# nb3-derivata_numerica

October 16, 2017

## Derivata numerica

Vogliamo calcolare la derivata di una funzione f(x) in un dato punto x. Sappiamo dalla definizione che dobbiamo calcolare un rapporto incrementale del tipo $(f(x+h)-f(x))/h$, facendo tendere a zero il passo $h$. Proviamo quindi ad impostare il nostro problema per due funzioni di test:

- $f(x) = sin(x)$
- $f(x) = 2x^2 + x$

Per queste due funzioni sappiamo gia' che la derivata di sin(x) e' cos(x) e che la derivata della $f$ e' $4x + 1$, per cui possiamo usare questa informazione per verificare il corretto funzionamento delle nostre procedure di calcolo. Per procedere iniziamo a definire nel nostro ambiente di calcolo due funzioni che ci serviranno:

```
In [ ]: from math import sin, radians
```

Con il comando precedente abbiamo estratto dal pacchetto **math** le due funzioni **sin** e **radians** (gia' scritte e collaudate da altri autori) che d'ora in avanti rimangono disponibili per tutta la sessione di lavoro.

Passiamo ora a definire le nostre funzioni (la funzione **sinus** si potrebbe anche evitare perche' il sin e' gia' disponibile, tuttavia la definiamo per completezza di esercizio):

```
In [53]: def sinus(x):
             # definizione della prima funzione
             return sin(radians(x))

         def f(x):
             # definizione della seconda funzione
             return 2*x**2+x

         def derivative(func, x):
             """
             definizione generale di derivata, con step fissato
             INP:   func     generica funzione gia' definita
             """
             h = 0.000000001    # fisso un valore piccolo per lo step
             return (func(x+h)-func(x))/h
```

```
        x=5.
        theta=60.
        print("derivata di f in x="+str(x)+': ',derivative(f, x))
        print("derivata di sin("+str(theta)+') :' ,derivative(sinus, theta))
```

```
derivata di f in x=5.0:  21.00000529026147
derivata di sin(60.0) : 0.008726686040461118
```

ATTENZIONE: La risposta per la funzione f e' coerente ma nel caso del $sin$ ci aspettiamo che la sua derivata nel punto a 60 gradi sia 0.5, invece il nostro calcolo qui sopra produce un valore anoamalo: 0,00872.....

Dov'e' l'errore ?

Il problema nasce dal fatto che quando la function **derivata** va ad utilizzare la function **sinus** in input, il numeratore viene calcolato correttamente a due angoli (in radianti) diversi mentre il denominatore non viene espresso in radianti come dovrebbe!. Per curare questo problema bastera' convertire $h$ in radianti.

Per far questo andiamo a correggere l'errore ridefinendo la function **derivative** in modo tale da riconoscere il caso in cui sia opportuno tradurre $h$ in radianti prima di calcolare il rapporto incrementale:

```
In [52]: def derivative(func, x):
             """
             definizione generale di derivata, con step fissato
             INP:   func     generica funzione gia' definita
             """
             h = 0.000000001    # fisso un valore dello step
             if func.__name__ is "sinus":
                 return (func(x+h)-func(x))/radians(h)
             else:
                 return (func(x+h)-func(x))/h
             print((func(x+h)-func(x))/h)
         # proviamo a richiamare i due casi:
         x=5.
         theta=60.
         print("derivata di f in x="+str(x)+': ',derivative(f, x))
         print("derivata di sin("+str(theta)+') :' ,derivative(sinus, theta))
```

```
derivata di f in x=5.0:  21.00000529026147
derivata di sin(60.0) : 0.5000022792541535
```

Abbiamo adottato un valore di $h$ piccolo, ma quanto piccolo deve essere per avvicinarci il piu' possibile alla derivata analitica nel punto?

E' bene notare che nel calcolo numerico spesso non e' una buona idea svolgere i calcoli esattamente come li faremmo sulla carta: dobbiamo infatti preoccuparci degli errori introdotti dalla precisione finita delle operazioni in floating point. E' utile quindi renderci conto di quale sia il livello di precisione del nostro ambiente di calcolo. Per avere questa informazione e' utile usare il seguente comando che richiama la funzione np.finfo (strano nome che sta per **n**umerical **p**ython.**f**loating point **info**)

```
In [69]: np.finfo(float).eps

Out[69]: 2.2204460492503131e-16
```

Come curiosita' verifichiamo che lo stesso risultato si ottiene sfruttando il modo in cui la macchina rappresenta internamente i numeri floating point:

```
In [60]: print(7./3 - 4./3 -1)

2.220446049250313e-16
```

