RESOLUTION DES EQUATIONS NUMERIQUES PAR NEWTON

par Charles Hubert

Le 03/01/11

Pour résoudre un système d'équations numériques on peut souvent utiliser la méthode de Newton. Le système doit être sous la forme

$$Eqn(x) = 0$$

le second membre doit être nul ; "x" est le vecteur des inconnues et "Eqn(x)" décrit les équations ; Eqn(x) et x ont la même dimension car il y a autant d'équations que d'inconnues.

La méthode construit une suite de vecteurs "xn" qui converge rapidement vers la solution, quand le départ de la suite est bien choisi. Cette suite se calcule par un algorithme de récurrence. Celui-ci consiste à remplacer au point xn le système à résoudre par le système linéaire approché au premier ordre. La formule de récurrence est alors

$$x_{n+1} = x_n - Eqn'(x_n)^{-1} \, Eqn(x_n)$$

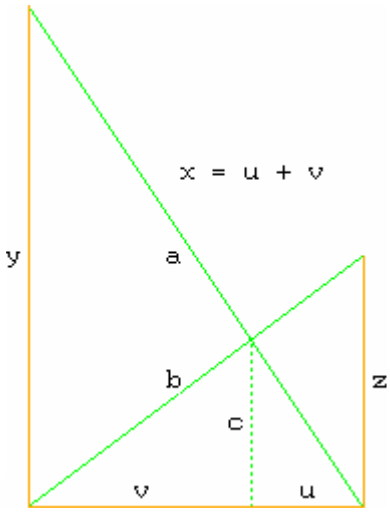
"n" est le numéro d'ordre dans la suite. "Eqn'(x)" est la matrice des dérivées partielles des composantes de "Eqn" par rapport aux composantes de "x" : le terme situé à la ligne "i" et à la colonne "j" de cette matrice est

$$Eqn'_{ij} = \frac{\partial Eqn_i}{\partial x_j}$$

On présente ici des fonctions qui mettent en oeuvre cette méthode.

L'algorithme de résolution

Considérons le problème suivant. Dans un couloir simple, sol horizontal et murs verticaux parallèles, sont disposées en croix deux échelles d'épaisseurs négligeables de a = 3m et b = 2m de long. Elles se coupent à c = 1m du sol. Il faut en déduire la largeur du couloir.



On peut écrire les équations de ce problème :

$$\frac{c}{y} = \frac{u}{u+v} \qquad \frac{c}{z} = \frac{v}{u+v} \qquad a^2 = x^2 + y^2 \qquad b^2 = x^2 + z^2$$

L'élimination de u et v conduit au système d'équations

```
      c(y + z) - yz = 0     x + y - a = 0     x + z - b = 0
                                  2   2   2         2   2   2
```

L'inconnue "x" n'apparaît qu'au carré ; on peut donc prendre x^2, y, z comme inconnues et calculer ensuite x par une racine carrée. Rassemblons y et z dans un seul tableau "yz". On peut décrire les équations par la fonction

```
    ∇  eq←EqaCouloirM incon
[1]    eq←incon
[2]    eq[; 0]←(c×+/yz)-×/yz
[3]    eq[; 1 2]←(x2+[0]yz*2)-[1](a,b)*2
    ∇
```

On prépare alors la résolution par la fonction :

```
    ∇  incon←IniCouloirM d
[1]    (a b c)←d
[2]    1 VarIncon 'x2 yz 2' ⍝ ou 1 VarIncon 'x2 yz' 2
[3]    Ayz a,b
    ∇
```

La ligne [1] affecte les valeurs de d à a, b, c. A la ligne [2] la fonction VarIncon déclare les inconnues x2 de dimension vide et yz de dimension 2. La ligne [3] initialise y et z.

La méthode de Newton a besoin que toutes les inconnues soient dans un seul tableau "incon", ce qu'on peut obtenir par la méthode décrite dans l'article

TABLEAUX DANS UN VECTEUR UNIQUE

Pour cela VarIncon crée la variable "incon" puis deux variables

```
      lx2  lyz
       0   1 2
```

permettant de sélectionner les colonnes de incon pour y retrouver x2 et yz. Elle crée aussi quatre fonctions :

```
    ∇  δ_x←x2                ∇  Ax2 δ_x
[1]    δ_x←incon[;lx2]   [1]    incon[;lx2]←δ_x
    ∇                        ∇
    ∇  δ_x←yz                ∇  Ayz δ_x
[1]    δ_x←incon[;lyz]   [1]    incon[;lyz]←δ_x
    ∇                        ∇
```

Ces fonctions permettent, par exemple, d'obtenir les valeurs des inconnues y et z en exécutant simplement "yz" et d'y affecter des valeurs par la fonction "Ayz". En fait les dimensions de x2 et yz sont

```
      ⍴'' x2   yz
          1    1 2
```

parce que "Newton" doit calculer les premiers membres des équations pour plusieurs combinaisons de valeurs des inconnues dans plusieurs lignes de "incon" au lieu d'une. La liste des inconnues est dans "varIncon". La liste des objets ainsi créés est dans "objIncon".

