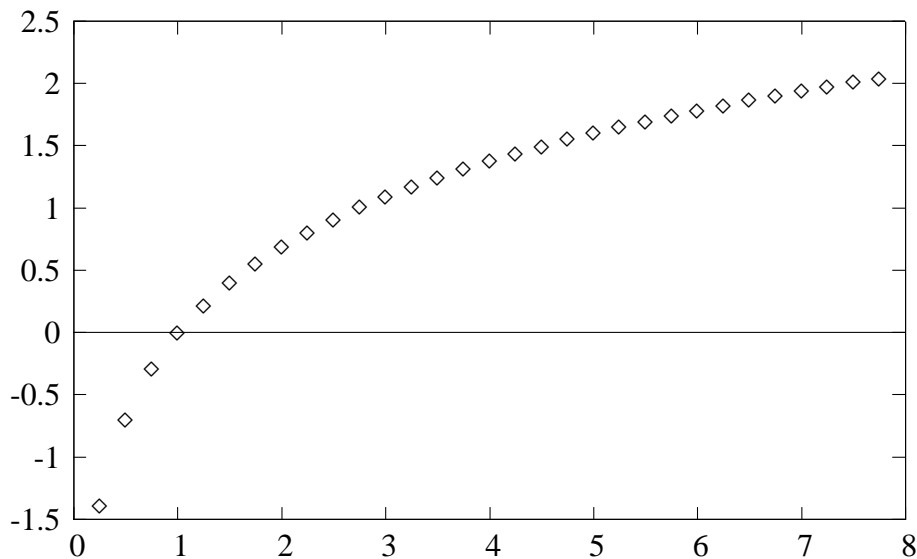


Figura 1: In questa figura vediamo i punti generati dal programma; cioè ciascun punto rappresenta il limite, per ϵ che tende a zero, di $(a^\epsilon - 1)/\epsilon$, per i diversi valori di a in ascissa..

Codice base commentato

Per prima cosa introduciamo alcune funzioni utili e osserviamo che ci appoggiamo alla libreria matematica che si precarica lanciando `bc` con argomento `-l`.

```
define abs(x)
{
  if (x > 0) return x;
  return -x;
}
```

Questa funzione molto semplicemente calcola il valore assoluto del suo argomento. Abbiamo poi bisogno di una funzione che calcola una potenza qualunque, anche non intera, di una base qualunque. Questo perché l'operatore elevamento a potenza di `bc` accetta solo esponenti interi, cioè la cui *scala* sia zero.

```
define pow(base, esp)
{
  if ( scale(esp) == 0 ) return(base^esp);
```

```
    return ( e(esp*l(base)) )
}
```

È abbastanza ovvio ciò che abbiamo fatto; in pratica abbiamo sfruttato la libreria matematica che definisce l'esponenziale e il logaritmo base naturale.

```
scale=500
```

Vogliamo lavorare con molte cifre anche se poi, dato il limite imposto da *soglia*, per il risultato finale non servirà tenerle.

Dobbiamo iterare più volte man mano che facciamo tendere ε a zero. Però è meglio impostare un limite massimo delle iterazioni:

```
iterazioni=200
```

Poi definiamo il valore di inizio del nostro epsilon e una “precisione”, da cui calcoliamo la soglia.

```
inizio=1
prec = 20
soglia = (1/(10^prec))
```

Poi scegliamo un valore per a . Tanto per fissare le idee questo sarà $a = 4$. Ma se vogliamo trovare la dipendenza funzionale, lo dovremo far variare in un intervallo e generare una “tabella” che potremo usare per esempio con *gnuplot* per generare un grafico.

```
a = 4
```

Ora scegliamo un valore iniziale “a caso” di epsilon (*eps*), poiché rendiamo epsilon sempre più piccolo dividendolo per 10 la cosa più ovvia sembra partire da una potenza di 10 appunto. Lo scopo è solo evitare che nella prima iterazione la differenza tra il vecchio valore calcolato e quello nuovo sia già inferiore della soglia. In seguito il vero valore di partenza sarà dato da *inizio*.

```
eps = 10
val = (pow(a,eps)-1)/eps
```

Dunque, iniziamo le iterazioni.

```

eps=inizio
for (i=0; i<iterazioni; i++)
{
  print val, "\n\n";
  val2 = (pow(a,eps) - 1)/eps
  dif = (val - val2)
  if ( abs(dif) < soglia ) { break; }
  val = val2
  eps /= 10
}

```

L'idea è molto semplice: per non più di 200 volte calcoliamo il rapporto $(a^\epsilon - 1)/\epsilon$, e ne facciamo la differenza con il valore calcolato nell'iterazione precedente. Se tale differenza è al di sotto del valore di soglia scelto, allora usciamo dal ciclo iterativo.

Per concludere, consideriamo che anche se i conti sono stati fatti mantenendo *scale* elevato (cioè con 500 cifre decimali), dato il valore di soglia determinato dalla variabile *prec* (che starebbe per "precisione"), non ha senso mantenere tutti quei decimali. Quindi il valore che andiamo a stampare in finale sarà con $prec - 1$ cifre.

```

print "IL VALORE DI LN(", a, ") E' CIRCA:\n\n"
scale = (prec - 1)
val /= 1
print val, "\n\n"
quit

```

(La "scala" è associata al numero e un solo cambio della variabile speciale *scale* non è sufficiente per diminuire le cifre decimali di *val*. L'unico modo, come si evince dalla documentazione di bc e da esempi forniti, è proprio quello di eseguire una divisione per uno.)

Codice per generare la funzione per punti

Per comodità abbiamo ommesso le funzioni già definite che il lettore può trovare nel codice su proposto. La sola cosa che vogliamo far notare è che abbiamo scelto una scala (il numero di cifre decimali secondo la definizione di bc), molto più ragionevole, cosa che consente una esecuzione più rapida. Inoltre si è usato `␣` per evidenziare uno spazio.

```

scale=80
prec = 20
soglia = (1/(10^prec))
iterazioni = 200

define funz(a)
{
    auto eps, i, val, val2, z
    val = (pow(a,10) - 1)/10
    eps=1

    for(i=0; i<iterazioni; i++)
    {
        val2 = (pow(a,eps)-1)/eps
        dif = (val - val2)
        if (abs(dif) < soglia ) { break; }
        val = val2
        eps /= 10
    }
    z = scale
    scale = prec - 1
    val /= 1
    scale = z
    return val
}

for(a=.25; a < 8; a += .25 )
{
    v = funz(a)
    z = scale
    scale = 5
    a /= 1
    print a, “□”, v, “\n”
    scale = z
}

quit

```