Ancora possiamo valutare approssimativamente l'epsilon della macchina aggiungendo all'unita' una quantita' **epsilon** via via piu' piccola fino a diventare 'zero' agli effetti pratici del calcolo. Questa condizione segnala che la epsilon raggiunta e' al disotto della rappresentazione interna della macchina:

```
In [79]: epsilon = 1.0
         while (1.0 + 0.5 * epsilon) > 1.0:
             epsilon = 0.5 * epsilon
         print (epsilon)

2.220446049250313e-16
```

Per non rischiare di ritrovarci con i nostri calcoli al limite della precisione della macchina adottiamo cautelativamente la radice quadrata della precisione come minimo valore per la $h$ :

```
In [54]: np.sqrt(np.finfo(float).eps)

Out[54]: 1.4901161193847656e-08
```

Il numero che appare (praticamente e' $10^{-8}$) rappresenta il minimo valore che adottiamo per una migliore implementazione della nostra funzione che calcola la derivata:

```
In [91]: def derivative(func, x, h=None):
             """
             definizione generale di derivata, con step fissato
             INP:   func     generica funzione gia' definita
             """
             if h is None:
                 # Notare il valore attribuito all'incremento h che
                 # corrisponde alla radice quadrata della precisione
                 # in floating point, questo valore si puo' ottenere
                 # chiamando: np.sqrt(np.finfo(float).eps)
                 h = 1.49011611938477e-08
             if func.__name__ is "sinus":
                 return (func(x+h)-func(x))/radians(h)
             else:
                 return (func(x+h)-func(x))/h
```

```
        # proviamo a richiamare i due casi:
        x=5.
        theta=60.
        print("derivata di f in x="+str(x)+': ',derivative(f, x))
        print("derivata di sin("+str(theta)+') :' ,derivative(sinus, theta))
```

```
derivata di f in x=5.0:  20.99999999999994
derivata di sin(60.0) : 0.5000001562093738
```

Nella precedente definizione della derivata abbiamo aggiunto la possibilita' di determinare in input con una specifica keyword il valore dell'intervallo $h$ su cui calcolare la derivata. Se la keuyword non viene data esplicitamente verra' assunto il valore di default definito prima.

Passiamo ora ad implementare la derivata usando il risultato analitico:

$$f' \simeq \frac{f_h - f_{-h}}{2h}$$

ricavato a lezione a partire dall'espansione in serie di Taylor.

In [89]:
```
def derivative_alt(f,x,h=None):
    '''
    calcolare la derivata numerica di una funzione (esterna) f
    rispetto ad una variabile x.
    INP   f   funzione da derivare
          x   punto in cui calcolare la derivata
          h   intervallo da usare per il calcolo
              (intorno al quale la f non deve oscillare)
    '''
    #Es: definisci una funzione esterna p.es:
    #         def f(x): return x*x
    # (una migliore implementazione dovrebbe valutare
    # l'incremento h sulla base della variabilita' della
    # funzione nell'intorno di x)
    if h is None:
        # Notare che il valore di default attribuito qui
        # all'incremento h corrisponde alla radice quadrata
        # della precisione in floating point, questo valore
        # si puo' ottenere chiamando: np.sqrt(np.finfo(float).eps)
        h = 1.49011611938477e-08

    deltaf=f(x+h/2.)-f(x-h/2.)
    if f.__name__ is "sinus":  # se abbiamo la funzione seno
        deltax=radians(h)      # esprimiamo deltax in radianti
    else:
        deltax=h

    slope =deltaf/deltax
    return slope
```

```
In [109]: # proviamo a calcolare i due casi con derivative e derivative_alt:
          x=5.
          theta=60.
          step=0.01
          print("derivata di f in x="+str(x)+': ',derivative(f, x,h=step))
          print("derivata di sin("+str(theta)+') :' ,derivative(sinus, theta,h=step
          print("derivata_alt di f in x="+str(x)+': ',derivative_alt(f, x,h=step))
          print("derivata_alt di sin("+str(theta)+') :' ,derivative_alt(sinus, thet

derivata di f in x=5.0:  0.199800266267
derivata di sin(60.0) : 0.4999244224879116
derivata_alt di f in x=5.0:  0.200000066667
derivata_alt di sin(60.0) : 0.4999999993657663
```

Per estendere l'uso del codice scritto finora ad altre funzioni bastera' ridefinire la funzione che daremo in input:

```
In [108]: def f(x):
              # cambio definizione della seconda funzione
              return np.log(x) # e' il log_naturale di x
```

e ripetendo la chiamata precedente:

```
In [112]: # proviamo a calcolare con la f appena modificata i due casi:
          # 1) con la function: derivative
          # 2) con la function: derivative_alt
          x=5.
          step=0.01
          print("derivata di f in x="+str(x)+': ',derivative(f, x,h=step))
          print("derivata_alt di f in x="+str(x)+': ',derivative_alt(f, x,h=step))

derivata di f in x=5.0:  0.199800266267
derivata_alt di f in x=5.0:  0.200000066667
```

```
In [ ]:
```