On résout alors les équations par "Newton" en exécutant

```
⍝                          ┌── inverse de la précision 1E¯9
⍝                          │          valeurs de a b c ──┬─┬─┐
      (incon iterRest)←'1E9×EqaCouloirM' Newton IniCouloirM 3 2 1
```

On trouve

```
      (x2*0.5),yz
 1.231185724 2.735723252 1.576128711
```

qui sont les valeurs des inconnues x, y, z ; puis

```
      iterRest
95
```

La fonction "Newton" admet par défaut jusqu'à 100 itérations ; 100-95=5 ont suffi.

Les inconnues et les équations sont disposées dans des matrices à 1 ligne ; c'est le choix indiqué par la valeur ''1'' de l'argument gauche de VarIncon. "Newton" calcule les dérivées partielles des équations en donnant des petites variations aux inconnues, ce qui se fait sur plusieurs lignes de la

matrice incon.

On peut résoudre à la fois pour plusieurs valeurs des données placées au niveau de profondeur 2 au moins :

```
      c←c 1 0.9 0.8 0.7
```

alors

```
      (incon iterRest)←'1E9×EqaCouloirM' Newton IniCouloirM 3 2 c
```

et on trouve

```
      iterRest ◇ ≡incon ◇ ⊃c,(x2*0.5),yz
95
2
 1           0.9         0.8         0.7
 1.231185724 1.456857099 1.62121259  1.742713714
 2.735723252 2.622511657 2.524216659 2.441915009
 1.576128711 1.370243552 1.171183051 0.9812996027
```

On aurait pu créer les fonctions

```
    ∇  eq←EqaCouloirV incon
[1]    eq←incon
[2]    eq[0]←(c×+/yz)-×/yz
[3]    eq[1 2]←(x2+yz*2)-(a,b)*2
    ∇
    ∇  incon←IniCouloirV d
[1]    (a b c)←d
[2]    0 VarIncon 'x2 yz 2'
[3]    Ayz a,b
    ∇
```

puis exécuter

```
      (incon iterRest)←'1E9×EqaCouloirV' Newton IniCouloirV 3 2 1
```

On aurait trouvé

```
      ⍴incon ◇ (x2*0.5),yz
3
1.231185724 2.735723252 1.576128711
```

Les inconnues et les équations sont alors disposées dans des vecteurs ; c'est le choix indiqué par la valeur ''0'' de l'argument gauche de VarIncon. "Newton" calcule les dérivées partielles des équations en donnant des petites variations aux inconnues, ce qui se fait dans chaque case du vecteur des inconnues. Le niveau 2 est ainsi réservé au calcul de ces dérivées partielles. A cause de cela, si on veut traiter plusieurs cas à la fois, il faut placer les valeurs d'un paramètre au niveau 3 au moins :

```
      c←cc 1 0.9 0.8 0.7
      (incon iterRest)←'1E9×EqaCouloirV' Newton IniCouloirV 3 2 c
```

On trouve

```
      iterRest ◇ ≡incon ◇ ⊃⊃c,(x2*0.5),yz
95
3
 1           0.9         0.8         0.7
 1.231185724 1.456857099 1.62121259  1.742713714
 2.735723252 2.622511657 2.524216659 2.441915009
 1.576128711 1.370243552 1.171183051 0.9812996027
```

la profondeur de "incon" a augmenté de un par rapport à l'exemple matriciel.

La fonction Newton s'utilise de la manière suivante

```
    (incon nbIterRest) ← {dx} {nbMaxIter} {'Eqn'} Newton inconInit
```

Les sous-arguments gauches sont facultatifs, mais il doit y avoir un argument, ne serait-ce qu'un vecteur vide. Chaque sous-argument vide prend la valeur par défaut. Si on l'appelle sans argument gauche, Newton indique sa syntaxe et ses valeurs par défaut.

"dx" est la variation que Newton applique aux inconnues pour calculer les dérivées partielles des équations par rapport aux inconnues.

"nbMaxIter" est le nombre maximum d'itérations.

"Eqn" est le nom de la fonction qui décrit les équations. Newton les considère satisfaites à 1 près ; c'est pourquoi on a multiplié les équations de base par `1E9` pour qu'elles soient vérifiées à `1E¯9` près.

"inconInit" est le point de départ de l'itération.

"incon" est la solution des équations à 1 près.

"nbIterRest" est le nombre d'itérations non utilisées ; si sa valeur est négative, Newton n'a pas trouvé de solution avec le nombre d'itérations autorisé.

Note

Si la dimension d'une inconnue "xyz" doit être (n,3), "n" étant une variable, on la déclare par
`1 VarIncon ... 'xyz' n 3 ...` ou `0 VarIncon ... 'xyz' n 3 ...`

La programmation

La fonction Newton crée une fonction locale `δ_N` qui exécute les itérations.

