

## TE 2 : Résolution numérique d'équations différentielles ordinaires (EDO)

**Objectif** On se propose de résoudre numériquement des équations différentielles en commençant par des équations scalaires du premier ordre, via des méthodes à un seul pas. L'objectif est d'analyser l'influence de l'ordre de la méthode et de la valeur du pas sur l'erreur commise. En simple précision, on mettra en évidence les erreurs d'arrondi d'aspect aléatoire. Dans un second temps, on convertira le code en vectoriel pour traiter des systèmes d'équations différentielles couplées et, par les mêmes programmes, des EDO d'ordre supérieur.

### 1 EDO scalaire du premier ordre

#### 1.1 EDO scalaire du premier ordre : solution gaussienne

On recherche la solution de l'équation différentielle (1) avec la condition initiale  $y(t_0) = y_0$ . On suppose que  $y_0 > 0$  et  $a > t_0$ .

$$\frac{dy}{dt} = -\frac{(t-a)y}{\tau^2} \quad (1)$$

Montrer que la solution analytique se met sous la forme (2) d'une gaussienne centrée en  $a$  et de largeur  $\tau$ . Ne pas programmer directement l'expression (2), mais la reformuler pour limiter les erreurs d'arrondi et les dépassements de capacité.

$$y(t) = y_0 \exp\left[\frac{1}{2}\left(\frac{t_0-a}{\tau}\right)^2\right] \exp\left[-\frac{1}{2}\left(\frac{t-a}{\tau}\right)^2\right] \quad (2)$$

#### 1.2 Organisation des codes

Le programme comporte quatre fichiers source principaux (voir schéma 1.2.1, p. 2) :

1. Un programme principal (fichier `ordre1_scal.{c,f90}`) assurant le dialogue avec l'utilisateur pour choisir l'abscisse de début, le pas et l'abscisse de fin, la valeur initiale  $y_0$ <sup>1</sup> et l'ordre  $m$  de la méthode de résolution ainsi que la fonction à étudier. Il **définit** aussi les valeurs des paramètres (**a** et **tau** par exemple) des fonctions à tester. Ce programme crée alors le tableau des abscisses et calcule les valeurs de la solution analytique en ces points. Il lance ensuite la boucle qui appelle le solveur choisi pour calculer de proche en proche la solution numérique. Enfin, il transmet à la procédure `ecrit` toutes les informations utiles pour écrire le fichier de résultats.
2. Un fichier `fcts.{c,f90}` définissant les différentes fonctions second membre `dydt(t,y)` à intégrer, ainsi que les solutions analytiques `g(t,t0,y_0)` éventuellement associées. C'est dans ce fichier que sont **déclarés** (mais non définis) les paramètres communs des fonctions (**a** et **tau** notamment) comme variables globales.
3. Un fichier `methodes.{c,f90}` hébergeant les fonctions d'intégration des différents ordres qui permettent de progresser d'un pas, de  $t_i$  à  $t_{i+1}$ . Chaque méthode d'intégration respecte la syntaxe :

```
u_plus1(u_i, t_i, h, dydt)
```

où `dydt` est la fonction second membre à intégrer, `u_i` la valeur initiale, `t_i` l'instant initial ( $t_i$ ) et `h` le pas.

Le module des méthodes sera conçu de façon à pouvoir traiter une fonction second membre quelconque sans être recompilé : à cette fin, il comportera la déclaration de l'interface d'une fonction formelle `dydt(t,y)`.

4. Un module utilitaire (fichier `util.{c,f90}`) contenant la procédure `ecrit` chargée de créer le fichier formaté des résultats. Le nom du fichier sera choisi en fonction de la méthode (`euler.dat` par exemple signifiant méthode d'Euler). Le corps du fichier des résultats comportera une ligne par point de grille sous la forme :

abscisse	ordonnée estimée	ordonnée analytique	écart
$t_i$	$u_i$	$y(t_i)$	$u_i - y(t_i)$

On spécifiera un format compatible avec la dynamique des valeurs et assurant une précision de l'ordre de  $10^{-7}$ . Une fois la méthode mise au point, on ajoutera dans l'entête du fichier de résultats le pas et la méthode utilisés ainsi que le maximum de la valeur absolue de l'écart et l'instant où cet écart est maximal<sup>2</sup>.

1. Attention : en C, `y0`, `y1` et `yn` sont des fonctions de Bessel, ainsi que `j0`, `j1` et `jn`. On évitera donc de nommer ainsi des variables.  
2. En langage C, sur le modèle de la fonction `maxfloat1d` développée pour les tableaux automatiques 1D au TE 1, on écrira aussi une fonction `maxloefloat1d` qui rend un entier localisant dans les tableaux l'instant et la valeur du maximum de la valeur absolue.

### 1.2.1 Les quatre fichiers sources principaux

```
ordre1_scal.f90 ou ordre1_scal.c
Choix des paramètres
(début t0, fin tfin, pas h, valeur initiale y_0)
...
Choix de la méthode m.
...
Création et remplissage du tableau t des abscisses
et de celui exact de la fonction analytique.
Boucle de calcul pas à pas
    Appel de la méthode d'intégration
    d'ordre m appliquée à un second membre
    particulier pour calculer u_{i+1}
    Remplissage du tableau estime avec
    l'ordonnée estimée u_{i+1}.
...
Appel de ecrit(t, estime, exact, m)
```

```
fcts.f90 ou fcts.c
Déclare (sans les définir) les paramètres communs des
fonctions comme variables globales
Définit les fonctions second membre
dydt(t, y)
...
et les solutions analytiques associées
g(t, t0, y_0)
...
```

```
methodes.f90 ou methodes.c
Contient les 4 fonctions d'intégration de type
u_iplus1 (u_i, t_i, h, dydt)
1. Euler progressive
2. Point milieu
3. Runge Kutta d'ordre 3 (facultatif)
4. Runge Kutta d'ordre 4
La fonction formelle dydt (de deux variables réelles
et à valeur réelle) passée en argument des méthodes
d'intégration sera déclarée :
— en fortran, de type PROCEDURE(fty)::
  où fty désigne son interface abstraite,
  (elle-même définie une fois pour toutes dans un
  module nommé abstrait),
— et en C, via un pointeur de fonction de deux
variables adéquat.
```

```
util.f90 ou util.c
Contient notamment la procédure d'écriture du fichier
ecrit(t, estime, exact, m) ...
1. détermination du nom du fichier en fonction de
l'ordre m de la méthode ;
2. calcul de l'écart entre estimation et solution ana-
lytique, recherche de son maximum en valeur ab-
solue ainsi que de l'abscisse associée.
3. écriture du fichier des résultats avec entête
```

À ces fichiers principaux, viennent s'ajouter les fichiers d'entête pour le langage C et les interfaces abstraites pour le fortran.

### 1.3 Précision modifiable

Comme dans le TE 1B, écrire **dès le début** tout le code en définissant un type réel de précision facilement modifiable. La dépendance en fonction du fichier `mncs_precision.{h,f90}` qui choisit la précision (32 ou 64 bits) devra figurer explicitement dans le fichier `makefile` pour tous les autres fichiers objet sous peine d'incohérences. Elle induira alors une recompilation de tous les codes en cas de changement de précision.

Concernant les entrées-sorties, on pourra procéder de la manière suivante suivant les langages :

**En fortran :** Ne pas se soucier de la variante de type réel utilisée pour les entrées effectuées en format libre.

Lors des écritures en format libre, le nombre de chiffres dépend de la précision, mais aussi du compilateur. On peut aussi préférer convertir en `REAL` par défaut avant d'écrire, ce qui permet de choisir un format `E` ou `G` en imposant le nombre de chiffres.

**En C :** Pour conserver les formats de lecture, on peut saisir au format `%g` dans les deux cas avec des variables supplémentaires de type `float`, puis recopier les paramètres lus dans des variables de précision paramétrable. Mais on préférera une autre solution (nécessaire pour utiliser `lapack`) qui consiste à tester avec `sizeof` la taille du type réel choisi pour sélectionner le format de lecture adéquat.

En écriture, la fonction `fprintf`, grâce au passage d'argument par copie, fait les conversions nécessaires des `float` en `double` et il faut garder le format `%g3` pour les deux précisions.

---

			float	double	long double
3. Rappel des formats utilisables en C :	entrée	<code>scanf</code>	<code>%g</code>	<code>%lg</code>	<code>%Lg</code>
	sortie	<code>printf</code>	<code>%g</code>	<code>%g</code>	<code>%Lg</code>

## 1.4 Graphiques sous python

Tracer les graphiques sous `python` : relire le corps du fichier de résultats (en ignorant l'entête) pour en faire un tableau à deux dimensions de type `np.array` de `numpy`. Exploiter ensuite l'entête lu comme un tableau de chaînes de caractères pour construire un titre identifiant la méthode et le pas.

### Impression des graphiques

On sauvegardera les figures au format `pdf` directement sous `python` avec `plt.savefig("fig.pdf")`. Mais, comme leur format n'est pas forcément A4, pour les imprimer, on utilisera impérativement la commande `Lpr-A4pdf fig.pdf`

## 1.5 Plan de travail (suivre les instructions pour le compte-rendu p. 6)

Chaque module sera compilé séparément à l'aide de l'outil `make`. On spécifiera dans le fichier `makefile` les options de compilation présentes dans les `alias` de `gcc-mni-c99` et `gfortran-mni`.

On cherche à retrouver la solution gaussienne dans l'intervalle  $[0, 20]$  avec  $a = 4$ ,  $\tau = \sqrt{2}$ ,  $t_0 = 0$  et  $y_0 = 0, 5$ .

1. Rédiger le programme complet avec seulement la méthode d'Euler sans le calcul d'erreur. Visualiser les résultats pour un pas de 0,2. [figure des solutions sol-euler-02.pdf](#) ← 1
2. Reprendre l'étude avec un pas de 0,1 et commenter.
3. Ajouter le calcul d'erreur et visualiser l'erreur<sup>4</sup> en fonction du temps. Dès que l'erreur est petite devant la solution, ne plus afficher les solutions sur la même figure : l'échelle doit s'adapter à l'erreur maximale. Contrôler l'erreur maximale en valeur absolue et sa position calculées par le programme et les noter.
4. Programmer progressivement les méthodes d'ordre supérieur et tester les pas de 0,2 et de 0,1. Avec chaque méthode, calculer [le rapport entre les erreurs maximales](#) obtenues pour un pas de 0,2 et un pas de 0,1. Le comparer avec le rapport attendu en fonction de l'ordre de la méthode. [figures d'erreur euler-01.pdf](#) et [rk4-01.pdf](#) ← 2  
← 3
5. Reprendre ces calculs entre les pas de 0,1 et 0,01. Avec quelle(s) méthode(s) obtient-on encore une amélioration en passant à un pas de 0,01, puis de 0,001 ? [Expliquer](#) en observant l'allure du signal d'écart. ← 4
6. Reprendre le calcul avec un pas de 0,01 en travaillant en double précision. [figure d'erreur rk4dp-001.pdf](#) Par ← 5  
prudence, on créera un répertoire `double` où l'on recopiera tous les fichiers sources et on procédera comme indiqué en TE1-B.

## 2 Mise en œuvre vectorielle des méthodes explicites à un pas

On note  $n$  le nombre d'instants et  $p$  le nombre de composantes du problème (nombre d'EDO scalaires multiplié par leur ordre). Ces dimensions des tableaux seront fixées à l'exécution par le programme principal.

### 2.1 En fortran (norme 2008)

Les seconds membres des EDO ainsi que les méthodes seront des fonctions à **argument tableau de rang 1**  $\vec{y}$  d'étendue  $p$  et à **résultat tableau** de rang 1 et de même étendue que  $\vec{y}$ .

0. **Les interfaces abstraites** des seconds membres et des solutions analytiques sont déclarées dans le module `abstrait`

```

use mncs_precision
abstract interface
  function fty(t, y) ! de la fonction abstraite passée en argument
    import :: wp ! pb gfortran 4.6 sur sappli1 : "import :: wp" ne fctne pas
    real(kind=wp), intent(in) :: t
    real(kind=wp), dimension(:), intent(in) :: y
    real(kind=wp), dimension(size(y)) :: fty ! résultat vecteur
  end function fty
  ...
end interface

```

---

4. Ne pas tracer sa valeur absolue : le signe de l'erreur est essentiel pour l'interprétation.

1. **Chacune des méthodes à un pas** (Euler, point milieu et Runge Kutta, fichier `methodes.f90`) est une fonction à résultat tableau 1D qui respecte l'interface suivante.

```

function u2_runge_kutta4(u1, t1, h, f)
! méthode de Runge Kutta d'ordre 4 permettant d'avancer d'un pas
  procedure(fty) :: f ! fonction 2nd membre
  real(kind=wp), dimension(:), intent(in) :: u1 ! vecteur initial
  real(kind=wp), intent(in) :: t1 ! instant initial
  real(kind=wp), intent(in) :: h ! pas d'integration
  ! vecteur résultat estimé, au temps t1+h
  real(kind=wp), dimension(size(u1)) :: u2_runge_kutta4 ! résultat vecteur
  ! variables locales
  real(kind=wp), dimension(size(u1)) :: k1, k2, k3, k4 ! pentes (vecteurs)
  ...
end function u2_runge_kutta4

```

Les pentes locales  $\vec{k}_i$  seront des **tableaux automatiques locaux de même étendue  $p$**  que  $\vec{u}_1$ .

2. **Le second membre de l'équation différentielle** (`fcts.f90`) doit respecter l'interface générique déclarée dans les méthodes. L'étendue des tableaux ne doit donc pas être précisée. Mais seules les composantes associées à l'équation différentielle effective doivent être référencées. Par exemple pour le pendule (EDO d'ordre 2), seuls les calculs de `dydt(1)` et `dydt(2)` sont codés :

```

! pendule non linéarisé y'' = -sin(y) décliné en 2 EDO du 1er ordre
! second membre vectoriel du système différentiel dy/dt = dydt(t, y)
function pendule(t, y)
  real(kind=wp), intent(in) :: t
  real(kind=wp), dimension(:), intent(in) :: y
  real(kind=wp), dimension(size(y)) :: pendule
  pendule(1) = ...
  pendule(2) = ...
end function pendule

```

3. **Dans le programme principal** (et dans la procédure d'écriture, fichier `util.f90`), les solutions vectorielles (par intégration et, éventuellement, analytique) sont représentées par des **tableaux 2D alloués dynamiquement**. Ces tableaux 2D sont transmis à la subroutine `ecrit` qui écrit toutes les composantes de la solution sur une seule ligne. Mais la dimension temporelle  $n$  n'est pas « vue » par les méthodes : elles travaillent sur des vecteurs d'étendue  $p$  dans un intervalle  $[t_i, t_{i+1}]$  à  $i$  fixé. La condition initiale  $u_0$  est un tableau 1D.

```

integer :: i, ok
real(kind=wp), dimension(:), allocatable :: t ! tableau des instants
real(kind=wp), dimension(:, :), allocatable :: u, uexact ! tableaux 2D approx et analytiques
real(kind=wp), dimension(:), allocatable :: u0 ! tableau des valeurs initiales

allocate(t(n), u(n,p), uexact(n, p), u0(p), stat=ok)

```

## 2.2 En C99 avec des tableaux automatiques

La déclaration tardive des tableaux automatiques (en particulier dans le programme principal) permet d'éviter les tableaux dynamiques alloués sur le tas.

Les seconds membres des EDO ainsi que les méthodes seront des fonctions à « argument tableau 1D »  $\vec{y}$  pour les arguments d'entrée et de sortie. La taille  $p$  de ces tableaux passés en argument sera déterminée à l'exécution mais ils seront déclarés<sup>5</sup> comme tableaux automatiques par la fonction appelante.

1. **Chacune des méthodes à un pas** (Euler, point milieu et Runge Kutta, fichier `methodes.c`) est une fonction sans valeur de retour avec des arguments tableaux 1D ( $\vec{u}_i$  et le résultat  $\vec{u}_{i+1}$ ) de taille  $p$ . Leur déclaration est prise en charge par l'appelant, soit ici le programme principal. La méthode respecte le prototype suivant :

<sup>5</sup>. Dans le cas du tableau du résultat **final** de la méthode, c'est le programme principal qui déclare le tableau automatique et le transmet à la méthode.

```
// C99 avec tableaux automatiques
void u2_milieu(int p, real u1[p], real t1, real h,
              void (*ptr_f) (real, int, real[], real[]), // pointeur de fct IID membre
              real u2[p]){                               // vecteur résultat à t1 + h
    int j;
    // vecteurs automatiques locaux
    real k1[p];          // pente à gauche en t1
    real k2[p];          // pente au milieu en t1 + h/2
    real u2 [p];         // valeur intermédiaire au milieu (t1+h/2)
    (*ptr_f)(t1, p, u1, k1); // second membre en (t1,u1) dans le vecteur k1
    ...

    // tableau u2 déclaré dans l'appelant et rempli ici dans une boucle
    return ;
}
```

Les pentes locales  $\vec{k}_i$  seront des **tableaux locaux automatiques**, dont le nombre dépend de la méthode.

2. **La fonction second membre de l'équation différentielle** (fichier `fcts.c`) doit respecter le prototype déclaré dans les méthodes. Elle remplit le tableau représentant  $\vec{f}(\vec{y}, t)$  qui a été déclaré par l'appelant, donc dans la méthode. L'étendue des tableaux doit rester une variable. Mais seules les composantes associées à l'équation différentielle effective doivent être référencées. Par exemple pour le pendule (EDO d'ordre 2), seuls les calculs de `dydt[0]` et `dydt[1]` sont codés :

```
void pendule(real t, int p, real u[p], real dydt[p]){
    // pendule non-linéaire

    dydt[0] = ...
    dydt[1] = ...

    return ;
}
```

Le nombre commun `p` d'éléments des tableaux `u` et `dydt` est donc fixé par l'appelant, c'est-à-dire la méthode, qui, elle-même, l'hérite du programme principal. C'est donc le programme principal qui fixe la dimension du problème.

3. **Dans le programme principal** et dans la procédure d'écriture (fichier `util.c`), les solutions vectorielles (par intégration et, éventuellement, analytique) sont représentées par des tableaux automatiques 2D déclarés tardivement. Mais la dimension temporelle n'est pas « vue » par les méthodes : elles travaillent sur des vecteurs dans un intervalle  $[t_i, t_{i+1}]$ . Cela impose que les composantes des vecteurs soient contiguës en mémoire, donc **le deuxième indice est celui des composantes, le premier celui du temps**. La condition initiale  $\vec{y}(t_0)$  est aussi vectorielle de même taille `p`.

```
// C99 tableaux automatiques 2D déclarés dans le main
real u[n][p]; // n instants et p composantes
real u0[p]; // condition initiale à p composantes
```

Dans la boucle d'intégration de l'EDO

```
// appel de la méthode du point milieu par exemple
u2_milieu(p, u[i], t[i], h, &pendule, u[i+1]);
// u[i] et u[i+1] : vecteurs à p composantes
// t[i] : scalaire
```

## 2.3 Plan de travail pour les EDO vectorielles

Pour maîtriser le traitement des EDO du second ordre, on choisit de traiter le cas du **pendule**, transformé en une EDO vectorielle du premier ordre avec  $p=2$  : on rappelle que la condition initiale est un vecteur de taille  $p$  représentant alors position et vitesse initiales. Le système couplé du premier ordre de Lotka-Volterra est plus facile à interpréter, mais son invariant est plus complexe.

1. Créer un répertoire spécifique pour implémenter les codes d'EDO vectorielles du premier ordre en gardant la même structure que pour le scalaire. Commencer par mettre au point le programme principal et les fonctions de la solution analytique avant d'aborder la partie résolution numérique.
2. Traiter d'abord le problème du pendule avec un second membre linéarisé  $y'' = -k^2y$ .
  - (a) Déterminer l'expression **générale** de la solution analytique en fonction des conditions initiales  $y_0$  (position) et  $y'_0$  (vitesse). Coder cette solution et la tracer. Choisir  $k = 1$ ,  $y_0 = 0$  et  $y'_0 = 0.2$  pour commencer.
  - (b) Écrire le système vectoriel du premier ordre de dimension 2 représentant l'EDO scalaire du second ordre. Implémenter la résolution numérique vectorielle. Calculer les deux composantes (position et vitesse) de l'écart avec la solution analytique et vérifier l'effet d'un changement de pas.
3. Compléter avec le second membre exact, non-linéaire.
  - (a) Comparer d'abord la solution numérique avec second membre non-linéaire pour des amplitudes très faibles, avec la solution analytique dans le cas linéaire.
  - (b) Tester ensuite l'évolution de la solution numérique quand  $y'_0$  augmente. Repérer en particulier le passage en régime aperiodique pour  $|y'_0| > 2k$ .
  - (c) Calculer l'énergie mécanique du système à chaque pas de temps et vérifier dans quelle mesure elle se conserve. On représentera l'écart relatif de l'énergie par rapport à la valeur initiale.

## Instructions pour le compte-rendu du TE2 EDO

Le compte-rendu doit comporter les listings des codes mis au point et les figures demandées au format pdf. Les points importants sont encadrés dans le texte et repérés par des numéros en marge : respecter le nommage des figures. Mais la partie analyse et interprétation des résultats est évidemment la plus importante : elle doit notamment comporter **la comparaison quantitative** des résultats numériques et des prévisions d'origine théorique (ce qui nécessite quelques calculs supplémentaires à partir des courbes), ainsi que des explications sur les écarts constatés.

La restitution se fait sous deux formes : un compte-rendu papier avec codes et figures commentées d'une part et un fichier d'archive compressé d'autre part.

1. Indiquer votre nom, prénom (sans accent) et numéro d'étudiant en commentaire, en début de tous les fichiers source. On rappelle les caractères introducteurs de commentaire selon les langages : `//` en C99, `!` en fortran, `#` pour `make` et `python`.
2. Créer un répertoire spécifique (un renommage peut suffire) **nommé impérativement** `TE2-nom-prenom` où `nom` et `prenom` sont vos nom et prénom sans signe diacritique ni espace.
3. Y copier tous les fichiers à rendre : en particulier les sources et les entêtes en C, le `makefile` et le fichier source pour `python`, ainsi que les figures en format pdf. Pour que l'exécutable puisse être lancé par redirection, les jeux de paramètres d'entrée devront aussi être fournis sous forme de fichiers texte de suffixe `.in`. Indiquer avec précision les noms des fichiers d'entrée dans le compte-rendu et sur les figures.
4. Choisir comme répertoire de travail le répertoire père de `TE2-nom-prenom`. Lancer la commande `tar cvzf TE2-nom-prenom.tgz TE2-nom-prenom`  
Vérifier (option `t` de `tar`) que le contenu de l'archive compressée contient tous les fichiers nécessaires.
5. Copier le fichier d'archive compressé dans le répertoire `/home/lefrere/M1/TE2-jeudi/` ou `/home/lefrere/M1/TE2-vendredi/`, selon le jour.