Fiche n^0 3. Résolution numérique d'EDOs en langage C

Exercice 1 **– EDOs du second ordre**
Etant donnés $\omega \in \mathbb{R}^*$, $y_0 \in \mathbb{R}$ et y'_0, on souhaite résoudre de façon approchée l'équation différentielle ordinaire linéaire du second ordre à coefficients constants suivante :

$$y'' = -\omega^2 y \text{ avec } y(0) = y_0 \text{ et } y'(0) = y'_0.$$

a) Quelle est la solution exacte de cette équation ?

Nous allons utiliser les schémas étudiés pour l'équation différentielle du premier ordre pour construire des schémas pour cette équation du second ordre.

b) Montrer que l'équation du second ordre s'écrit de façon équivalente sous la forme du système du premier ordre suivant

$$\left\{ \begin{array}{rcl} y' & = & \omega z \\ z' & = & -\omega y \end{array} \right.$$

dont on donnera la condition initiale.

c) Ecrire les schémas d'Euler explicite, implicite et de Crank-Nicolson pour ce système.

d) Dans le schéma d'Euler explicite, en combinant l'expression de $y_{n+2} - y_{n+1}$ avec celle de $y_{n+1} - y_n$, montrer que l'on peut éliminer z pour obtenir le schéma à trois points suivants sur y :

$$\frac{y_{n+2} - 2y_{n+1} + y_n}{\Delta t^2} = -\omega^2 y_n.$$

Montrer en utilisant les résultats sur les récurrences à trois termes rappelés dans la fiche n^0 2 que ce schéma est inconditionnellement instable (i.e. instable pour toute valeur du pas de temps). Montrer toutefois que si on raisonne sur un temps de simulation fini (i.e. $n\Delta t = T$ avec T fixé), alors cette instabilité se réduit à mesure que Δt tend vers 0.

e) Quels schémas sur y obtient on en éliminant z dans les schémas d'Euler implicite et de Crank-Nicolson ? Etudier leur stabilité et montrer que pour Euler implicite, l'amplitude de la solution y_n tend vers 0 tandis qu'elle est préservée pour Crank-Nicolson. Dans ce dernier cas, trouver quelle est la pulsation numérique ω_{num} et la comparer à ω.

Réponses

a) L'équation caractéristique associée est $r^2 = -\omega^2$, la solution est donc de la forme $y(t) = A\cos(\omega t) + B\sin(\omega t)$. La condition $y(0) = y_0$ fournit $A = y_0$ et la condition $y'(0) = y'_0$ fournit $B = \frac{y'_0}{\omega}$.

b) Posons $z(t) = \frac{y'(t)}{\omega}$. Alors bien sûr $y' = \omega z$ et par ailleurs on a $z' = \frac{y''}{\omega}$. Mais lorsque y est solution de l'EDO de départ, on a $y'' = -\omega^2 y$ et donc $z' = -\frac{\omega^2 y}{\omega} = -\omega y$. Donc si y est solution de l'EDO de départ, alors (y, z) est solution du système du premier ordre.
Inversement, si (y, z) est solution du système du premier ordre, alors en dérivant la première équation on obtient $y'' = \omega z'$ puis en utilisant la seconde équation du système on obtient $y'' = -\omega^2 y$.
Pour la condition initiale, on a bien sûr $y(0) = y_0$ et $z(0) = \frac{y'(0)}{\omega} = \frac{y'_0}{\omega}$.

c) Dans les trois schémas, les dérivées premières sont approchées par la différence finie entre l'instant $(n+1)\Delta t$ et l'instant $n\Delta t$. Pour Euler explicite, le membre de droite est évalué à l'instant $n\Delta t$, pour Euler implicite à l'instant $(n+1)\Delta t$. Pour Crank-Nicolson, il est évalué comme la demi-somme des valeurs aux instants $(n+1)\Delta t$ et $n\Delta t$. Cela fournit donc :
Euler explicite :

$$\left\{ \begin{array}{rcl} y_{n+1} - y_n & = & \Delta t \omega z_n \\ z_{n+1} - z_n & = & -\Delta t \omega y_n \end{array} \right.$$

Euler implicite :

$$\left\{ \begin{array}{rcl} y_{n+1} - y_n & = & \Delta t \omega z_{n+1} \\ z_{n+1} - z_n & = & -\Delta t \omega y_{n+1} \end{array} \right.$$