Pour cela Newton sélectionne parmi ses propres lignes celles qui commencent par `⍝n, ⍝0 et ⍝1` et y copie la syntaxe de la fonction "équations". Par exemple, si on appelle Newton par

```
⍝              équations
⍝         ┌──────────────┐
      ... '1E6× a b Equat c' Newton ...
```

la copie textuelle de la fonction "équations" lui fait solliciter les équations ainsi

```
⍝            équations
⍝        ┌──────────────┐
      ... 1E6× a b Equat c (argumentLocal)
```

Les fonctions

On peut trouver les fonctions

Newton, VarIncon, IndVec, ValVec, EqaLin

dans le fichier APL*PLUS

EQUAT.SF

La composante 1 est une table des matières, les autres composantes sont les `⎕vr` des fonctions de ce fichier.

Les codes de IndVec et ValVec, appelées par VarIncon, sont listés dans l'article

TABLEAUX DANS UN VECTEUR UNIQUE

Le code de EqaLin, appelée par Newton, est listé dans l'article

RESOLUTION DES EQUATIONS ALGEBRIQUES LINEAIRES

```
    ∇  δ_r VarIncon δ_x;δ_o
[1]    δ_o← 'incon' 'varIncon' 'objIncon'
[2]    δ_x←IndVec '1 ',,⍕δ_x
[3]    varIncon←(⎕io+1)⊃δ_x
[4]    δ_o←δ_o,⊃,''/2↑δ_x
[5]    incon←(∈(δ_r⍴1),¯1↑δ_x)⍴0
[6]    δ_o←δ_o,('. ',(δ_r⍴';'),(⎕io⊃δ_o)    ) ValVec 2↑δ_x
[7]    δ_o←δ_o,('A ',(δ_r⍴';'),(⎕io⊃δ_o),'←') ValVec 2↑δ_x
[8]    objIncon←δ_o
    ∇
```

```
    ∇  δ_z←δ_d Newton δ_x;δ_N;δ_c;δ_g;δ_i;δ_y
[1]    δ_g← 'EqaNum' '2*¯10' '100'
[2]   ⍝∇ (xSolu nbIterRest) ← {dx} {nbMaxIter} {'Eq'} Newton xInitial
[3]   ⍝∇ xSolu      ≡ solution des équations
[4]   ⍝∇ Eq         ≡ équations |Eq(x)| ≤ 1        ; % par défaut
[5]   ⍝∇ dx  pour calculer les dérivées partielles ; % par défaut
[6]   ⍝∇ nbMaxIter  ≡ nombre max d'itérations      ; % par défaut
[7]    δ_z←(δ_z[;⎕io]='⍝')≠δ_z←⎕cr 'Newton'
[8]    →(⎕nc 'δ_d')ρδ_A ⋄ δ_z← 0 2 ↓(δ_z[;⎕io+1]='∇')≠δ_z
[9]    (('%'=,δ_z)/,δ_z)←δ_g ⋄ δ_z←'⍝',⊃∈¨⊂[⎕io+1]δ_z ⋄ →0
[10]  δ_A:δ_y←' '=↑¨0ρ¨∈¨δ_d←,δ_d
[11]   δ_c←∈δ_y/δ_d ⋄ δ_c←∈δ_c,(0=ρδ_c)↑δ_g ⋄ δ_d←2ρ((~δ_y)/δ_d),0ρ¨
       2 2
[12]   (δ_d δ_i)←⍉(,¨δ_d),¨(ρ¨∈¨δ_d)↓¨⍋¨1↓δ_g
[13]   δ_i←⌊0.5+↑∈δ_i ⋄ δ_g←2>ρρδ_x ⋄ δ_d←((δ_g↓1),ρδ_d)ρδ_d
[14]   δ_z← 0 2 ↓(δ_z[;⎕io+1]∈'n',⍕δ_g)≠δ_z
[15]   (('⍵'=,δ_z)/,δ_z)←⊂δ_c ⋄ 0 0 ρ⎕fx⊃∈¨⊂[⎕io+1]δ_z
[16]   δ_c←(2ρ¯1↑ρδ_x)ρ0 ⋄ (⎕io ⎕io ⍵δ_c)←0.5×δ_d ⋄ δ_c←⊂[⍳δ_g]δ_c;0;
      -δ_c
[17]   δ_N
[18]   δ_z← δ_x δ_i
[19]  ⍝n δ_N
[20]  ⍝nδ_A:→(0>δ_i←δ_i-1)↑0
[21]  ⍝0 δ_y←⍵(δ_c+(ρδ_c)ρδ_x)
[22]  ⍝1 δ_y←⊃[⎕io]⍵(δ_c+δ_x)
[23]  ⍝n δ_y←⍵-≠(2, 1 0 +¯1↑ρδ_y)ρδ_y;0
[24]  ⍝0 δ_x←δ_x-δ_d×⍵EqaLin δ_y
[25]  ⍝1 δ_x←δ_x-δ_d×⊂",EqaLin δ_y
[26]  ⍝n →(∨/∈1<|δ_y[;⎕io+↑ρδ_y])↑δ_A
    ∇
```