Premiers pas en FORTRAN 95

Nicolas Depauw

26 septembre 2011

Dans ce petit exemple, nous écrivons un programme en FORTRAN 95 qui résoud les équations du second degré à coefficients réels. Nous détaillons au passage quelques aspects de la syntaxe du langage. Tout le code est tapé avec le jeu de caractères ASCII. En particulier, sans caractères accentués.

1 Structure

Un programme en FORTRAN 95 contient une section de déclarations et une section d'instructions. Par les déclarations, nous indiquons le type de chaque variable. Les instructions décrivent le déroulement des opérations que le programme exécute.

1 ⟨*edsd.f95* 1⟩≡

```
program edsd
  ⟨déclarations 2⟩
  ⟨instructions 5⟩
end program
```

2 La résolution de l'équation

Le problème mathématique consiste à trouver les solutions de $ax^2 + bx + c = 0$ pour trois réels a, b, c. On peut, selon le signe du discrimant Δ, avoir deux solutions réelles x_1, x_2, ou bien deux solutions complexes conjuguées z_1, z_2.

2.1 Déclarations

On déclare donc des variables correspondant à ces nombres, et une indiquant le cas rencontré. On utilise naturellement les types de base `logical`, `real` et `complex`. À cause de la déclaration `implicit none`, le compilateur considère comme une erreur l'usage (pendant les exécutions) d'un identificateur de variable non déclaré au préalable (dans les spécifications). Tant que vous ne serez pas experts, n'oubliez pas cette déclaration, en premier. La variable `delta_positif` est initialisée à vrai, en considérant que c'est le cas le plus courant.

2 ⟨*déclarations* 2⟩≡ (1) 4▷

```
implicit none
logical :: delta_positif=.true.
real :: a,b,c,delta,x1,x2
complex :: z1,z2
```

2.2 Calculs

Le discriminant est donné par $\Delta = b^2 - 4ac$. De son signe dépend si les racines sont réelles ou complexes conjuguées. Tester le signe ou la taille d'une variable `real` n'a pas de sens en soi. Ici on doit décider du signe de Δ par rapport à b : si $|\Delta| < \epsilon\,|b|$, alors on doit considérer Δ comme nul. On choisit ϵ de l'ordre de la précision machine pour le type considéré, donnée par `epsilon(b)` si on considère le type de la variable `b`. Dans le cas de racines réelles, pour éviter une perte de précision par cancellation, on calcule d'abord la racine qui conduit à ajouter deux nombres de mêmes signes. Puis on utilise la relation $ax_1x_2 = c$ pour trouver l'autre. La fonction `sign(x,y)` renvoie un nombre qui a la valeur absolue de x et le signe de y.

3 ⟨*calculs* 3⟩≡ (5)

```
delta=b**2-4*a*c
if (abs(delta)<epsilon(b)*abs(b)*2) then
   x1=-b/2/a
   x2=x1
else if (delta>0) then
   x1=-(b+sign(sqrt(delta),b))/2/a
   x2=c/a/x1
else! (delta<0)
   z1=cmplx(-b, sqrt(-delta))/2/a
   z2=conjg(z1)
   delta_positif=.false.
end if
```

3 Les entrées-sorties

Pour ce premier programme, nous utilisons une interface avec l'utilisateur. L'inconvénient majeur est le temps qu'on risque de consacrer à cette interface, alors que vous devez apprendre à résoudre des problèmes mathématiques. C'est pourquoi dans la suite nous nous interdirons de telles élaborations. Les entrées-sorties des données-résultats se feront par lecture-écriture sur des fichiers. Pour ce petit premier programme, exceptionnellement, on fabrique une boucle qui demande à l'utilisateur s'il veut continuer ou quitter, puis, s'il veut continuer, les trois coefficients a, b, c. Les résultats sont simplement affichés à l'écran. En quittant, le programme affiche en dernier le nombre d'équations résolues. On déclare donc de nouvelles variables de type `character` et `integer`.

4 ⟨*déclarations* 2⟩+≡ (1) ◁2

```
character :: choix
integer :: nbre_equations
```

En FORTRAN 95, l'instruction de contrôle `do...end do` est complètée des instructions `exit`, pour arrêter complètement la boucle en cours et passer à l'instruction suivante ; et `continue`, pour arrêter le tours de boucle en cours et passer au tour suivant.