Crank-Nicolson :

$$\begin{cases} y_{n+1} - y_n &= \frac{1}{2}\Delta t\omega(z_{n+1} + z_n) \\ z_{n+1} - z_n &= -\frac{1}{2}\Delta t\omega(y_{n+1} + y_n) \end{cases}$$

d) Pour Euler explicite, en appliquant les formules avec $(n+1)$ à la place de n, on a : $y_{n+2} - y_{n+1} = \Delta t\omega z_{n+1}$. En retranchant l'expression $y_{n+1} - y_n = \Delta t\omega z_n$, on obtient

$$y_{n+2} - 2y_{n+1} + y_n = \Delta t\omega(z_{n+1} - z_n).$$

on remplace alors $(z_{n+1} - z_n)$ par sa valeur, à savoir $-\Delta t\omega y_n$ et on obtient l'expression demandée.
La récurrence à trois termes s'écrit donc $y_{n+2} - 2y_{n+1} + [1 + (\omega\Delta t)^2]y_n = 0$. L'équation caractéristique associée est alors $r^2 - 2r + [1 + (\omega\Delta t)^2] = 0$, dont le discriminant est strictement négatif. Les solutions sont donc complexes conjuguées $r_\pm = \rho\exp(\pm i\theta)$. Et on a $\rho^2 = r_+ \times r_-$. Mais le produit des racines vaut $[1 + (\omega\Delta t)^2]$ et donc $\rho^2 = [1 + (\omega\Delta t)^2] > 1$. La solution pour y_n est donc sous la forme

$$y_n = \rho^n(A\cos(n\theta) + B\sin(n\theta))$$

avec $\rho > 1$: la solution est en effet instable inconditionnellement (i.e. pour toute valeur de Δt), son module n'étant pas bornée car ρ^n tend vers $+\infty$ avec n. Toutefois, pour un temps de simulation fini T, on a $n_{\text{fin}}\Delta t = T$ et à l'instant final on aura $\rho^{n_{\text{fin}}} = [1 + (\omega\Delta t)^2]^{\frac{T}{2\Delta t}}$. Pour Δt tendant vers 0, le développement limité de $\ln(1+x)$ en 0 nous donne que $\ln(\rho^{n_{\text{fin}}}) \sim_0 \frac{1}{2}T\omega^2\Delta t$, ce qui montre que $\rho^{n_{\text{fin}}}$ va décroître vers 1 lorsque Δt va tendre vers 0. C'est ce qui fait dire que le schéma d'Euler explicite est 'de moins en instable' lorsque Δt tend vers 0.

e) Pour Euler implicite, en appliquant les formules avec $(n+1)$ à la place de n, on a : $y_{n+2} - y_{n+1} = \Delta t\omega z_{n+2}$. En retranchant l'expression $y_{n+1} - y_n = \Delta t\omega z_{n+1}$, on obtient

$$y_{n+2} - 2y_{n+1} + y_n = \Delta t\omega(z_{n+2} - z_{n+1}).$$

on remplace alors $(z_{n+2} - z_{n+1})$ par sa valeur, à savoir $-\Delta t\omega y_{n+2}$ et on obtient

$$y_{n+2} - 2y_{n+1} + y_n = -(\omega\Delta t)^2 y_{n+2}.$$

La récurrence à trois termes s'écrit donc $[1 + (\omega\Delta t)^2]y_{n+2} - 2y_{n+1} + y_n = 0$. L'équation caractéristique associée est alors $[1 + (\omega\Delta t)^2]r^2 - 2r + 1 = 0$, dont le discriminant est strictement négatif pour toute valeur de Δt. Les solutions sont donc complexes conjuguées $r_\pm = \rho\exp(\pm i\theta)$. Et on a $\rho^2 = r_+ \times r_-$. Mais le produit des racines vaut $[1 + (\omega\Delta t)^2]^{-1}$ et donc $\rho^2 = [1 + (\omega\Delta t)^2]^{-1} < 1$. La solution pour y_n est donc sous la forme

$$y_n = \rho^n(A\cos(n\theta) + B\sin(n\theta))$$

avec $\rho < 1$: la solution tend donc vers 0 pour toute valeur de Δt lorsque n tend vers $+\infty$.

Pour Crank-Nicolson, en appliquant les formules avec $(n+1)$ à la place de n, on a : $y_{n+2} - y_{n+1} = \frac{1}{2}\Delta t\omega(z_{n+2} + z_{n+1})$. En retranchant l'expression $y_{n+1} - y_n = \frac{1}{2}\Delta t\omega(z_{n+1} + z_n)$, on obtient

$$y_{n+2} - 2y_{n+1} + y_n = \frac{1}{2}\Delta t\omega(z_{n+2} - z_n) = .\frac{1}{2}\Delta t\omega[(z_{n+2} - z_{n+1}) + (z_{n+1} - z_n)];$$

on remplace alors $(z_{n+2} - z_{n+1})$ et $(z_{n+1} - z_n)$ par leurs valeurs respectives et on obtient

$$y_{n+2} - 2y_{n+1} + y_n = -\frac{1}{4}(\omega\Delta t)^2(y_{n+2} + 2y_{n+1} + y_n).$$

En posant $\alpha = \frac{1}{4}(\omega\Delta t)^2$, la récurrence à trois termes s'écrit donc $(1+\alpha)y_{n+2} - 2(1-\alpha)y_{n+1} + (1+\alpha)y_n = 0$. L'équation caractéristique associée est alors $(1+\alpha)r^2 - 2(1-\alpha)r + (1+\alpha) = 0$, dont le discriminant vaut -16α et est strictement négatif pour toute valeur de Δt. Les solutions sont donc complexes conjuguées $r_\pm = \rho\exp(\pm i\theta)$. Et on a $\rho^2 = r_+ \times r_-$. Mais le produit des racines vaut 1 et donc $\rho = 1$. La solution pour y_n est donc sous la forme

$$y_n = (A\cos(n\theta) + B\sin(n\theta));$$

son amplitude est donc conservée au cours du temps, et ce pour toute valeur de Δt.
Pour comparer avec la solution exacte, on définit une pulsation numérique de la façon suivante : $\omega_{\text{num}} = \frac{\theta}{\Delta t}$; en effet on a alors, en posant $t_n = n\Delta t$

$$y_n = (A\cos(\omega_{\text{num}}t_n) + B\sin(\omega_{\text{num}}t_n)).$$

La valeur y_n étant une approximation de la solution exacte au temps t_n, comparer ω et ω_{num} permet d'avoir une idée de l'erreur commise par l'approximation numérique.

Les solutions de l'équation caractéristique étant $\frac{(1-\alpha)\pm 2i\sqrt{\alpha}}{(1+\alpha)}$, on a $\tan(\theta) = \frac{2\sqrt{\alpha}}{(1-\alpha)} = \frac{\omega\Delta t}{(1-\alpha)}$. Pour Δt suffisamment petit et à l'aide du développement limité de la fonction arctan en 0 ($\arctan(x) = x - \frac{x^3}{3} + \mathcal{O}(x^5)$), on obtient :

$$\omega_{\text{num}} = \frac{1}{\Delta t}\left[\frac{\omega\Delta t}{(1-\alpha)} - \frac{(\omega\Delta t)^3}{3(1-\alpha)^3}\right] + \mathcal{O}(\Delta t^4)$$

En utilisant le fait que $\frac{1}{(1-\alpha)} = (1+\alpha) + \mathcal{O}(\alpha^2) = (1+\alpha) + \mathcal{O}(\Delta t^4)$, on obtient finalement, en ne gardant que les termes d'ordre le plus élevé :

$$\omega_{\text{num}} = \omega\left[1 + \alpha - \frac{(\omega\Delta t)^2}{3}\right] + \mathcal{O}(\Delta t^4) = \omega\left[1 - \frac{(\omega\Delta t)^2}{12}\right] + \mathcal{O}(\Delta t^4).$$

On remarque que pour $\omega\Delta t = \frac{1}{2}$, c'est-à-dire pour environ 12 pas de temps par période, l'erreur relative sur la pulsation est d'environ 2%.

Exercice 2 – **Implémentation en Langage C - Allocation dynamique**

À des fins de post-traitement et de visualisation, on souhaite maintenant conserver toutes les valeurs y_n des schémas ci-dessus.

