Fiche n^0 4. Résolution numérique de problèmes aux limites en langage C

Exercice 1 – **EDOs du second ordre avec conditions aux limites**
Etant donnés $\omega \in \mathbb{R}$, $y_0 \in \mathbb{R}$, $T > 0$ un temps final et $y_T \in \mathbb{R}$, on souhaite résoudre de façon approchée l'équation aux dérivées ordinaires linéaire du second ordre à coefficients constants suivante :

$$y'' = -\omega^2 y \text{ avec } y(0) = y_0 \text{ et } y(T) = y_T.$$

a) Dans quel cas existe-t-il une unique solution à cette équation ? Quelle est son expression ?

Nous allons utiliser les schémas étudiés dans la fiche précédente pour approcher, lorsqu'elle existe, la solution de cette équation.

b) Quelle était la forme générale des solutions numériques de cette équation à l'aide des schémas vus dans la fiche précédente ? Sous quelle condition liée à la pulsation numérique ω_{num} existe-t-il une solution unique au schéma numérique ? Quel problème entrevoyez vous, lié à la différence entre les pulsations exacte et numérique ?

Lors de la résolution du problème aux conditions initiales correspondant ($y(0)$ et $y'(0)$ donnés), il était possible de calculer y_2 à partir de y_0 et y_1, et ensuite y_{n+2} à partir de y_{n+1} et y_n, jusqu'au temps final.

Ici ce n'est plus le cas, car on ne donne pas y_1 et on souhaite que $y_N = y_T$, où $N = T/\Delta t$.

L'idée est alors d'utiliser une "méthode de tir" selon le principe suivant : connaissant y_0, on va chercher y_1 de telle sorte qu'en utilisant la récurrence à trois terme donnant y_{n+2} à partir de y_{n+1} et y_n, on aboutisse à $y_N = y_T$.

c) Montrer que, si y_0 est donné et que le schéma numérique est linéaire, alors y_2 est une fonction affine de y_1. En déduire par récurrence que y_N est une fonction affine de y_1 si le schéma est linéaire.

d) En déduire une façon de calculer le y_1 qui est tel qu'en appliquant la récurrence, on obtienne $y_N = y_T$.

---

a) La solution générale de cette équation est, comme dans la Fiche n^{0}3, $y(t) = A\cos(\omega t) + B\sin(\omega t)$. La condition $y(0) = y_0$ fournit $A = y_0$. La condition $y(T) = y_T$ s'écrit :

$$y_0 \cos(\omega T) + B \sin(\omega T) = y_T.$$

Pour avoir une solution unique pour B, il faut et il suffit que $\sin(\omega T) \neq 0$, et donc $\omega T \neq k\pi$, avec $k \in \mathbb{Z}$.

b) La solution générale était :

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

puisque les solutions de l'équation caractéristique associée étaient complexes conjuguées et s'écrivaient $\rho\exp(\pm i\theta)$.

Comme pour la solution continue, on obtient

$$y_0 = A,$$

et ensuite, en notant $N = \frac{T}{\Delta t}$

$$y_N = y_T = \rho^N (y_0 \cos(N\theta) + B\sin(N\theta))$$

En définissant la pulsation numérique $\omega_{\text{num}} = \frac{\theta}{\Delta t}$, on obtient la condition suivante pour que cette équation fournisse une unique solution pour B :

$$N\theta = N\Delta t \omega_{\text{num}} = \omega_{\text{num}} T \neq k\pi, \text{ avec } k \in \mathbb{Z}.$$

Le problème que l'on entrevoit ici, c'est que, même si on s'est placé dans le cas où la solution exacte existe et est unique, à savoir $\omega T \neq k\pi$, la solution numérique peut poser problème, car on a besoin de $\omega_{\text{num}} T \neq k\pi$, mais comme $\omega_{\text{num}} \neq \omega$, cela n'est pas assuré ! Une façon de s'en sortir est de prendre Δt "suffisamment petit" (à déterminer selon les schémas) pour que les deux propriétés "$\omega T \neq k\pi$ et "ω_{num} tend vers ω lorsque Δt tend vers 0" assurent que $\omega_{\text{num}} T \neq k\pi$.

c) On peut montrer cette propriété par récurrence : Le schéma à trois points est supposé linéaire, c'est-à-dire qu'il existe a et b tels que

$$y_{n+2} = a y_{n+1} + b y_n.$$

(On le suppose pour simplifier à coefficients constants, i.e. les coefficients a et b ci-dessus ne dépendent pas de n, mais l'analyse se généralise facilement au cas coefficients non-constants, l'important étant que les coefficients ne dépendent pas des valeurs de la suite (y_n).)

On a donc :

$$y_2 = a y_1 + y_0.$$

Pour y_0 fixé, ce qui est bien le cas dans la pratique, y_2 est bien une fonction affine de y_1. Supposons maintenant la propriété vraie jusqu'au rang $n \geq 2$, et vérifions que c'est encore le cas pour le rang $n+1$. On a

$$y_{n+1} = a y_n + y_{n-1}.$$

Mais comme on suppose que y_n et y_{n-1} sont des fonctions affines de y_1, alors une combinaison linéaire de deux fonctions affines est elle-même affine, ce qui prouve que y_{n+1} est elle-même une fonction affine de y_1, ce qui prouve le résultat.

d) Puisque y_N est une fonction affine de y_1, elle s'écrit

$$y_N = \alpha y_1 + \beta,$$

mais on ne connaît pas α et β ; on pourrait calculer leurs valeurs à l'aide de la récurrence, mais on peut aussi (c'est ce qu'on fait en pratique car c'est généralisable à d'autres cas) utiliser l'appel aux fonctions écrites dans la fiche précédente pour trouver leurs valeurs. On procède de la sorte : On a toujours y_0 donné. Dans un premier temps, on va choisir une première valeur pour y_1, nommons la $y_1^{(1)}$ (par exemple y_0), et on applique ensuite la récurrence à trois termes ; cela va permettre de calculer $y_2^{(1)}$, puis $y_3^{(1)}$ et ainsi de suite jusqu'à $y_N^{(1)}$ qui vaut $\alpha y_1^{(1)} + \beta$. Dans un deuxième temps, on choisit une autre valeur pour y_1, (par exemple $y_0 + \Delta t$) que l'on nomme $y_1^{(2)}$ et on applique ensuite la récurrence à trois termes ; cela va permettre de calculer $y_2^{(2)}$, puis $y_3^{(2)}$ et ainsi de suite jusqu'à $y_N^{(2)}$ qui vaut $\alpha y_1^{(2)} + \beta$. On en déduit alors aisément les valeurs de α et β :

$$\alpha = \frac{(y_N^{(2)} - y_N^{(1)})}{(y_1^{(2)} - y_1^{(1)})}$$

et

$$\beta = y_N^{(1)} - \alpha y_1^{(1)}.$$

Connaissant α et β, il ne reste plus qu'à calculer le y_1 qui va résoudre $\alpha y_1 + \beta = y_T$, puisque nous voulons atteindre y_T et y_N. Cela donne :

$$y_1 = \frac{(y_T - \beta)}{\alpha}.$$

Il n'y a plus qu'à faire un troisième appel aux procédures de récurrence à trois termes en choisissant ce y_1 pour que cette fois-ci le résultat final donne bien $y_N = y_T$.

Commentaire : dans le cas où le schéma n'est pas linéaire (EDO non linéaire par exemple), alors ce n'est plus aussi simple : y_N devient une fonction non-linéaire de y_1, et on peut procéder par dichotomie, ou par toute autre façon d'approcher la solution d'une équation non-linéaire dont on ne connaît pas la forme explicite, mais pour laquelle on sait calculer la valeur de la fonction en un point (ici par la procédure de récurrence à trois termes).

Exercice 2 – Implémentation en Langage C de la méthode de tir

Ecrivez un programme principal dont l'exécutable prendra comme argument un fichier dans lequel se trouveront :

- *le nombre de pas de temps de la simulation,*
- *le temps final de la simulation,*
- *la pulsation ω,*

— *la condition initiale y_0 et*
— *la conditin finale y_T.*

Ce programme appellera une fonction qui implémentera la méthode de tir. Celle-ci devra appeler les schémas programmés lors de la dernière séance (voir fiche $n^0 3$).

On propose ce type d'implémentation, avec le schéma d'Euler implicite vu dans la fiche $n^0 3$.

```
--------------------------- fichier main.c

#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#include "tir_euler_imp.h"



int main(int argc, char *argv[])

{
  if (argc != 2)
  {
                  printf("\nLe programme doit recevoir un et un seul argument\n");
                  return(EXIT_FAILURE);
  }

int N ;
double temps_final;
double omega ;
double y0 ;
double yT;

FILE* fichier = NULL;

fichier = fopen(argv[1],"r");

    if (fichier == NULL)
    {
         printf("Impossible d'ouvrir le fichier %s \n",argv[1]);
         return(EXIT_FAILURE);
    }

fscanf(fichier, "%d %lf %lf %lf %lf", &N , &temps_final, &omega , &y0 , &yT);

printf("Nombre de pas de temps: %d \n",N);
printf("Temps final: %f \n",temps_final);
printf("Periode de la sinusoide: %f \n",omega);
printf("Condition initiale: %f \n",y0);
printf("Condition finale: %f \n",yT);


double* ynimp = NULL;

double dt;
```

```
dt = temps_final/N;

printf("Pas de temps %f \n",dt);

ynimp = tir_euler_imp(omega,dt,y0,yT,N);

free(ynimp);
fclose(fichier);

  return 0;
}
```

Le fichier `tir_euler_imp` *va implémenter la méthode de tir :*

```
------------------------ fichier tir_euler_imp.c

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "euler_imp.h"
#include "tir_euler_imp.h"

 double* tir_euler_imp(double omega, double pas_de_temps, double cond_init,
                       double cond_finale, int nombre_de_pas)
{
double val1;
double epsilon = 0.01;
double aa;
double bb;
double eul1, eul2;
double* valeur = NULL;

/** premier essai **/
val1 = cond_init;

valeur = euler_imp(omega, pas_de_temps, cond_init, val1, nombre_de_pas);
eul1 = valeur[nombre_de_pas];
free(valeur); // on libère la mémoire allouée dans euler_imp
valeur=NULL;

/** deuxième essai **/
val1 = cond_init + epsilon;

valeur = euler_imp(omega, pas_de_temps, cond_init, val1, nombre_de_pas);
eul2 = valeur[nombre_de_pas];
free(valeur); // on libère la mémoire allouée dans euler_imp
valeur=NULL;

aa = (eul2 - eul1)/epsilon;
bb = eul1 - aa*cond_init;

/** valeur de y1 et verif **/
val1 = (cond_finale - bb)/aa;
```

```
valeur = euler_imp(omega, pas_de_temps, cond_init, val1, nombre_de_pas);
printf("verification %lf %lf \n",valeur[nombre_de_pas],cond_finale);

return(valeur); // le programme principal pourra utiliser tout le tableau

}
```

C'est la fonction `euler_imp.c` *qui réservera l'espace mémoire nécessaire et qui construira le tableau de valeurs calculées par le schéma d'Euler implicite ayant comme premiers itérés* `cond_init` *et* `val1`.
