Récursion à la main et passage de fonction en argument en Fortran 95

Nicolas Depauw

5 octobre 2011

Une *méthode de quadrature* est un algorithme pour calculer une valeur approchée d'une intégrale $I = \int_a^b f(x)\,dx$. Dans ce troisième exemple, nous implantons les trois méthodes usuelles de quadrature : Trapèzes, Simpson et Romberg.

L'intervalle $[a, b]$ est successivement subdivisé en 2, 4, 8,..., 2^n sous-intervalles d'égales longueurs. On peut ainsi passer du rang n au rang $n+1$ en prenant en compte les évaluations de f déjà calculées.

Ces trois méthodes sont implantées dans des sous-routines qui reçoivent en argument la fonction f à intégrer (et aussi, bien sûr, les bornes de l'intervalle, *etc*).

Le programme teste ces trois méthodes pour l'intégrale $\int_0^2 x^4 \ln(x + \sqrt{x^2+1})\,dx$. Cet exemple est tiré du livre Numerical Recipes, 3rd ed publié par Cambridge University Press .

1 L'itération de base

Pour $n \geq 1$, notons T_n et M_n les valeurs approchées de I par la méthode des trapèzes et par celle du point milieu sur N intervalles quand précisément $N = 2^{n-1}$, :

$$T_n = h\left(\frac{1}{2}f(x_0) + \sum_{j=1}^{N-1} f(x_j) + \frac{1}{2}f(x_N)\right), \qquad x_j = a + hj \text{ pour } j \text{ de } 0 \text{ à } N,$$

$$M_n = h\sum_{j=1}^{N} f(y_j), \qquad y_j = \frac{1}{2}(x_{j-1} + x_j) \text{ pour } j \text{ de } 1 \text{ à } N,$$

avec $h = \frac{(b-a)}{N}$. (La somme $\sum_{j=1}^{N-1} f(x_j)$ est vide donc nulle quand $n = 1$ et donc $N = 1$). Noter que N, h, les x_j et y_j dépendent de n.

Théorème 1 *$T_n = \frac{1}{2}(T_{n-1} + M_{n-1})$ pour $n \geq 2$.*

Dans cette formule on lit que pour passer de T_{n-1} à T_n il suffit d'évaluer f aux seuls $y_j = a + h(j - \frac{1}{2})$ pour j de 1 à N intervenant dans la formule de M_{n-1} où $h = \frac{(b-a)}{N}$ avec $N = 2^{n-2}$.

On implémente cette récursion de la forme T_n fonction de T_{n-1} et de n, a, b, f dans la sous-routine `trapiter`. On veut que

1. l'appel `call trapiter(T,1,a,b,f)` affecte à `T` la valeur $T_1 = (b-a)\frac{1}{2}(f(a)+f(b))$;
2. si `n` contient la valeur n, avec $n \geq 2$, et `T` contient la valeur T_{n-1},
   l'appel `call`
   `trapiter(T,n,a,b,f)`, affecte à `T` la valeur T_n.

1 ⟨*trapiter* 1⟩≡ (7)

```
subroutine trapiter(T,n,a,b,f)
  ⟨declarations dans trapiter 2⟩
  if (n==1) then
     T=.5*(b-a)*(f(a)+f(b))
  else
     nbp=2**(n-2)
     h=(b-a)/nbp
     somme=0.
     do j=1,nbp
        somme=somme+f(a+(j-.5)*h)
     end do
     T=.5*(T+somme*h)
  end if
end subroutine trapiter
```

Les déclarations vont de soi, sauf pour l'argument `f` qui réfère à une fonction. On doit utiliser un bloc d'`interface`.

2 ⟨*declarations dans trapiter* 2⟩≡ (1)

```
real, intent(inout) :: T
integer, intent(in) :: n
real, intent(in) :: a,b
interface
   function f(x)
     real :: f,x
   end function f
end interface
integer :: nbp,j
real :: h,somme
```

2 Les quadratures

Les trois fonctions de quadratures prennent les mêmes arguments dans le même ordre. Le début des déclarations est donc le même. Comme pour tout algorithme itératif, un point essentiel est la condition d'arrêt. On passe pour cela deux arguments en entrée. `er` indique *l'erreur relative souhaitée*. L'itération s'arrête quand la différence entre la nouvelle approximation et la précédente est suffisamment petite par rapport l'approximation précédente. `nmax` est *le nombre maximal d'itérations*. Rappelons que chaque itération multiplie par deux le nombre de sous-intervalles de la subdivision.

En sortie, `er` reçoit l'erreur relative atteinte, et `nmax` le nombre d'itérations effectuées.

3 ⟨*arguments des quadratures* 3⟩≡ (4–6)

```
real, intent(in) :: a,b
interface
   function f(x)
     real :: f,x
   end function f
end interface
real, intent(inout) :: er
integer, intent(inout) :: nmax
```

3 Quadrature par les Trapèzes

Il suffit d'organiser les appels successifs à `trapiter`. On fait au moins trois tours, de sorte que le nombre d'intervalles de la subdivision est au moins 4.