On a donc besoin de tableaux de `double` *de taille le nombre de pas de temps +1 (puisqu'on va stocker aussi y_0), mais cette taille n'est pas connue au moment de la compilation puisqu'elle est lue dans un fichier de données.*

Un moyen de s'en sortir est l'allocation dynamique : on va demander de réserver une zone mémoire de taille $(N+1)$ fois la taille d'un double (obtenue avec `sizeof(double)`*). Ceci se fait de la façon suivante*

```
double* y_imp = NULL; // y_imp est un pointeur vers un double
y_imp = malloc ((N+1) * sizeof(double)); // reservation de l'espace memoire
if (y_imp == NULL)
{
        printf("Impossible d'allouer le tableau y_imp \n");
         return(EXIT_FAILURE);
}
```

`y_imp` *est une adresse memoire d'une zone contenant un double (ce double étant la première valeur stockée dans le tableau). On peut accéder aux valeurs dans le tableau par* `y_imp[i]`*, ou par* `*(y_imp+i)`*, la valeur* `i` *étant comprise entre* 0 *et (taille du tableau - 1).*

Lorsqu'on n'a plus besoin du tableau, on peut libérer l'espace mémoire correspondant en utilisant

```
free (y_imp);
```

Maintenant que nous avons rappelé comment allouer de façon dynamique un tableau de taille donnée, il y a deux façons de procéder pour que `y_imp` *contienne le résultat issu d'un calcul effectué dans une fonction :*

- *Soit on écrit dans le programme principal* `y_imp = euler_imp(omega,dt,y0,yprime0,N);` *mais, puisque* `y_imp` *est un* `double*`*, il faut que la fonction* `euler_imp` *retourne un* `double*`
- *Soit on passe simplement un argument de plus à* `euler_imp` *et on la déclare* `void`*; on écrit alors dans le programme principal* `euler_imp_bis(omega,dt,y0,yprime0,N,y_imp);`

Examinons à présent ces deux cas :

Première possibilité

Programme principal

```
int main(int argc, char *argv[])
{
int N;
[on récupère la valeur de N]
```

```
double* y_imp = NULL; // y_imp est un pointeur vers un double pour l'instant ini-
tialisé à NULL (y_imp ne pointe vers aucune case mémoire)

y_imp = euler_imp(omega,dt,y0,yprime0,N); // on affecte à y_imp le pointeur de
double retourné par la fonction euler_imp

[on utilise y_imp en accédant à ses valeurs par y_imp[i]]
free(y_imp); // on libère l'espace mémoire occupé par les valeurs du tableau
return 0;
}
```

La fonction `euler_imp` *doit retourner un* `double*` *:*

```
double* euler_imp(double omega, double pas_de_temps, double cond_init,
double deriv_init, int nombre_de_pas)
{
int i;
double* valeur = NULL; // valeur est un pointeur vers un double

valeur = malloc ((nombre_de_pas + 1) * sizeof(double)); // on a maintenant réservé
un espace mémoire de taille (nombre_de_pas + 1) * sizeof(double) et valeur pointe
vers la première case de cet espace mémoire

if (valeur == NULL)
{
        printf("Impossible d'allouer le tableau valeur dans euler imp \n");
         return(NULL);
}
[On remplit le tableau valeur en écrivant valeur[i] = ...;]

return(valeur); // on retourne l'adresse du premier élément du tableau valeur
}
```

Deuxième possibilité

Programme principal

```
int main(int argc, char *argv[])
{
int N;
[on récupère la valeur de N]

double* y_imp = NULL; // y_imp est un pointeur vers un double
y_imp = malloc ((N+1) * sizeof(double)); // reservation de l'espace memoire
if (y_imp == NULL)
{
        printf("Impossible d'allouer le tableau y_imp \n");
         return(EXIT_FAILURE);
}

euler_imp_bis(omega,dt,y0,yprime0,N,y_imp); // on passe en argument de la fonction
l'adresse du premier élément du tableau y_imp

[on utilise y_imp en accédant à ses valeurs par y_imp[i]]
free(y_imp); // on libère l'espace mémoire occupé par les valeurs du tableau
```

```
return 0;
}
```

La fonction `euler_imp_bis` *doit prendre un argument de plus, de type* `double*`

```
void euler_imp_bis(double omega, double pas_de_temps, double cond_init,
double deriv_init, int nombre_de_pas, double* valeur)
{
[On remplit le tableau valeur en écrivant valeur[i] = ...;]

// on ne retourne rien puisque euler_imp_bis est de type void
}
```

Quelque soit l'option retenue, à la fin du main on écrira dans un fichier les valeurs suivantes (une ligne par pas de temps) :

```
temps écoulé, valeur exacte de la solution, valeur estimée par Euler explicite,
valeur estimée par Euler implicite, valeur estimée par Crank-Nicolson
```

et on visualisera les courbes grâce à `gnuplot` *(voir la fiche précédente).*
