Programmation Scientifique 2 - TP1 - Quadrature de Gauss-Legendre

Yann LE GUILLY

February 20, 2016

1 Introduction

La quadrature de Gauss-Legendre est une méthode numérique permettant d'approcher la valeur d'une intégrale. Le calcul de l'intégrale lui-même est remplacé par une somme pondérée prise en un certain nombre de points du domaine d'intégration. La méthode de Gauss-Legendre est exacte pour un polynôme de degré $3n-1$ avec n points.

Nous allons, dans le cadre de ce TP, mettre en œuvre cette méthode en langage FORTRAN. Nous avons écrit un MAKEFILE afin de faciliter la compilation et séparé les fonctions/sous-routines dans le fichier $functions.f90$ pour une maintenance future du code plus aisée.

2 Quadrature de Gauss-Legendre: généralités

La valeur de l'intégrale que nous cherchons à calculer est donnée par:

$$I = \int_a^b f(x)dx \approx \sum_{k=1}^{n} \omega_k f(x_k)$$

où x_k sont les racines du polynôme de Legendre $P_n(x)$ et ω_k sont les poids de la quadrature.

Dans le cadre de ce TP, nous prendrons:

$$\omega_k = \frac{2}{(1-x_k^2)[P_n'(x_k)]^2}$$

avec $P_n'(x_k)$ la dérivée de $P_n(x)$ en $x = x_k$.

3 La sous-routine GL_xw

Cette sous-routine sera chargée de mettre en place les éléments de base afin d'évaluer la quadrature. Ainsi, nous allons, dans un premier temps, calculer les racines des polynômes de Legendre $P_n(x)$ par la méthode de Newton-Raphson:

$$x_k = x_i - \frac{P_n(x_i)}{P_n(x_i)'}$$

Cependant, en commençant à un x_i prit au hasard, nous risquons de nous retrouver éventuellement sur un extremum local, voire de ne pas converger du tout. Nous allons donc réaliser une première approximation de la racine:

$$x_0 = \cos(\frac{(4k-1)\pi}{4n+2}) \quad (k = 1, ..., n)$$

Nous appliquerons l'algorithme de Newton-Raphson à partir de cette valeur.

Une fois obtenue une racine, on calcule la fonction de poids associée, c'est à dire $\omega_k(x_k)$.

3.1 Pseudo-code de la sous-routine

```
sous-routine GL_xw(n,xk,wk)
espilon=1e-10
pour k=1 jusqu'à n
    x=cos((4*k-1)*pi/(4*n+2))
    pour (boucle infinie)
        P0=1
        P1=x
        pour i=2 jusqu'à n
            P=(((2*i-1)*x*P1-(i-1)*P0))/i
            P0=P1
            P1=P
        fin pour
        dP=(n*P0-n*x*P1)/(1-x*x)
        dx = P/dP
        x=x-dx
        Si abs(dx) < espilon, on interrompt la boucle infinie
    fin pour
    xk(k)=x
    wk(k)=2/((1-x*x)*(dP*dP))
fin pour
fin de la sous-routine GL_xw
```

n est l'ordre de la quadrature et est choisi via un prompt par l'utilisateur ou via une boucle; la valeur d'epsilon représente la valeur pour laquelle nous obtiendrons une dérivée (quasi-)nulle (inférieure à ϵ), auquel cas on estimera avoir trouvé la valeur de la racine la plus précise possible.

Ces valeurs sont stockées dans les vecteurs xk et wk de taille n qui sont à leur tour retournés au programme principal pour être réutilisés plus tard.

4 La sous-routine Quad_GL

Cette sous-routine est beaucoup plus simple dans la mesure où elle se contente d'effectuer la somme:

$$I \approx \sum_{k=1}^{n} \omega_k f(x_k)$$

4.1 Pseudo-code de la sous-routine

```
subroutine Quad_GL(f,a,b,n,xk,wk)
    appel de GL_xw (n, xk, wk)
    gl=0
    pour k=1 jusqu'à n
        Si xk(k)>a ET xk(k)<b alors gl=gl+wk(k)*f(k)
    fin pour
    afficher gl
end subroutine Quad_GL
```

On prend soin de vérifier malgré tout, que xk est bien compris dans l'intervalle $[-1, 1]$. $f(k)$ est un vecteur contenant les valeurs de la fonction f aux n points x_k.

4.2 Résultats obtenus

Nous avons observé les résultats sur des fonctions simples en fonction de l'ordre de la quadrature. Ces résultats sont reportés dans le tableau suivant.

Ordre	x^2	$\exp(x)$	$\exp(-x^2)$	$\sin(x)$
1	7.2E-033	2.0	2.0	1.2246E-016
2	0.66666666666666696	2.3426960879097316	1.4330626211475792	0.0
3	0.66666666666666341	2.3503369286800044	1.4986795956600263	0.0
4	0.66666666666644558	2.3504020921544209	1.4933346224479012	0.0
5	0.66666666665907548	2.3504023864327834	1.4936639206830629	0.0
10	0.66666666656696238	2.3504023871007997	1.4936482655608354	0.0
WolframAlpha	0.666666666666667	2.35040238728760	1.493648265624854	0.0

On peut voir que dès l'ordre 3, la quadrature donne déjà une bonne approximation.

5 La fonction poly

L'objectif de cette fonction est d'évaluer un polynôme. Pour cela, nous pouvons utiliser la méthode "naïve" qui consiste à remplacer x par la valeur souhaitée. Il se peut cependant que dans certains cas il y ait des erreurs d'absorption par exemple. Pour cela, nous allons utiliser la règle de Horner. C'est à dire que pour un polynôme:

$$P = a_n X^n + a_{n-1} X^{n-1} + \ ... \ + a_0$$

La règle de Horner nous donne:

$$P(x_0) = ((...((a_n x_0 + a_{n-1})x_0 + \ ...)x_0 + a_1)x_0 + a_0$$

5.1 Pseudo-code de la sous-routine

```
fonction poly(npoly,cf,x0)
    vérifier que le fichier cf.txt existe
    ouvrir le fichier cf.txt
    lire an et an-1
    poly=an*x0+an-1
    pour i=npoly-2 jusqu'à 0 avec un incrément de -1
        lire an-i
        poly=poly*x+an-i
    fin pour
    fermer le fichier
fin fonction poly
```

Nous avons choisi d'appeler le fichier "cf.txt" un fichier contenant les coefficients du polynôme à évaluer et stocké dans le même répertoire que le programme lui-même. Le programme a donc besoin de ce fichier pour effectuer le calcul.

6 Intégrale sur $[-1, 1]$ et précision

A ce stade, maintenant que nous avons bien compris comment calculer une intégrale via la quadrature de Gauss-Legendre, nous allons directement modifier notre programme principal. Nous appellerons *GL_xw* puis la fonction *poly* et finalement *Quad_GL* afin d'évaluer l'intégrale du polynôme voulu (dont les coefficients sont stockés dans le fichier "cf.txt").

6.1 Résultats obtenus

Dans un premier temps, on considère le polynôme suivant:

$$P = -10x^4 + 7x^3 + 14x^2 - 6x - 50$$

Notre programme nous donne:

Ordre	résultat obtenu
1	-100.00000000000000
2	-92.888888888888943
3	-94.666666666666416
4	-94.666666666577214
5	-94.666666665443159
10	-94.666666660629659
WolframAlpha	-94.666666666666667

La valeur donnée par le moteur de recherche WolframAlpha semble peu pertinente. On ne sait effectivement rien sur les méthodes numériques employées et la précision associée.

Testons notre code sur un autre polynôme:

$$P = -20x^7 + 10x^6 + 8x^5 - 2x^4 - x^2 + x$$

Ordre	résultat obtenu
1	1.2246467991473530E-016
2	-0.37037037037037057
3	0.93333333333332902
4	1.3904761904763956
5	1.3904761904817424
10	1.3904761901686105
WolframAlpha	1.3904761904761905

6.2 Estimation de la précision

Afin d'évaluer la précision que nous pouvons obtenir, nous allons calculer:

$$I = \int_{-1}^{1} \frac{2}{1+x^2}\, dx = \pi = 3.1415926535897932384...$$

On peut remarquer qu'à l'ordre ≈ 15 on arrive à 3.1415926535. Nous n'arriverons pas à faire mieux. Cependant, si on continue à augmenter l'ordre, le résultat perds en précision pour retrouver 3.1415926530 à l'ordre 187 (sur notre machine tout du moins...). On obtient donc une précision de l'ordre de 10^{-9} dès l'ordre 15 environ.

7 Généralisation sur $[a, b]$ et validation du code

La quadrature de Gauss-Legendre ne nous permet d'intégrer que sur $[-1, 1]$. Il convient donc, afin de l'adapter à un domaine généralisé $[a, b]$ d'effectuer un changement de variable:

$$I \approx \frac{b-a}{2} \sum_{k=1}^{n} \omega_i f(\frac{b-a}{2} x_k + \frac{a+b}{2})$$

7.1 Validation du code

Afin de valider notre code, nous allons effectuer un calcul sur un résultat préalablement connu:

$$I = \int_{-\pi}^{\pi} \sin^2(t) \, = \pi$$

Le code nous renvoie: ≈ 3.1415926536 à l'ordre 15. On peut donc estimer que le code est validé, il fournit en effet des résultats attendus sur un cas connu et avec une précision connue.

8 Conclusion

Ce TP nous aura permis de mieux comprendre la quadrature de Gauss-Legendre. En effet, nous avons pu expérimenter la mise en œuvre dans un cas simple puis nous l'avons généralisé. De plus, nous aurons effectué la validation du code qui constitue une partie essentielle et absolument nécessaire dans ce type de projet.