**Cours d'informatique et de calcul scientifique**
**Première année apprentissage, mécanique des fluides - 2018-2019**

# TP : Recherche de zéros par la méthode de Newton.

Durée : 4 h.

## 1 Introduction

Il arrive très couramment, lorsqu'on étudie numériquement des problèmes scientifiques, en ingénierie, en physique, de devoir résoudre $f(x) = 0$, où $f$ est un vecteur de fonctions de dimension $n$ et $x$ est un vecteur de dimension $m$.

Dans ce TP on va implémenter la méthode de Newton pour chercher les zéros d'une fonction scalaire $f(x)$ dans un intervalle donné $x \in [a\ \ b]$. Cette méthode est très efficace et très robuste.

## 2 Cas considérés

Effectuer les tests sur les fonctions différentes :

1. $f(x) = x\ (x-1)\ (x-2)$ pour les deux cas $x \in [0.5\ \ 1.5]$ et pour $x \in [0\ \ 2]$.
2. $f(x) = \cos(x)$ pour $x \in [0\ \ \pi]$
3. $f(x) = y_0$ où $f$ est la fonction de Fanno (aérodynamique compressible), $y_0$ est une valeur supérieure à 0, $y_0$ est à définir par l'utilisateur avec :

$$f(x) = \frac{1-x^2}{\gamma x} + \frac{\gamma+1}{2\gamma}\ln\frac{(\gamma+1)x^2}{2+(\gamma-1)x)}, \qquad \gamma = 1.4, x > 0$$

On cherchera la solution de $f(x) = 0.5$.

## 3 Algorithme de Newton

### 3.1 Mathématiques

On sait d'après un développement de Taylor à l'ordre 1 que :

$$f(x+h) = f(x) + h\frac{df}{dx}(x)$$

En supposant qu'on cherche $h$ pour obtenir $f(x+h) = 0$ alors on vérifie :

$$h = -f(x)/\frac{df}{dx}(x)$$

### 3.2 Algorithme

L'algorithme peut ainsi être décrit :

*on chercher une solution $s_0$ en fonction d'une valeur cible $y_0$ et d'une fonction $y = f(s)$. Si on connaît la dérivée (gradient) $df/ds$ analytiquement on peut l'introduire dans la boucle itérative au lieu de le calculer avec un schéma d'Euler en différence finie.*

Il est facilement programmable dans n'importe quel langage à partir de la descrition de la table 1. Il peut être modifié pour être rendu encore plus efficace...

```
# définir au préalable la fonction f(s), s est la variable

FONCTION s = Newton(ValeurCible,s_init,...)
# on modifie l'algorithme ci-dessous en fonction des paramètres de la fonction
  DEBUT
    # paramètre
    Epsilon     = 0.0001     # petit paramètre pour le calcul du gradient
    ErreurMax   = 0.00001    # définition de l'erreur maximale
    IterMax     = 20         # nombre d'iteration avant non convergence
    #ăValeurCible =  ???     # valeur  à choisir (on résout f(s)=ValeurCible)
                             # paramètre d'entrée
    # s_init    = ???        # paramètre d'entrée
    # initialisation
    s           = s_init     # initialisation de l'algorithme
    variation   = 1          # doit être supérieure à ErreurMax
    iter        = 0          # compteur d'itérations
    y           = f(s)       # première valeur nécessaire - (f doit être déclarée)
    REPETER TANT QUE  (|variation| > ErreurMax) et  (iter < IterMax)
        iter      = iter+1
        gradient  = (f(s+Epsilon)-y ) / Epsilon      # ou gradient analytique
        variation = - (y - ValeurCible) / gradient
        s         = s + variation
        y         = f(s)
        affichage de iter,s,variation,y

    SI     iter = IterMax alors affichage 'non convergence'
    SINON  affichage de la valeur trouvée : s
  FIN
```

TABLE 1 – Algorithme général de Newton pour résoudre $f(s) = \mathrm{ValeurCible}$

# 4 Programmation

— Chaque étudiant programme suivant son style et ses choix. Il faudra être rapide et efficace.
— Pour les trois fonctions, le programme devra déterminer le ou les zéros dans les intervalles considérés.
— Si on a fini avant la fin, il faut modifier le programme pour calculer la position des maximuns ou minimuns dans l'intervalle donné.
— Le code écrit sera envoyé en fin de séance à christophe.airiau@imft.fr

Contenu de la taille du programme, on peut tout écrire dans un seul fichier fortran qui contiendra une section "program", une fonction "function", et un ou plusieurs "subroutine". Il est possible, sans que cela soit nécessaire d'avoir un "module".

Avec un seul fichier on peut compiler avec une ligne de commande (pas besoin de makefile).

Une structure du fichier est donnée ci-dessous :

```
MODULE Newton
...
! avec éventuellement des fonctions ou subroutines
END MODULE Newton

FUNCTION y=f(x,option)
....
END FUNCTION f

SUBROUTINE Recherche_zeros(...)

END SUBROUTINE Recherche_zeros

PROGRAM Zeros
USE Newton
....
END PROGRAM Zeros
```

FIGURE 1 – Structure possible du programme `zeros.f90`