4 ⟨*quatrap* 4⟩≡ (7)

```
function quatrap(a,b,f,er,nmax) result(T)
  ⟨arguments des quadratures 3⟩
  real :: T,Ta
  integer :: j
  call trapiter(T,1,a,b,f)
  call trapiter(T,2,a,b,f)
  j=3
  Ta=T
  call trapiter(T,3,a,b,f)
  do while (abs(T-Ta)>er*abs(Ta) .and. j<nmax)
     j=j+1
     Ta=T
     call trapiter(T,j,a,b,f)
  end do
  er=abs(T/Ta-1)
  nmax=j
end function quatrap
```

4 Quadrature par Simpson

Les coefficients $1, 4, 2, 4, 2, \ldots, 4, 1$ de la quadrature de Simpson s'obtiennent aussi en combinant deux approximations successives des Trapèzes (dont les coefficients sont $1, 2, 2, \ldots, 2, 1$) avec les poids 4 pour la dernière, et -1 pour la précédente. Ainsi vient la formule `S=(4*T-Ta)/3`.

On peut aussi voir cette méthode comme la meilleure utilisation possible de deux termes consécutifs des Trapèzes pour accélérer la convergence : éliminer le premier terme du développement asymptotique de l'erreur $T_n - I = ch^2 + O(h^4)$ avec $h = (b-a)2^{1-n}$, donc $T_n - I = c2^{-2n} + O(2^{-4n})$, et $T_{n-1} - I = c2^{-2(n-1)} + O(2^{-4n})$, ce qui entraîne $(4T_n - T_{n-1})/3 - I = O(2^{-4n})$.

5 ⟨*quasimp* 5⟩≡ (7)

```
function quasimp(a,b,f,er,nmax) result(S)
  ⟨arguments des quadratures 3⟩
  real :: S,Sa,T,Ta
  integer :: j
  call trapiter(T,1,a,b,f)
  Ta=T
  call trapiter(T,2,a,b,f)
  Sa=(4.*T-Ta)/3.
  j=3
  Ta=T
  call trapiter(T,3,a,b,f)
  S=(4.*T-Ta)/3.
  do while (abs(S-Sa)>er*abs(Sa) .and. j<nmax)
```

```
      j=j+1
      Ta=T
      call trapiter(T,j,a,b,f)
      Sa=S
      S=(4.*T-Ta)/3.
   end do
   er=abs(S/Sa-1)
   nmax=j
end function quasimp
```

5 Quadrature par Romberg

Pourquoi se limiter à deux termes successifs comme dans Simpson. On sait que si la fonction f à intégrer est infiniment régulière, on a un développement à un ordre arbitraire : $T_n - I = \sum_{p=1}^{q-1} c_p 2^{-2np} + O(2^{-2nq})$.

Si on dispose de k termes successifs des Trapèzes, (donc de $k-1$ termes successifs de Simpson, *etc*), on peut éliminer $k-1$ termes du développement asymptotique en appliquant $k-1$ fois l'astuce permettant de passer des Trapèzes à Simpson.

La méthode de Romberg exploite au maximum cette possibilité, en éliminant un terme du développement asymptotique à chaque itération supplémentaire de Trapèzes. Mais cela nécessite de mémoriser toutes les approximations de Romberg précédente, pas seulement la dernière. Le tableau `romb` sert à cela : `romb(1)` contient la dernière valeur des Trapèzes, `romb(2)` la dernière valeur de Simpson, *etc*, et `romb(j)` la dernière valeur la meilleure de Romberg.

6 ⟨*quaromb* 6⟩≡ (7)

```
function quaromb(a,b,f,er,nmax) result(R)
  ⟨arguments des quadratures 3⟩
  real :: R,Ra,t
  real, dimension(nmax) :: romb
  integer :: j,k
  call trapiter(romb(1),1,a,b,f)
  Ra=romb(1)
  call trapiter(romb(1),2,a,b,f)
  romb(2)=(4.*romb(1)-Ra)/3.
```

```
  j=3
  Ra=romb(1)
  call trapiter(romb(1),3,a,b,f)
  R=(4.*romb(1)-Ra)/3.
  Ra=romb(2)
  romb(2)=R
  romb(3)=(16.*romb(2)-Ra)/15.
  do while (j<nmax .and.&
       & abs(romb(j)-romb(j-1))>er*abs(romb(j-1)))
     j=j+1
     Ra=romb(1)
     call trapiter(romb(1),j,a,b,f)
     do k=1,j-1
        t=4.**k/(4.**k-1)
        R=t*romb(k)-Ra/(4.**k-1)
        Ra=romb(k+1)
        romb(k+1)=R
     end do
  end do
  er=abs(romb(j)/romb(j-1)-1)
  nmax=j
end function quaromb
```

6 Le module quadratures

Ce module contient les sous-routines de quadratures.

7 ⟨*quadratures.f95* 7⟩≡

```
module quadratures
  implicit none
contains
  ⟨trapiter 1⟩
  ⟨quatrap 4⟩
  ⟨quasimp 5⟩
  ⟨quaromb 6⟩
end module quadratures
```

7 Le modules `fonctions`

Ce module contient les fonctions définies *à la main* que l'on veut intégrer.
Pour rester simple, il contient seulement $f_1(x) = x^4 \ln(x + \sqrt{x^2 + 1})$.

8 ⟨*fonctions.f95* 8⟩≡

```
module fonctions
  implicit none
contains
  function f1(x)
    real, intent(in)::x
    real::f1
    f1=x**4*log(x+sqrt(x**2+1))
  end function f1
end module fonctions
```

8 Le programme `test`

On sait trouver, par intégration par partie, une primitive de la fonction f_1. On en déduit que $\int_0^2 f_1(x)\,dx = \frac{32}{5}\log(2+\sqrt{5}) - \frac{8}{15}(\sqrt{5}-2)$. Le programme compare les valeurs approchées obtenues par les quadratures à cette valeur exacte. Pour chacune des trois quadratures, les paramètres d'appels sont les mêmes : une erreur relative proche de la précision machine (`1.e-6` à comparer aux 23 bits de la mantisse) ; et un maximum d'itérations raisonnable (`NMAX=20` ce qui signifie au plus $1+2^{19}$ évaluation de f_1. À chaque fois on affiche le nombre de tours effectué, la valeur approchée, l'erreur relative de la dernière valeur par rapport à la précédente et l'erreur relative de la dernière valeur par rapport à la valeur exacte.

9 ⟨*test.f95* 9⟩≡

```
program test
  use quadratures
  use fonctions
  implicit none
  integer, parameter :: NMAX=20
  real, parameter :: EPS=1.e-6
  real :: a,b,er,T,S,R,C
  integer :: n
  a=0.;b=2.
  er=EPS;  n=NMAX
  C=(32*log(2.+sqrt(5.))-(sqrt(5.)-.2)*8/3.)*.2
  print*,'calcule',0,C
```

```
  er=EPS;  n=NMAX
  T=quatrap(a,b,f1,er,n)
  print*,'trapeze',n,T,er,abs(T/C-1)
  er=EPS;  n=NMAX
  S=quasimp(a,b,f1,er,n)
  print*,'simpson',n,S,er,abs(S/C-1)
  er=EPS;  n=NMAX
  R=quaromb(a,b,f1,er,n)
  print*,'romberg',n,R,er,abs(R/C-1)
end program test
```

9 Le makefile

9.1 Compilation et assemblage

Cette section décrit le fichier `makefile`, que lit la commande `make`, destiné à compiler et assembler les sources en un exécutable nommé `test`.

10 ⟨*makefile* 10⟩≡ 11 ▷

```
# -*- makefile-mode -*-
WARN=-Wall -Wextra
COMP=gfortran $(WARN)
PROJ=integration
SOURCES=test.f95 quadratures.f95 fonctions.f95
OBJECTS=test.o quadratures.o fonctions.o
MODULES=quadratures.mod fonctions.mod
test : $(OBJECTS)
        $(COMP) -o $@ $+
test.o : test.f95 $(MODULES)
        $(COMP) -c $<
%.o %.mod : %.f95
        $(COMP) -c $<
```

9.2 Et encore de la programmation documentée

Nous rajoutons dans le `makefile` les commandes pour produire d'une part les sources en FORTRAN 95 (`*.f95`) et d'autre part cette documentation pdf (`modele.pdf`) à partir d'un unique fichier `modele.nw`, texte au format `noweb`. Pour obtenir le fichier `pdf` il suffira de lancer la commande `make doc`.

11 ⟨*makefile* 10⟩+≡ ◁10

```
# -*- makefile-mode -*-
$(SOURCES) : $(PROJ).nw
        notangle -R$@ $< > $@
$(PROJ).pdf : $(PROJ).tex
        pdflatex $(PROJ)
        pdflatex $(PROJ)
        pdflatex $(PROJ)
$(PROJ).tex : $(PROJ).nw
        noweave -delay -index $<  | sed -e s/-/-/g >$@
.PHONY : doc clean purge source
source : $(SOURCE)

doc :  $(PROJ).pdf
        echo "pour visualiser la doc : xpdf $(PROJ).pdf"
clean :
        rm -f *.log *.aux $(OBJECTS) $(MODULES)
purge :
```

```
        rm -f test *.pdf *.tex *~ $(SOURCES)
makefile : $(PROJ).nw
        notangle -Rmakefile -t2 $(PROJ).nw >makefile
```

A Résumé des éléments de FORTRAN 95 rencontrés

Rappelons que le numéro souligné correspond à la *définition*, c'est-à-dire en fait à la première occurrence, et donc peut-être à une explication.

`interface`: 2, 3

A Bouts de code

⟨*arguments des quadratures* 3⟩ 3, 4, 5, 6
⟨*declarations dans trapiter* 2⟩ 1, 2
⟨*fonctions.f95* 8⟩ 8
⟨*makefile* 10⟩ 10, 11
⟨*quadratures.f95* 7⟩ 7
⟨*quaromb* 6⟩ 6, 7
⟨*quasimp* 5⟩ 5, 7
⟨*quatrap* 4⟩ 4, 7
⟨*test.f95* 9⟩ 9
⟨*trapiter* 1⟩ 1, 7