nb4-integrazione_numerica

October 29, 2017

Integrazione

Questo notebook e' dedicato al calcolo numerico dell'integrale di una funzione di una variabile. Nelcorso delle lezioni ci limiteremo a discutere tre possibili metodi: - il metodo dei trapezi - il metodo di Simpson - Il metodo di Gauss

1 Metodo dei trapezi

In particolare qui discutiamo i primi due, iniziando dal metodo dei trapezi che si basa essenzialmente su una suddivisione, piu' o meno fine, dell'intervallo di integrazione con punti equidistanti. Una volta suddiviso l'intervallo si tratter'a di usare, in ognuno dei sottointervalli che si vengono a determinare, una semplice approssimazione lineare alla funzione da integrare.

L'integrale viene cosi' calcolato come la somma delle aree dei trapezi che corrispondono ai sottointervalli prestabiliti.

```
In [2]: def integral(f,a,b,nsteps=None,tol=1e-3):
            '''  Calcolo l'integrale della funzione f nell'intervallo [a,b]
                 eseguendo un numero di passi determinato dalla accuratezza
                 (espressa in input da 'tol') con cui si desidera calcolare
                 l'integrale.
                 Metodo: l'integrale viene calcolato due volte. La prima volta
                         come somma delle aree di 'nsteps' rettangoli adiacenti
                         che approssimano la funzione integranda.
                         La seconda volta raddoppiando il numero dei rettangoli.

                         Confrontando i due risultati e tenendo conto della tollera
                         'tol' richiesta, siamo in grado di decidere se aumentare i
                         numero di rettangoli usati oppure fermare il calcolo.
            INP:   f       funzione gia' definita
                   a,b     limiti di integrazione
                   nsteps  n. di suddivisioni iniziali dell'intervallo [a,b]
                           (nsteps puo' aumentare per raggiungere la tolleranza ric
                   tol     tolleranza accettabile
            OUT:   valore dell'integrale
            '''
```

```
    #
    if nsteps is None: nsteps=50   # inizializzo
    integrale=0.
    base= float(b-a)/nsteps
    for i in range(nsteps):
        altezza=f(a+i*base)
        area=base*altezza # area di un rettangolino
        integrale=integrale+area

    integrale2=0.
    base2     =base/2.
    nsteps2   =nsteps*2
    for i in range(nsteps2):
        altezza=f(a+i*base2)
        area=base2*altezza # area di un rettangolino
        integrale2=integrale2+area

    accu=(integrale2-integrale)/integrale
    if accu <= tol:
        print("integrale di f tra {} e {}: {}   tolleranza:{}  npassi: {}".
        return integrale2
    else:
        # la funzione chiama se stessa, ma con un numero
        # di passi (e quindi rettangoli) raddoppiato
        integral(f,a,b,nsteps2,tol)
```

Per usare il codice appena preparato e' ora necessario specificare la funzione integranda. Qui di seguito usiamo $f = x^3$, ma lo studente modifichi la definizione per rappresentare anche altre funzioni.

```
In [3]: def f(x=None):
            ''' Definizione della funzione che verra` poi usata
                negli esercizi che seguono sugli integrali.
            '''
            import math
            if x is None:
                print("In f va specificata l'ascissa x")
            else: return x*x*x    # math.pow(x,3.4)


In [5]: integral(f,0,3,nsteps=1000,tol=1e-5)

integrale di f tra 0 e 3: 20.24984179718396   tolleranza:7.81257630401808e-06  npas
```

Ora che abbiamo preso dimestichezza con il calcolo di un integrale con il metodo dei trapezi, possiamo esplorare le differenze che si incontrano utilizzando il # Metodo di Simpson

Ricordiamo che il metodo di Simpson e' **esatto** per polinomi fino al grado 3. Il motivo e' legato al fatto che l'errore di approssimazione per questo metodo va come $h^5\ f''''$ e la derivata quarta di un polinomio di grado 3 e' nulla.

Prepariamo ora una funzione che possa mettere in pratica questo metodo:

In [6]:
```
def adaptive_simpson( f, a, b, tol=1e-3, recur=False):
    """
    Calcola l'integrale di f(x) sull'intervallo [a,b].

    USO:
        s = adaptive_simpson( f, a, b, tol )

    INPUT:
        f          - funzione da integrare
        a, b       - limite sinistro e destro dell'intervallo di integra
        tol        - (opzionale) tolleranza accettabile sull'approssimaz
        recur      - (opzionale) se la function viene chiamata da se ste

    OUTPUT:
        float      - valore dell'integrale

    NOTE:
        Integra la funzione f(x) sull'intervallo [a,b] con un errore
        massimo pari a 'tol', usando in modo adattivo la regola di Simp
        per approssimare il valore dell'integrale. Questa implementazio
        non e' la piu' efficiente perche' calcola ripetutamente il
        valore della funzione nello stesso punto. D'altra parte la
        scrittura cosi' appare piu' chiara e comprensibile ed e' piu'
        facile capire le idee che stanno dietro ad uno schema "adattivo
    """
    # Se con 1 e 2 indichiamo rispettivamente l'approssimazione ottenut
    # con un passo h ed h/2 si ha che, per una f(x) derivabile 4 volte,
    # l'errore assoluto al passo 2 nel metodo di Simpson va come
    #                    e2 = |J1-J2)| / 15
    # dove J1 e J2 sono le approssimazioni ottenute al passo 1 e 2.
    # (wikipedia: regola di cavalieri-simpson).
    import time
    FileName='adaptive_simpson_results.txt'
    accumulatore=0.
    if recur is False:
        outFile = open(FileName, "w") # "a" per appendere
        outFile.write('# ------------ ' +time.strftime("%d/%m/%Y")+'
        outFile.close()
        accumulatore=0.

    tol_factor = 15.0    # fattore moltiplicativo per valutare la tolle

    h = 0.5 * ( b - a )  # h rappresenta la meta' dell'intervallo di in
                         # il cui centro e' rappresentato dal punto x2
    x0 = a
    x1 = a + 0.5 * h
```

```
    x2 = a + h
    x3 = a + 1.5 * h
    x4 = b

    f0 = f( x0 )
    f1 = f( x1 )
    f2 = f( x2 )
    f3 = f( x3 )
    f4 = f( x4 )

    # sull'intervallo [a,b] calcolo l'integrale in due possibili appros
    # che corrispondono ad usare 3 e 5 punti (e quindi due diversi valo
    s0 = h * ( f0 + 4.0 * f2 + f4 ) / 3.0                              # a
    s1 = h * ( f0 + 4.0 * f1 + 2.0 * f2 + 4.0 * f3 + f4 ) / 6.0       # a
    # se si usano 5 punti l'intervallo h e' dimezzato rispetto al caso
    # per questo la divisione nel caso a 5 punti e' per 6 invece che pe
    # Ora confronto i due risultati per decidere se conviene suddivider
    # ulteriormente l'intervallo in base alla tolleranza raggiunta:
    if abs( s0 - s1 ) >= tol_factor * tol:
        s = adaptive_simpson( f, x0, x2, 0.5 * tol, recur=True ) + \
            adaptive_simpson( f, x2, x4, 0.5 * tol, recur=True )
    else:
        s = s1 + ( s1 - s0 ) / 15.0    # ad s1 si aggiunge la stima del
        print("integrale di f tra {} e {}: {}  con tolleranza: {} ".for
        outFile = open(FileName, "a") # "a" per appendere
        outFile.write(str(a)+',  '+str(b)+',  '+str(s)+',  '+str(abs( s
        outFile.close()

    return s, abs( s0 - s1 )
```

Proviamo ora ad usare la funzione appena definita e proviamo anche a modificare la f in modo da verificare che il calcolo e' esatto, e non dipende dalla tolleranza scelta, quando il polinomio da integrare e' di grado 3 o minore.

```
In [8]: int,tolleranza=adaptive_simpson( f, 0, 3, tol=1e-5)

integrale di f tra 0 e 3: 20.25  con tolleranza: 0.0  0.0
```

Usando la funzione **adaptive_simpson** puo' succedere che, quando il grado della funzione integranda e' maggiore di 3, per raggiungere il livello di precisione richiesto l'integrale sia stato suddiviso in piu' parti e per ogni parte il risultato sia stato scritto in un file esterno. In questo caso, leggendo i valori riportati nel file e sommando i contributi dei vari pezzi calcoliamo il risultato finale dell'integrale:

```
In [28]: # leggiamo il contenuto del file con i risultati:
         fname='adaptive_simpson_results.txt'
         with open(fname,'rb') as file:
             lines = file.readlines()    # ogni riga viene letta come un'unica stri
```

```
        # qui usiamo la "," per suddividere la linea nelle sue componenti
        # elements = np.genfromtxt(lines[-5:],delimiter=',',skip_header=1) # legge
        elements = np.genfromtxt(lines[:],delimiter=',')
        if len(elements.shape) == 2:
            # se elements e' 2D allora bisogna somare integrali parziali
            # gli integrali parziali sono stati scritti in colonna 2:
            # sommiamo sulla colonna 2 per avere il risultato finale
            print('Somma dei valori ottenuti per gli integrali parziali:', sum(ele
```

```
Somma dei valori ottenuti per gli integrali parziali: 121.5
```

2 Ancora sul metodo di Simpson

Per rendersi conto di come ci possano essere piu' possibili soluzioni computazionali allo stesso problema esploriamo ora un'altra implementazione della regola di Simpson in modo adattivo (adattato da Wikipedia)

```
In [9]: def simpsons_rule(f,a,b):
            # integrale con la formula di Simpson a 3 punti:
            c  = (a+b) / 2.0
            h  = abs(b-a) / 2.0             # come definito a lezione
            return h/3.0*(f(a) + 4.0*f(c) + f(b))

        def recursive_asr(f,a,b,eps,whole):
            """ Data una funzione f calcola l'integrale tra i due limiti
                (a,b) dividendo l'intervallo in due parti. Su ogni parte
                usa la regola di Simpson e verifica se la somma dei due
                integrali cosi' ottenuti e' confrontabile con il risultato
                ottenuto usando il solo intervallo totale. Un confronto tra
                l'integrale totale e la somma dei due integrali parziali
                decide poi se iterare nella suddivisione dell'intervallo di
                integrazione per raggiungere l'accurartezza richiesta.
            """
            c = (a+b) / 2.0                 # punto di mezzo tra (a,b)
            left = simpsons_rule(f,a,c)     # integrale lato sinistro
            right = simpsons_rule(f,c,b)    # integrale lato destro
            if abs(left + right - whole) <= 15*eps:
                return left + right + (left + right - whole)/15.0
            return recursive_asr(f,a,c,eps/2.0,left) + recursive_asr(f,c,b,eps/2.0,

        def adaptive_simpsons_rule(f,a,b,eps):
            "Calculate integral of f from a to b with max error of eps."
            return recursive_asr(f,a,b,eps,simpsons_rule(f,a,b))

        from math import sin
        print (adaptive_simpsons_rule(f,0,3,0.00001))
```

```
20.25
```

In [ ]: