Quadratures Adaptatives

Pol Tibau
Universitat Politècnica de Catalunya

November 27, 2023

0.1 Introducció

En aquesta pràctica tractarem d'utilitzar mètodes numèrics per aproximar integrals, en concret després d'intentar-ho amb quadratures compostes definirem una quadratura adaptativa, la qual tindrà més subintervals depenent de com varia la funció durant el domini d'integració. La integral amb la que treballarem és:

$$\int_0^2 sin(e^{2x})\,dx$$

El valor de la integral calculat numèricament és: 0.3159042850900473

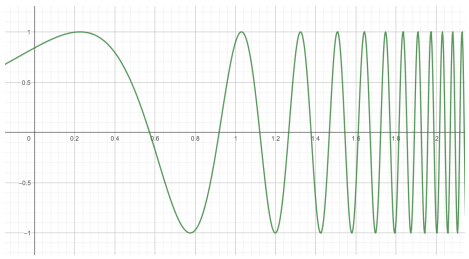


Figure 1: Funció a integrar

Prèviament, abans de començar, tenim un seguit de funcions que necessitarem usar durant el nostre codi. Les dues primeres ens troben els punts equiespaiats d'un interval donat i quins han de ser els pesos per cada punt, i les dues segones ens permeten trobar la fòrmula del trapezi i de Simpson, usant les dues primeres:

```
def trobaPunts(n,a,b):
    X = np.zeros(n+1)
    for i in range(n+1):
        X[i] = a + (i/(n))*(b-a)
    return X


def trobaPesos(n, a, b, X):
    #Definim la matriu del sistema:
    B = np.ones((n+1,n+1))
    for i in range(1,n+1):
        B[i] = X**i

    #Definim el vector solucio:
    S = np.zeros(n+1)
    for i in range(n+1):
        S[i] = (b**(i+1)-a**(i+1))/(i+1)

    #Resolem el sistema per trobar els pesos
```

```
    W = np.linalg.solve(B,S)
    return W
def trapeziCompost(f,a,b,m):
    h = (b-a)/m
    #Tindrem un vector de vectors de pesos per cada seccio W
    W = np.zeros((m,2))
    X = np.zeros((m,2))
    for i in range(m):
        X[i] = trobaPunts(1, a + i*h, a + (i+1)*h)
        W[i] = trobaPesos(1, a + i*h , a + (i+1)*h, X[i])
    return calcula(W,X,f)

def SimpsonCompost(f,a,b,m):
    h = (b-a)/m
    #Tindrem un vector de vectors de pesos per cada seccio W
    W = np.zeros((m,3))
    X = np.zeros((m,3))
    for i in range(m):
        X[i] = trobaPunts(2, a + i*h, a + (i+1)*h)
        W[i] = trobaPesos(2, a + i*h , a + (i+1)*h, X[i])
    return calcula(W,X,f)


def calcula(W,X,f):
    integral = 0
    for i in range(len(W)):
        for j in range(W[0].size):
            integral += W[i][j]*f(X[i][j])
    return integral
```

0.2 Aproximació amb fòrmules compostes del trapezi i Simpson

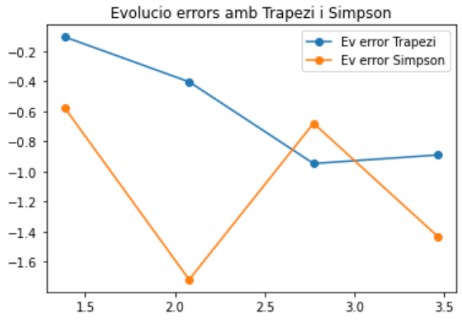
Veurem primer quines aproximacions aconseguim obtenir mitjançant les fòrmules compostes més senzilles. El nostre objectiu és aconseguir 6 xifres significatives. Farem 4 proves amb 4, 8, 16 i 32 subintervals entre 0 i 2.

Amb el mètode del trapezi obtenim com aproximacions:
$[1.1027, 0.7115, 0.4292, 0.1867]$

Amb el mètode de Simpson obtenim com aproximacions:
$[0.5811, 0.3350, 0.1060, 0.3529]$
(Arrodonint tots els resultats al cinquè decimal)

És clar que no ens acostem als objectius desitjats, ja que no som ni capaços de donar dues xifres significatives en els resultats obtinguts.

Podem observar com tots dos mètodes tenen un error que augmentant el nombre d'intervals en tendència general es redueix. Tot i que tenim una funció



molt poc adient per estimar l'error de l'aproximació sense calcular-la explícitament, procedim a mirar si es satisfan les fites que podem aconseguir.

Prenent les fòrmules indicades per cada mètode, obtenim que per 4,8,16 i 32 subintervals:

Amb el mètode del trapezi obtenim com aproximacions:
$[124.207, 31.051, 7.763, 1.941]$

Amb el mètode de Simpson obtenim com aproximacions:
$[6170.9101, 385.681, 24.1051, 1.5066]$

Pel primer cas hem pres com a fita de la segona derivada $8e^8$, pel segon hem pres per fitar la quarta derivada $16e^{16}$, al variar tant les derivades entre 0 i 2, ens veiem obligats a prendre aquestes fites tant elevades, tot i que els errors que tenim son molt mes petits com podem comprovar.

No obstant, si prenem un nombre de subintervals considerable com m=128, veurem que l'error ja és més realista. Amb les fòrmules obtenim que els errors són fitats per [0.1213, 0.0059], mentre que els errors reals respecte la integral són [0.000895 1.27552e-05]

0.3 Estimem els subintervals necessaris per la convergècia desitjada

Veient que amb els nombres de subintervals escollits anteriorment quedem lluny de les xifres significatives que volem aconseguir, usarem de nou les fòrmules d'estimació de l'error, per trobar quants subintervals hem de prendre. En tal cas podriem imposar que volem un error menor de $5 \cdot 10^{-6}$. Considerem l'error fitat per 128 subintervals calculat anteriorment. Aleshores, per cada mètode tenim:

Trapezi: $0.000895 \approx E_{128} = \frac{k}{128^2} \implies k = 128^2 \cdot 0.000895 = 14.664 \implies$
$m* = \sqrt{\frac{14.664}{0.5 \cdot 10^{-6}}} = 5415.5332$

Simpson: $1.27552e-05 \approx E_{128} = \frac{k}{128^4} \implies k = 128^4 \cdot 1.27552e-05 =$

$3423.9479 \implies m* = \sqrt{\frac{3423.947}{0.5 \cdot 10^{-6}}} = 287.6665$

De manera que podem estimar que amb 5416 subintervals pel mètode del trapezi i 288 subintervals pel mètode de Simpson obtindrem la precisió desitjada en el càlcul de la integral.

Per comprovar-ho, si iterem els mètodes afegint un subinterval a cada iteració fins arribar a la fita esperada, malgrat hem d'esperar una bona estona, observem que amb 5264 subintervals pel mètode del trapezi i 282 subintervals pel mètode de Simpson ja obtenim les 6 xifres significatives desitjades, fet que prova l'encert de les estimacions anteriors. El codi usat és el següent:

```
AproxT = False
AproxS = False
compteT = 32
compteS = 32

while(not AproxT):
    t = trapeziCompost(f, a, b, compteT)
    if(abs(t - scipy.integrate.quad(f,a,b)[0]) < 0.5e-6):
        AproxT = True
    else:
        compteT += 1

while(not AproxS):
    s = SimpsonCompost(f, a, b, compteS)
    if(abs(s - scipy.integrate.quad(f,a,b)[0]) < 0.5e-6):
        AproxS = True
    else:
        compteS += 1
```

0.4 Implementació d'una quadratura adaptativa

Després de provar amb les quadratures compostes i veient com d'agressiva es la funció en el domini d'integració, especialment en l'interval (1,2), tractarem d'implementar un algorisme recursiu que en funció de l'error que aparegui en cada subinterval, prendrà més subintervals dintre d'ell, mentre que si d'entrada en una secció ja tenim un error decent no hi aprofunditzarem més.D'aquesta manera podrem reduir notablement el nombre de subintervals ja que enlloc de subintervals equiespaiats, aquests seran escollits molt més eficientment. L'algorisme implementat és el següent:

```
#EXERCICI 4
#Plantegem una quadratura adaptativa,
def SimpsonAdaptativa(f,a,b,eps):
    I = SimpsonCompost(f, a, b, 1)
    II = SimpsonCompost(f, a, (a+b)/2, 1)
```

```
III = SimpsonCompost(f, (a+b)/2, b, 1)
Eab = abs(I - (II + III))
if(Eab < eps*(b-a)):
    return I
else:
    return SimpsonAdaptativa(f, a, (a+b)/2,eps) +
           SimpsonAdaptativa(f, (a+b)/2, b, eps)
```

Vegem com amb aquest mètode aconseguim una aproximació amb un error absolut (estimat) menor que 2ϵ

Suposant que només tinguem 1 sol interval, és fàcil veure que l'error serà menor de $\epsilon \cdot (b-a) = 2\epsilon$

En cas de tenir més intervals, observem que per cada "adentració" dins d'un interval, obtenim dos subintervals de distància 1/2 respecte l'anterior, i de nou al trobar-ne la quadratura es passarà pel barem de mirar si $E < \frac{b-a}{2n} \cdot \epsilon$. Si desfem la recursió sumant totes les cotes d'errors obtenim un sumatori de tots els termes multiplicats per epsilon, del que en podem treure factor comú i fitar-ho com $(b-a) \cdot \epsilon$:

$E_{ab} < \sum_{i=1}^{m} E_{a_i b_i} < \sum_{i=1}^{m} \epsilon \cdot \frac{b-a}{2i} \cdot k_i = \epsilon \cdot \sum_{i=1}^{m} \frac{b-a}{2i} \cdot k_i = \epsilon \cdot (b-a)$

On k_i és el nombre de vegades que apareix cada factor $\frac{b-a}{2i}$. k_i Sempre valdrà com a molt $2i$, ja que en cas contrari sobrepassaríem el tamany de l'interval inicial (a,b) al sumar els subintervals obtinguts.

0.5 Càlcul de la integral amb l'algorisme recursiu (quadratura adaptativa)

Usarem l'algorisme anterior per calcular la integral, primer amb un error de 10^{-3} i seguidament per un error de 10^{-6}. Introduirem a l'algorisme que $\epsilon_1 = 0.5 \cdot 10^{-3}$ i $\epsilon_2 = 0.5 \cdot 10^{-6}$, ja que com hem demostrar el nostre algorisme assegura una aproximació amb un error absolut esimat menor que 2ϵ. Executant l'algorisme obtenim la resposta següent:

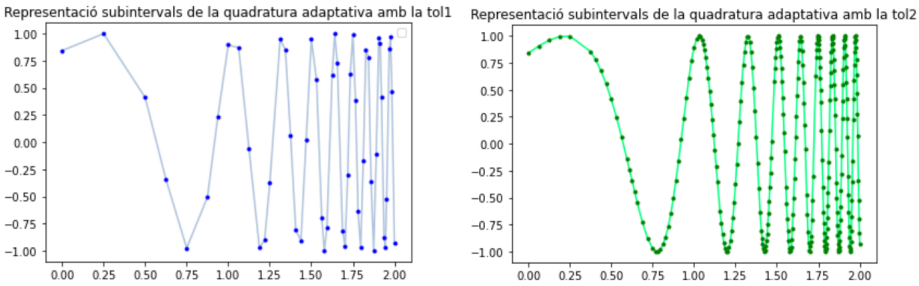
```
Usant un m tode recursiu i una toler ncia de 1e-3 obtenim  0.31601409736230845
Usant un m tode recursiu i una toler ncia de 1e-6 obtenim  0.31590419564785666
El primer resultat t  un error absolut de: 0.0001098122822511538
El segon resultat t  un error absolut de: 8.943220064505653e-08
```

En efecte obtenim els errors desitjats, de fet en el segon cas fins i tot aconseguim un error molt menor del que exigíem.

0.6 Algorisme recursiu modificat

Per veure més clarament el funcionament de les quadratures adaptatives, modificarem el nostre algorisme recursiu per tal que mentre calcula la quadratura

també emmagatzemi en un vector tots els punts entre [a,b] que generen els subintervals. Finalment veurem plasmat en el gràfic de la funció que integrem els punts escollits. Aplicant l'algorisme obtenim:



Podem veure com tots dos gràfics acumulen un percentatge de punts més alt per una x propera a 2. Això es deu a que la funcio oscil·la molt més i per tant al prendre els subintervals és molt més probable que apareixi un error mes considerable. Per contrarrestar aquest fet amb les quadratures adaptatives aqui prenem més subintervals per poder aconseguir la precisió desitjada més eficientment, sense prendre subintervals equiespaiats minimitzant l'error de parts de la funció on ja tenim una aproximació prou bona.

0.7 Comparació amb quadratures compostes

Finalment, si observem el tamany del vector de punts, veiem que imposant que volem un error de 10^{-6} la longitud d'aquest es 334, i per tant tenim un total de 333 subintervals. Imposant que volem aquest error amb quadratures de Simpson compostes no adaptatives, obtenim que necessitaríem un total de 237 intervals de Simpson, que realment son un total de 474 subintervals (ja que tenim un punt al mig de cada interval de Simpson compost).

D'una forma similar, imposant que volem un error e 10^{-3}, obtenim que necessitem 59 subintervals. Si tal i com acabem de fer imposem aquest error per quadratures de Simpson adaptatives, resulta que necessitem un total de 53 subintervals, que igual que abans es tradueixen en 107 punts i 106 subintervals en el mateix sentit que la quadratura adaptativa.

Així doncs, podem concloure clarament que les quadratures adaptatives ens permeten aconseguir amb un nombre molt reduït de subintervals el mateix resultat que la quadratura de Simpson estàndar.