5 ⟨*instructions* 5⟩≡ (1)

```
nbre_equations=0; choix='c'
do
   print*,"Continuer (c) ou Quitter (q)?"; read*,choix
   select case (choix)
   case ('c')
      print*,"Coefficients ('real') a,b,c?"; read*,a,b,c
      ⟨calculs 3⟩
      ⟨affichage 6⟩
      nbre_equations=nbre_equations+1
   case ('q')
      exit
   case default
      print*,"Merci de repondre par 'c' ou 'q'."
      continue
   end select
end do
print*,"Nous avons resolu ",nbre_equations," equations."
```

L'affichage dépend du signe du discriminant.

6 ⟨*affichage* 6⟩≡ (5)

```
if (delta_positif) then
   print*,"discriminant positif, x1 et x2 :"
   print*,x1,x2
else
   print*,"discriminant negatif, z1 et z2 :"
   print*,z1,z2
end if
```

4 Compilation, exécution

Ayant écrit notre programme, il faut encore le compiler. Sur nos machines, nous utilisons le compilateur GNU `gfortran`. Dans un terminal (ou bien même depuis EMACS, avec `Meta-!`), la ligne de commande pour compiler le fichier source `edsd.f95` en un exécutable `edsd`, en déclenchant les options donnant beaucoup d'avertissements, est

```
gfortran -Wall
```

`-Wextra -o edsd edsd.f95` La ligne de commande pour exécuter est alors

```
./edsd
```

5 Et la programmation documentée ?

En fait le programme `edsd` et ce document qui l'explique ont été produit à partir d'un même fichier source, `edsd.nw`, qui contient tout à la fois les explications (dans l'ordre de lecture choisi par l'auteur pour ses vertus pédagogiques) et le code (découpé en petits morceau pour coller aux explications), dans un format compréhensible par l'outil `noweb`. Pour produire tout le reste, nous regroupons les commandes dans un petit script nommé `mk` (comme *make*).

La première partie consiste à produire le code `edsd.f95`. Puis le fichier LaTeX`edsd.tex`. Que l'on compile avec `pdflatex` tandis qu'on compile le programme avec `gfortran`.

7 ⟨*mk* 7⟩≡

```
notangle -Redsd.f95 edsd.nw > edsd.f95
noweave -delay -index edsd.nw > edsd.tex
pdflatex edsd
pdflatex edsd
gfortran -Wall -Wextra -o edsd edsd.f95
```

Il reste enfin deux *détails*. D'une part, dans un terminal, afficher ce document avec `xpdf edsd.pdf` et lancer l'exécution avec `./edsd`. D'autre part, avant de pouvoir exécuter `./mk`, il aura bien fallu le produire, avec

`notangle -Rmk edsd.nw > mk; chmod 744 mk`.

A Résumé des éléments de FORTRAN 95rencontrés

Nous utilisons la capacité d'indexation de notre outil de programmation documentée (noweb) pour rappeler les mots réservés et les fonctions prédéfinies que nous avons utilisés. Le numéro souligné correspond à la *définition*, c'est-à-dire en fait à sa première occurrence, et donc peut-être à une explication. Noter qu'à cause des `end program`, `end if` et `end do`, nous avons défini `end` ici seulement.

B Bouts de code

Voici, juste pour le plaisir, comment noweb collecte les bouts de code.

C Exercice

1. Compiler et executer le programme, détecter une erreur de programmation.
2. Corriger cette erreur dans le programme, recompiler, tester à nouveau.
3. Sur le même modèle, écrire un programme qui, en boucle, demande un réel x et affiche une valeur approchée de $\exp(x)$ et $\ln(x)$, sans faire appel aux functions intrinsèques `exp` et `log`, mais en utilisant uniquement les opérations arithmétiques usuelles. On cherchera une valeur approchée aussi exacte que possible, sachant qu'*a priori* x sera choisit dans l'intervalle $[1, 2)$.

   Indications
   - On peut utiliser les séries de Taylor $\exp(x) = \sum_{i=0}^{a} x^i/i!$, et $\ln(x) = -\ln(1-u) = \sum_{i=1}^{b} u^i/i$ avec $u = 1 - 1/x \in [0, 1/2)$. Le nombre de termes à sommer est à mettre en regard de la précision relative de l'encodage des réels flottants simple précision : 23 bits dans la mantisse.
   - Dans une somme de termes de mêmes signes variants de façon monotone, les erreurs d'arrondis seront moins grandes si on commence la somme par les petits termes, donc ici ceux de degrés élevés.
   - On économise des instructions en suivant le shéma de Hörner. Par exemple $((((x/5+1)*x/4+1)*x/3+1)*x/2+1)*x+1*$.
   - Pour apprécier les résultats, on pourra les comparer aux valeurs exactes données par exemple par les fonctions intrinsèques `exp` et `log`.