# Méthodes numériques de résolution d'équations différentielles

Brian Stout
brian.stout@fresnel.fr

Université de Provence
Institut Fresnel, Case 161 Faculté de St Jérôme
Marseille, France

Fevrier 2007

## Table des matières

Une équation différentielle est une équation qui dépend d'une variable $t$ et d'une fonction $x(t)$ et qui contient des dérivées de $x(t)$. Elle s'écrit :

$$F\left(t, x(t), x^{(1)}(t), \ldots, x^{(m)}(t)\right) = 0 \qquad \text{où} \qquad x^{(m)}(t) \equiv \frac{d^m x}{dt^m} \tag{1}$$

L'ordre de cette équation est déterminé par sa dérivée d'ordre le plus élevé. Donc l'équation (1) est d'ordre $m$.

La solution du problème consiste à trouver une fonction $x(t)$ qui soit solution de (1) et dérivable sur un intervalle fini de $t \in [t_0, t_0 + T]$ de $\mathbb{R}$. Souvent dans les applications, la variable $t$ représente le temps, et $t_0$ *est alors l'instant initial*. En général, il n'existe une solution unique à une équation différentielle qu'une fois certaines conditions limites imposées sur $x(t)$ et ses dérivées. Dans l'exemple de l'équation (1) les *conditions initiales* sont les valeurs de $x(t_0)$, $x^{(1)}(t_0), \ldots, x^{(m-1)}(t_0)$.

# 1 Problème de Cauchy :

La plupart des méthodes numériques pour résoudre les équations différentielles s'appliquent à des problèmes du type *problème de Cauchy* suivant le nom donné par les mathématiciens. Ce problème se formule de la manière suivante :

Trouver $y(t)$ définie et dérivable sur $[t_0, t_0 + T]$ et à valeurs dans $\mathbb{R}^m$ telle que :

$$\begin{cases} \frac{dy(t)}{dt} = f(t, y(t)) & \forall t \in [t_0, t_0 + T] \\ \\ y(t_0) = y_0 \end{cases} \tag{2}$$

où $f(t, y(t))$ est une fonction de $\mathbb{R}^{m+1}$ dans $\mathbb{R}^m$ et $y_0 \in \mathbb{R}^m$. Concrètement l'expression, "trouver $y(t)$ à valeurs dans $\mathbb{R}^m$ avec $y_0 \in \mathbb{R}^m$" consiste à dire pour des applications comme Matlab, que l'inconnue $y(t)$ est un vecteur de $m$ fonctions inconnues avec pour condition limite le vecteur $y_0$ :

$$y(t) = \begin{bmatrix} y_1(t) \\ y_2(t) \\ \vdots \\ y_m(t) \end{bmatrix} \qquad y_0 = y(t_0) = \begin{bmatrix} y_1(t_0) \\ y_2(t_0) \\ \vdots \\ y_m(t_0) \end{bmatrix} = \begin{bmatrix} y_{0,1} \\ y_{0,2} \\ \vdots \\ y_{0,m} \end{bmatrix} \tag{3}$$

De même, $f(t, y(t))$ est une fonction de $t$ et du vecteur $y(t)$ et doit retourner un vecteur colonne :

$$\frac{dy(t)}{dt} \equiv \frac{d}{dt}\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix} = f(t, y(t)) \equiv \begin{bmatrix} f_1 \\ f_2 \\ \vdots \\ f_m \end{bmatrix} \tag{4}$$

Pour la plupart des problèmes qui intéressent les scientifiques et les ingénieurs, des théorèmes mathématiques assurent l'existence et l'unicité d'une solution au problème de Cauchy. Néanmoins, souvent la solution ne peut être exprimée *analytiquement*. Pour de tels problèmes, on doit donc chercher à déterminer la fonction $y(t)$ par des méthodes *numériques*.

# 2 Transformations vers un problème de Cauchy

Dans Matlab (Octave), de puissant programmes (fonctions) existent sous le nom générique de ODEs (Ordinary Differential Equation Solvers). Ils résolvent les systèmes de la forme de l'équation (2). Le travail principal d'un utilisateur de Matlab consiste donc le plus souvent à transformer son problème sous la forme de l'équation (2). Dans bien des domaines, surtout ceux des équations à dérivées partielles, les transformations d'un problème donné sous la forme d'un problème de Cauchy sont toujours d'actualité comme problèmes de recherche.

## 2.1 Traitement d'une équation différentielle d'ordre $> 1$

Dans ce cours, nous ne regarderons que la transformation d'une équation différentielle d'ordre supérieur à 1, en problème de Cauchy. Considérons donc une équation différentielle d'ordre $m$ de la forme suivante :

$$x^{(m)}(t) \equiv \frac{dx^{(m-1)}}{dt} = \varphi\left(t, x(t), x^{(1)}(t), \ldots, x^{(m-1)}(t)\right) \qquad \forall t \in [t_0, t_0 + T] \tag{5}$$

Posons de nouvelles fonctions $y_i(t)$ avec $i \in [1, 2, \ldots, m]$ définies telles que :

$$y_1(t) \equiv x(t), \quad y_2(t) \equiv x^{(1)}(t) \quad , \ldots, \quad y_m(t) \equiv x^{(m-1)}(t) \tag{6}$$

Grâce à ces définitions, l'équation (5) d'ordre $m$ s'écrit comme un système de $m$ équations d'ordre 1 :

$$\begin{cases} \frac{dy_1(t)}{dt} = y^{(2)}(t) \\ \vdots \\ \frac{dy_{m-1}(t)}{dt} = y^{(m)}(t) \\ \frac{dy_m(t)}{dt} = \varphi(t, y_1(t), y_2(t), \ldots, y_m(t)) \end{cases} \tag{7}$$

Ce système a donc la forme d'un problème de Cauchy en posant :

$$y(t) = \begin{bmatrix} y_1(t) \\ \vdots \\ y_{m-1}(t) \\ y_m(t) \end{bmatrix} \quad \text{et} \quad f(t, y(t)) = \begin{bmatrix} y_2(t) \\ \vdots \\ y_m(t) \\ \varphi(t, y_1, \ldots, y_m) \end{bmatrix} \tag{8}$$

L'équation (5) s'écrira alors :

$$\frac{dy(t)}{dt} = f(t, y(t)) \qquad \forall t \in [t_0, t_0 + T] \tag{9}$$

Pour obtenir alors un problème de Cauchy, il faut spécifier les conditions initiales ($y_1(t_0), y_2(t_0), \ldots, y_m(t_0)$) ce qui revient à dire d'après l'équation (6), qu'il faut connaître $x(t)$ et ses dérivées jusqu'à l'ordre $m-1$ au 'temps' initial $t_0$ : $\left(x(t_0), x^{(1)}(t_0), \ldots, x^{(m-1)}(t_0)\right)$. On remarque qu'une équation différentielle d'ordre $m$ d'une seule fonction inconnue, $x(t)$, se traduit par un problème de Cauchy avec $m$ fonctions inconnues, $y_i(t)$, et $m$ conditions initiales.

## 2.2 Equations différentielles à coefficients constants

En particulier, les équations différentielles à coefficients constants constituent une classe d'équations de la forme de l'éq.(5). Notamment quand $\varphi$ est de la forme :

$$\varphi\left(t, x\left(t\right), x^{(1)}\left(t\right), \ldots, x^{(m-1)}\left(t\right)\right) = s\left(t\right) - a_1 x\left(t\right) - a_2 x^{(1)}\left(t\right) - \ldots - a_m x^{(m-1)}\left(t\right) \tag{10}$$

l'équation l'éq.(5) peut s'écrire comme une équation différentielle à coefficients constants :

$$a_1 x\left(t\right) + a_2 x^{(1)}\left(t\right) + \ldots + a_m x^{(m-1)}\left(t\right) + x^{(m)}\left(t\right) = s\left(t\right) \tag{11}$$

où la fonction $s\left(t\right)$ est communément appelée un terme source.

Pour des équations de la forme de l'éq.(11), les substitutions de l'éq.(6) amènent à un système d'équations de forme matricielle. Par exemple, une équation à coefficients constants d'ordre 4 s'écrit :

$$a_1 x\left(t\right) + a_2 x^{(1)}\left(t\right) + a_3 x^{(2)}\left(t\right) + a_4 x^{(3)}\left(t\right) + x^{(4)}\left(t\right) = s\left(t\right) \tag{12}$$

Après les substitutions de l'équation (6), cette équation s'écrit :

$$a_1 y_1\left(t\right) + a_2 y_2\left(t\right) + a_3 y_3\left(t\right) + a_4 y_4\left(t\right) + \frac{d}{dt} y_4\left(t\right) = s\left(t\right) \tag{13}$$

et l'équation (9) peut s'écrire sous une forme matricielle :

$$\frac{d}{dt}\begin{bmatrix} y_1\left(t\right) \\ y_2\left(t\right) \\ y_3\left(t\right) \\ y_4\left(t\right) \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ -a_1 & -a_2 & -a_3 & -a_4 \end{bmatrix} \begin{bmatrix} y_1\left(t\right) \\ y_2\left(t\right) \\ y_3\left(t\right) \\ y_4\left(t\right) \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ s\left(t\right) \end{bmatrix} \tag{14}$$

Même s'il est intéressant de voir ce type de problème comme une équation matricielle, nous ne devons pas oublier que la formulation de l'équation (9) nous permet de traiter bien des problèmes qui ne prennent pas la forme d'une équation matricielle. On remarque aussi qu'il y a beaucoup de zéros dans l'équation (14) et donc une multiplication de matrice n'est pas la façon la plus éfficace de programmer $f\left(t, y\left(t\right)\right)$ (Voir la fonction (A) de la section 2.3 ci-dessous).

## 2.3 Exemple - Vol d'un point solide dans un champ de pesanteur.

Imaginons qu'on cherche à résoudre numériquement le problème du mouvement d'un point solide de masse $m$ à la position $\overrightarrow{\boldsymbol{x}}\left(t\right) = x\widehat{\boldsymbol{x}} + y\widehat{\boldsymbol{y}} + z\widehat{\boldsymbol{z}}$ ayant une vitesse $\overrightarrow{\boldsymbol{v}} = \frac{d\overrightarrow{\boldsymbol{x}}}{dt}$ dans un champ de pesanteur $\overrightarrow{\boldsymbol{g}}$. (figure 1)

La mécanique du point nous dit qu'il suffit d'appliquer la relation fondamentale de la dynamique au point solide :

$$m\frac{d\overrightarrow{\boldsymbol{v}}}{dt} = \overrightarrow{\boldsymbol{P}} = m\overrightarrow{\boldsymbol{g}} \tag{15}$$

Puisqu'il s'agit d'une équation vectorielle, nous avons en principe trois équations scalaires à résoudre, mais nous savons que le vol du point s'effectue dans un plan parallèle au plan défini par $(xOz)$. On arrive donc à un système de deux équations différentielles de deuxième ordre à résoudre :

$$\begin{cases} \frac{d^2 x}{dt^2} = 0 \\ \frac{d^2 z}{dt^2} = -g \end{cases} \tag{16}$$

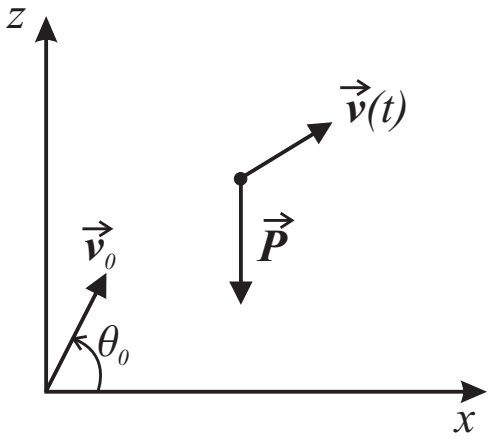


FIG. 1 – Mouvement d'un point de masse dans un champ de pesanteur

Avec les conditions limites

$$\begin{aligned} x\left(t_0\right) &= x_0 \qquad x^{(1)}\left(t_0\right) = v_{0,x} \\ z\left(t_0\right) &= z_0 \qquad z^{(1)}\left(t_0\right) = v_{0,z} \end{aligned} \tag{17}$$

nous connaissons la solution exacte de chacune de ces deux équations :

$$\begin{aligned} x\left(t\right) &= x_0 + v_{0,x}t \\ z\left(t\right) &= z_0 + v_{0,z}t - \frac{1}{2}gt^2 \end{aligned} \tag{18}$$

Nous voulons simplement tester notre capacité à trouver la solution de façon numérique. La connaissance d'une solution exacte nous permet de tester différentes méthodes de résolution numérique d'équations différentielles.

Pour résoudre les équations différentielles d'ordre 2 de l'éq.(16) on va définir des fonctions du système $u\left(t\right)$ (pour ne pas confondre avec la position $y\left(t\right)$) et invoquer les substitutions de l'éq.(6) :

$$\begin{aligned} u_1\left(t\right) &\equiv x\left(t\right) \\ u_2\left(t\right) &\equiv x^{(1)}\left(t\right) = v_x\left(t\right) \end{aligned} \tag{19}$$

L'équation $\frac{d^2x}{dt^2} = 0$ devient donc le système matriciel :

$$\frac{d}{dt}\begin{bmatrix} u_1\left(t\right) \\ u_2\left(t\right) \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}\begin{bmatrix} u_1\left(t\right) \\ u_2\left(t\right) \end{bmatrix} \tag{20}$$

De même on peut définir

$$\begin{aligned} u_3\left(t\right) &\equiv z\left(t\right) \\ u_4\left(t\right) &\equiv z^{(1)}\left(t\right) = v_z\left(t\right) \end{aligned} \tag{21}$$

et le système $\frac{d^2z}{dt^2} = -g$ devient :

$$\frac{d}{dt}\begin{bmatrix} u_3\left(t\right) \\ u_4\left(t\right) \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}\begin{bmatrix} u_3\left(t\right) \\ u_4\left(t\right) \end{bmatrix} + \begin{bmatrix} 0 \\ -g \end{bmatrix} \tag{22}$$

On peut regrouper ces deux équations sous la forme d'une seule grande équation matricielle :

$$\frac{du}{dt} \equiv \frac{d}{dt}\begin{bmatrix} u_1(t) \\ u_2(t) \\ u_3(t) \\ u_4(t) \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} u_1(t) \\ u_2(t) \\ u_3(t) \\ u_4(t) \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ -g \end{bmatrix} \tag{23}$$

ou de manière équivalente :

$$\frac{du}{dt} = f(t, u(t)) = \begin{bmatrix} u_2(t) \\ 0 \\ u_4(t) \\ -g \end{bmatrix} \tag{24}$$

La solution de cette équation va donc nous fournir les fonctions : $x(t) = u_1(t)$, $v_x(t) = u_2(t)$, $z(t) = u_3(t)$, et $v_z(t) = u_4(t)$. On peut facilement programmer une fonction en Octave/Matlab pour calculer $f(t, u(t))$ :

```
function f = fprojectile(u,t)        % en Matlab - 'function f = fprojectile(t,u)'
% fonction f (t, u (t)) pour une particule dans un champ de pesanteur
  nc = length(u) ;
  f = zeros(nc,1) ;
  f(1) = u(2) ;
  f(2) = 0 ;
  f(3) = u(4) ;
  f(4) = -9.8 ;                  % valeur de g
end
```

(A)

**Important :** Il est très important que la fonction retourne un *vecteur colonne*. Les fonctions Matlab (Octave) pour résoudre une équation différentielle ne marchent pas si la fonction retourne un vecteur ligne. L'écriture de la fonction 'fprojectile' permet à l'argument $u$ d'être un vecteur ligne ou un vecteur colonne. Il s'avère assez commode de prendre $u$ comme un vecteur ligne.

Maintenant on peut utiliser la fonction 'lsode' d'Octave ('ODE45' de MatLab) pour résoudre numériquement l'équation différentielle. Cette fonction prend en argument le nom de la fonction $f(t, u(t))$, un vecteur contenant les valeurs de $t = [t_0, t_1, \ldots, t_N]$ pour lesquelles on veut connaître les valeurs de $u(t)$, $[u(t_0), u(t_1), \ldots, u(t_N)]$. Pour certaines applications, on ne s'intéressera qu'à une seule et unique valeur $u(t_N)$ ; dans ce cas, on donne simplement un vecteur $t = [t_0, t_N]$. Dans les cas où il faut connaître une trajectoire, il faut que $t = [t_0, t_1, \ldots, t_N]$ contienne suffisamment d'éléments pour que les courbes générées par 'plot' paraissent lisses.

Le script suivant résout l'équation différentielle de l'équation (16) avec les conditions initiales $x_0 = z_0 = 0$ , $v_0 \equiv \left\|\vec{\boldsymbol{v}}_0\right\| = 100\text{ms}^{-1}$, $\theta_0 \equiv (\widehat{\boldsymbol{x}}, \vec{\boldsymbol{v}}_0) = 30°$ et montre la position de la particule pour $N + 1 = 31$ temps compris entre 0 et 11s.

```
tmin = 0 ;          % temps initial t_0
tmax = 11 ;         % temps finale t_0 + T
v0 =100 ;           % vitesse intiale en mètres par seconde
thetdeg =30 ;       % angle initial en degrés
x0 = 0 ;            % position x_0 initiale
z0 = 0 ;            % position z_0 initiale
vx0 = v0*cos(thetdeg*pi/180) ;      % vitesse v_{x,0} initiale
vz0 = v0*sin(thetdeg*pi/180) ;      % vitesse v_{z,0} initiale
Nint = 30 ;         % nombre d'intervalles : N
```

```
h = (tmax-tmin)/Nint ;                % taille du pas  h = t_{n+1} - t_n
t = linspace(tmin,tmax,Nint+1) ;      % vecteur des t = [t_0, t_1, ... t_{N-1}, t_N = t_0 + T]
u0 = [x0 vx0 z0 vz0] ;                % conditions initiales u_0 = [x_0 v_{0,x} z_0 v_{0,z}]
usol =lsode("fprojectile",u0,t) ;     % résolution numérique par Octave
% [t,usol] = ODE45('fprojectile',t,u0) ;   % résolution numérique par Matlab
xpos = usol( :,1) ;                   % position des x  : première colonne de usol
zpos = usol( :,3) ;                   % position des z : troisième colonne de usol
plot(xpos,zpos,'o')                   % plot de z en fonction de x.
```

(B)

Il faut remarquer que la matrice 'usol' est une matrice avec $N+1$ lignes pour les $N+1$ valeurs de $t_n$ et 4 colonnes correspondant respectivement à la position et à la vitesse suivant $x$ et à la position et à la vitesse suivant $z$ : $u(t_n) = [x(t_n), v_x(t_n), z(t_n), v_z(t_n)]$. On illustre sur la figure (2) une comparaison entre la solution générée par 'ODE45' de Matlab ('lsode' d'Octave) et la solution exacte. L'erreur absolue, $\sqrt{(u_1(t_N) - x(t_N))^2 - (u_3(t_N) - z(t_N))^2}$, sur la position du point solide après 11 secondes de vol est de l'ordre de $10^{-13}$m sous Matlab.

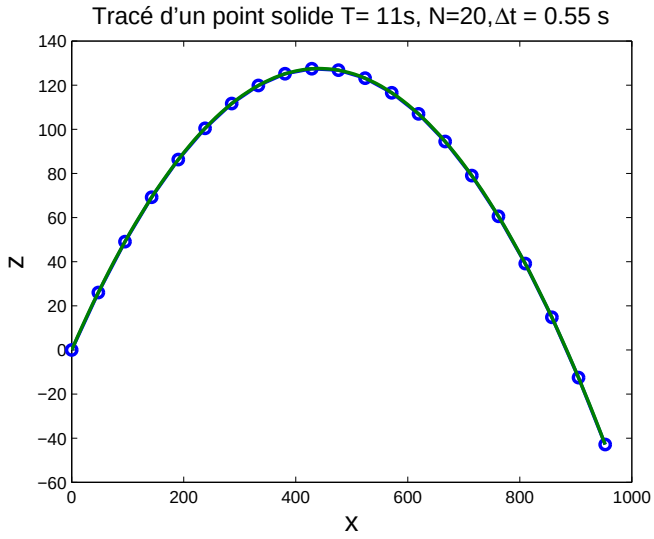


FIG. 2 – Mouvement d'un point de masse $m$, de vitesse $\boldsymbol{v}$ dans le champ de pesanteur $\boldsymbol{g}$. Solution de ODE45 avec $x_0 = z_0 = 0$, $v_0 = 100\text{ms}^{-1}$, et $\theta_0 = 30°$

## 2.4 Détermination des paramètres initiaux

Un des buts souvent recherchés lors des calculs différentiels est de déterminer où "d'optimiser" les paramètres initiaux afin d'obtenir un certain comportement désiré de la solution. Par exemple, dans le cas d'un projectile on pourrait s'intéresser au problème suivant : pour quel angle de lancement le projectile ira-t-il le plus loin ? Dans ce cas, on pourrait donc écrire une fonction simple qui trouve de façon approximative, le point d'atterrissage du point solide. Une telle fonction peut comporter d'abord une boucle qui permet de trouver les deux instants entre lesquels le projectile a dû toucher le sol. Plus précisément, la boucle chercherait l'instant $t_n$ où $z(t_n) > 0$ et $z(t_{n+1} = t_n + h) < 0$. On peut ensuite faire une approximation linéaire entre ces deux points, i.e. on cherche à déterminer les coefficients $a$ et $b$ d'une fonction linéaire $z(t) = at + b$

tels que :

$$z_n \equiv z(t_n) = at_n + b \tag{25a}$$
$$z_{n+1} \equiv z(t_{n+1}) = at_{n+1} + b \tag{25b}$$

En soustrayant l'éq.(25a) de l'éq.(25b), on obtient :

$$z_{n+1} - z_n = a\,(t_{n+1} - t_n) \tag{26}$$

d'où on tire une expression pour $a$ :

$$a = \frac{z_{n+1} - z_n}{(t_{n+1} - t_n)} \tag{27}$$

On remet cette expression pour $a$ dans l'éq.(25a) et on trouve :

$$z_n = \frac{(z_{n+1} - z_n)\,t_n + b\,(t_{n+1} - t_n)}{(t_{n+1} - t_n)} \tag{28}$$

ce qui donne après simplification :

$$b = \frac{z_n t_{n+1} - t_n z_{n+1}}{t_{n+1} - t_n} \tag{29}$$

Le point d'atterrissage, $t_\mathrm{a}$, est donc le zéro de la fonction linéaire $z\,(t_\mathrm{a}) = at_\mathrm{a} + b = 0$, i.e. $t_\mathrm{a}$ est donné par :

$$at_\mathrm{a} = -b$$
$$t_\mathrm{a} = \frac{z_n t_{n+1} - t_n z_{n+1}}{(z_n - z_{n+1})} \tag{30}$$

En général, la vitesse sur $x$ n'est pas constante, et on peut faire une approximation linaire sur $x\,(t)$ :

$$x_n \equiv x(t_n) = ct_n + d \tag{31}$$
$$x_{n+1} \equiv x(t_{n+1}) = ct_{n+1} + d \tag{32}$$

Une approximation à la position horizontale (des $x$) au point d'atterrissage est maintenant :

$$x\,(t_\mathrm{a}) = \frac{x_{n+1} - x_n}{(t_{n+1} - t_n)} t_\mathrm{a} + \frac{x_n t_{n+1} - t_n x_{n+1}}{t_{n+1} - t_n} \tag{33}$$

Et le point d'atterrissage est :

$$(t_\mathrm{a}, x\,(t_\mathrm{a})\,, z = 0) \tag{34}$$

Le fonction 'point_atterrissage' suivante écrite sous Matlab (Ocatve) permet d'effectuer ce calcul du point d'atterissage. Les arguments de cette fonction sont les trois vecteurs : $t = [t_0, t_1, \ldots, t_N]$, $x = [x(t_0), x(t_1), \ldots, x(t_N)]$, et $z = [z(t_0), z(t_1), \ldots, z(t_N)]$. Elle ne fonctionne de manière satisfaisante que si l'intervalle $\Delta t \equiv t_{i+1} - t_i$ est suffisamment petit.

```
function [ta,xa] = point_atterrissage(t,x,z)
% trouver le point d'atterissage d'un point solide
nc = length(t) ;
n = 1 ;      % boucle pour trouver l'interval [t_n,t_n+1] où z(t_n+1)< 0
```

```
while( z(n+1) > 0 && n<nc )
      n = n+1 ;
end  % while                                                    (C)
 if (n ~= nc)
     ta=( z(n)*t(n+1)-t(n)*z(n+1) ) / ( z(n)-z(n+1)) ;
     Dt = t(n+1) - t(n) ;
     xa = ta*(x(n+1) - x(n))/Dt + (t(n+1)*x(n)-t(n)*x(n+1)) /Dt ;
  else
     disp('Pas de point atterissage dans les données')
      ta = -1 ;
      xa = -1 ;
  end  %end if
end
```

On voit en figure (3) les tracées des trajectoires et le point d'atterrissage de trois tirs avec différents angles de lancement avec $v_0 \equiv \left\|\overrightarrow{v}_0\right\| = 100\text{ms}^{-1}$. Bien entendu, en l'occurrence un calcul analytique simple nous permettrait de trouver l'angle 'optimal' de lancement. Néanmoins, des modélisations plus réalistes (ex. forces de frottement non-linéaires) peuvent facilement mener à des situations où un calcul analytique devient impossible.

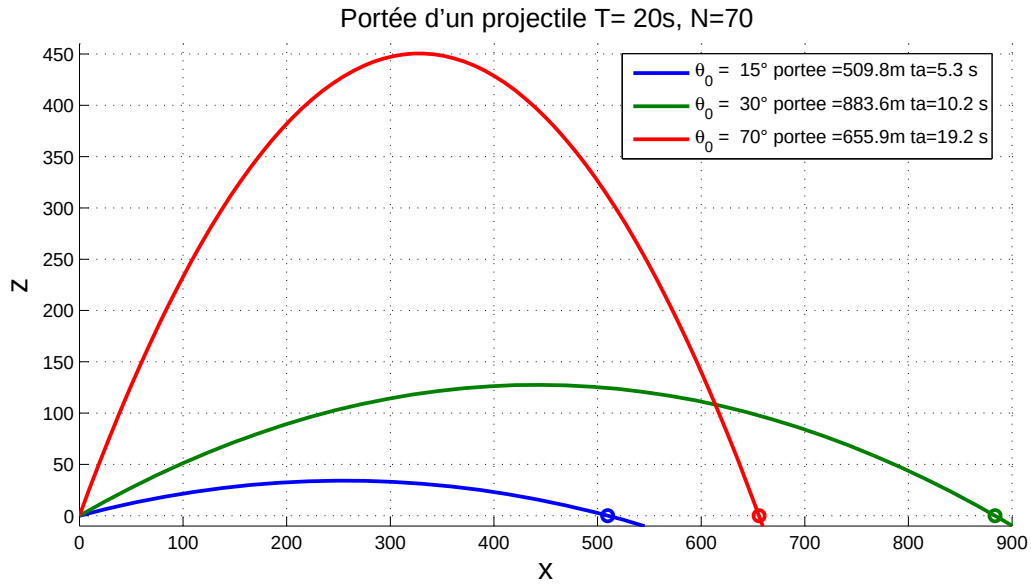


FIG. 3 – Tracées des trajectoires et le point d'atterrissage de trois tirs avec différents angles de lancement.

Nous avons vu dans cette section l'utilité d'avoir une résolution fiable et rapide d'équations différentielles. Dans la prochaine section nous étudierons des méthodes de résolution numérique d'équations différentielles.

## 3 Solutions numériques des équations différentielles

Il existe un grand nombre de façons de résoudre une équation différentielle et aucune méthode n'est clairement supérieure à toutes les autres dans toutes les circonstances. De nos jours, la

technique Runge Kutta d'ordre 4 et à pas adaptatif est souvent utilisée. Nous verrons les raisons de ce choix dans ce qui suit. Néanmoins, les recherches dans ce domaine ne sont pas encore terminées et le développement des techniques numériques continue, surtout pour les situations où une très grande précision est demandée.

Dans tout ce qui suit, il faut garder à l'esprit que dans le cas le plus général, $y$ est un vecteur colonne, et $f(t, y(t))$ est un "vecteur" dont les "éléments" sont des fonctions. Néanmoins, afin de simplifier la discussion, nous ne considérons dans les exemples que des équations simples à une seule fonction inconnue. Par la suite, dans la section 3.5, nous verrons les généralisations qui permettent de traiter multiples fonctions inconnues $y_i(t)$.

## 3.1 Formulation générale

On commence en remarquant qu'une solution formelle de l'équation

$$\frac{d}{dt} y(t) = f(t, y(t)) \tag{35}$$

est obtenue en intégrant de $t_0$ à $t$ :

$$\begin{aligned} \int_{y(t_0)}^{y(t)} dy &= \int_{t_0}^{t} f(s, y(s))\, ds \\ y(t) - y(t_0) &= \int_{t_0}^{t} f(s, y(s))\, ds \end{aligned} \tag{36}$$

donc la solution $y(t)$ s'écrit

$$y(t) = y_0 + \int_{t_0}^{t} f(s, y(s))\, ds \tag{37}$$

où $y_0 \equiv y(t_0)$. Le problème avec cette solution formelle est que l'inconnue $y(t)$ se trouve sous l'intégrale.

## 3.2 Méthode itérative de Picard

Dans la méthode de Picard, on peut itérer l'équation (37) afin de générer une série d'approximations à $y(t)$, dénotées ${}_1y(s)$, ${}_2y(s)$, ${}_ky(s)$,... . On initialise l'itération en prenant $y(t) \simeq y_0$ sous l'intégrale :

$$\begin{aligned} {}_1y(t) &= y_0 + \int_{t_0}^{t} f(s,\ y_0)\, ds \\ {}_2y(t) &= y_0 + \int_{t_0}^{t} f(s,\ {}_1y(s))\, ds \\ &\vdots \\ {}_ky(t) &= y_0 + \int_{t_0}^{t} f(s,\ {}_{k-1}y(s))\, ds \end{aligned} \tag{38}$$

La méthode de Picard est surtout utilisée dans les études mathématiques des équations différentielles. Néanmoins, cette méthode peut s'avérer intéressante quand il est possible de faire une intégration analytique de la suite $\int_{t_0}^{t} f(s, {}_k y(s))\, ds$. Les logiciels de calcul symbolique comme Mathématica et Maple sont capable de calculer ce type d'intégrale.

### 3.2.1 Exemple : méthode de Picard pour résoudre l'équation $\frac{d}{dt}y(t) = t - y(t)$

Dans ce problème, il n'y a qu'une seule fonction inconnue $y(t)$. Ce problème est suffisamment simple que l'on puisse trouver une solution analytique. Prenons la condition limite $(0, 1)$ i.e. $y(t_0 = 0) = 1$ ; la solution analytique de l'équation

$$\frac{d}{dt}y(t) = t - y(t) \tag{39}$$

est obtenue par les techniques habituelles d'équations différentielles à coefficients constants. Vous pouvez vérifier que la solution de (39) est :

$$y(t) = (y_0 + 1)\, e^{-t} + t - 1 \tag{40}$$

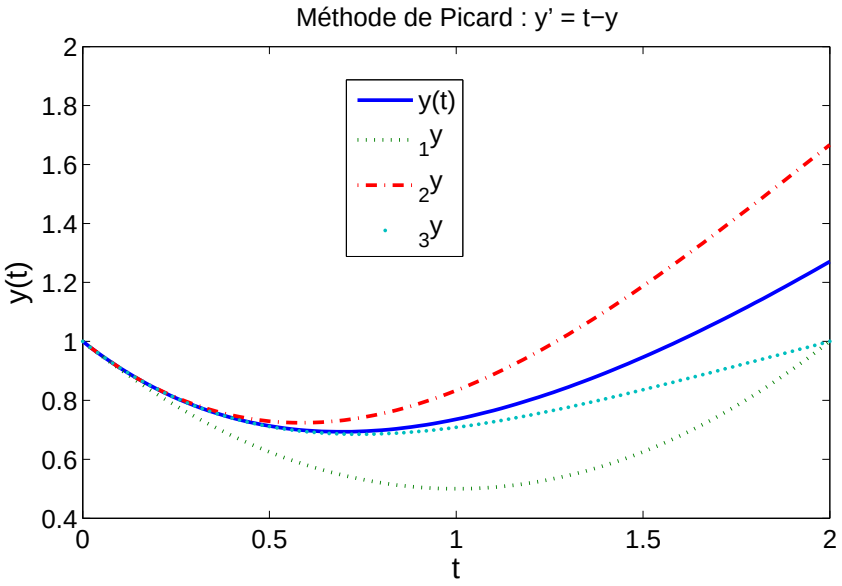


FIG. 4 – Méthode de Picard. Comparaison des solutions de $\frac{d}{dt}y(t) = t - y(t)$ avec les approximations de Picard ${}_1y$, ${}_2y$, ${}_3y$.

Avec la condition limite $(0, 1)$ i.e. $y(t_0 = 0) = 1$, la méthode de Picard donne la suite :

$$\begin{aligned}
{}_1y(t) &= 1 + \int_0^t (s-1)\, ds = 1 - t + \frac{t^2}{2} \\
{}_2y(t) &= 1 + \int_0^t \left[ s - \left( 1 - s + \frac{s^2}{2} \right) \right] ds = 1 - t + t^2 - \frac{t^3}{6} \\
{}_3y(t) &= 1 + \int_0^t \left[ s - \left( 1 - t + t^2 - \frac{t^3}{6} \right) \right] ds = 1 - t + t^2 - \frac{t^3}{3} + \frac{t^4}{24} \\
&\vdots
\end{aligned} \tag{41}$$

En dépit de son efficacité, la méthode de Picard n'est intéressante que pour un nombre limité de problèmes où nous sommes capables d'effectuer analytiquement les intégrations. On préfère

en général des techniques qui sont de nature plus numérique mais de domaine d'application plus vaste.

Sur la figure (4), on représente une comparaison des solutions de la suite de Picard de l'équation (41) à la solution exacte. Ce graphique est généré par le code scripte, (D), ci-dessous. La série de solutions de Picard illustre une propriété souvent rencontrée par des solutions numériques aux équations différentielles, notamment que plus l'intervalle $T$ sur lequel on veut trouver la solution est grande, plus la solution numérique risque de s'éloigner de la solution exacte quand $t \to T$. Néanmoins on doit garder à l'esprit que l'ampleur de ce phénomène dépend de la nature de l'équation différentielle et de la méthode employée pour résoudre l'équation.

La Figure (4) est générée par le code suivant :

```
function Picard(tmax,Nint)
% Compare les 3 premiers termes dans la suite de Picard ...
% avec la solution exacte de y' = t - y conditions limites (0,1)
% ici t0 = 0 , et y0 = 1
  t = linspace(0,tmax,Nint+1);                                        (D)
  yanal = 2*exp(-t) + t - 1;                % solution analytique
  y1 = 1 - t + t.^2/2;                      % Approximation de Picard : 1y
  y2 = 1 - t + t.^2 - t.^3/6;               % Approximation de Picard : 2y
  y3 = 1 - t + t.^2 - t.^3/3 + t.^4/24;     % Approximation de Picard : 3y
  axes('FontSize',14);
  plot(t,yanal,t,y1,' :',t,y2,'-.',t,y3,'.','LineWidth',2);
  xlabel(' t ','FontSize',16);
  ylabel(' y(t) ','FontSize',16);
  legend('y(t)','_1y','_2y','_3y')
end
```

On doit remarquer que nous avons simplifié notre traitement de la suite de Picard par un choix judicieux des conditions initiales. Les choses se compliquent un peu si l'on veut utiliser la suite de Picard pour $y_0$ et $t_0$ arbitraires :

$$
\begin{aligned}
{}_1y(t) &= y_0 + \int_{t_0}^{t} (s-1)\,ds = y_0 + y_0\,(t_0 - t) + \frac{1}{2}\left(t^2 - t_0^2\right) \\
{}_2y(t) &= y_0 + \int_{t_0}^{t} \left\{ s - \left[ y_0 + y_0\,(t_0 - s) + \frac{1}{2}\left(s^2 - t_0^2\right) \right] \right\} ds \\
&= y_0 + \frac{1}{6}\left\{ t^2\,(3-t) + 3\,\left(t^2 - 2\,t + 2\right)\,y_0 + 6\,(1-t)\,t_0\,y_0 + 3\,t_0^2\,(t + y_0 - 1) - 2\,t_0^3 \right\} \\
&\vdots
\end{aligned}
\tag{42}
$$

### 3.3 Méthodes basées sur la série de Taylor

Les méthodes basées sur la série de Taylor se prêtent bien à des traitements numériques. Commençons par subdiviser l'intervalle $[t_0, t_0 + T]$ en $N$ sous intervalles de même longueur $h$ et appelons $t_n$ les points de subdivision. Nous avons donc :

$$
h = \frac{T}{N} \qquad \text{et} \qquad t_n = t_0 + nh \qquad \text{pour} \qquad n = 0, \ldots, N \tag{43}
$$

Par conséquent en MatLab, $t$ est un vecteur avec $N + 1$ éléments :

$$
t = [t_0, \ldots, t_N] \tag{44}
$$

Le développement de la série de Taylor de $y(t_{n+1})$ jusqu'à l'ordre $m$ autour du point $t_n$ s'écrit :

$$y(t_{n+1}) = y(t_n) + hy^{(1)}(t_n) + \frac{h^2}{2!}y^{(2)}(t_n) + \ldots + \frac{h^m}{m!}y^{(m)}(t_n) + O\left(h^{m+1}\right) \tag{45}$$

### 3.3.1 Méthode d'Euler

Si on arrête la série de Taylor à l'ordre 1 on obtient :

$$y(t_{n+1}) \simeq y(t_n) + hy^{(1)}(t_n) \tag{46}$$

L'application de cette formule pour calculer les $y(t_n)$ est connue comme *méthode d'Euler*. On remarque que l'erreur commise à chaque étape est de l'ordre de $O\left(h^2\right)$. Il est clair qu'on peut augmenter la précision de cette méthode en diminuant la taille de $h$, mais un $h$ petit implique qu'on doit évaluer la fonction un grand nombre de fois ce qui est coûteux en temps d'exécution. Qui plus est, un grand nombre d'évaluations de la fonction peut provoquer un cumul des erreurs numériques commises par la machine.

En Matlab, on peut facilement programmer la méthode d'Euler avec la fonction suivante :

```
function [t,y] = Euler(f,tmin,tmax,Nint,y0)   % Méthode d'Euler
 % Nint - nombre de sous intervalles N
 % tmin - temps t0
 % tmax - temps t0 + T
 % f est une fonction avec comme arguments t et y(t) : f(t,y(t))
 % y0 à l'entrée est composé des valeurs des conditions limites y0 = y(t0)
  h = (tmax-tmin)/Nint ;                  % valeur du pas
  t = linspace(tmin,tmax,Nint+1) ;        % vecteur de t discrétisé t=[tmin,tmax]
  y(1) = y0 ;                             % initialisation : y(1)=y(t0) = y0
  for n = 2 : Nint+1
     y(n) =y(n-1) + h*feval(f,t(n-1),y(n-1)) ;        % Calcul d'Euler
  end    % for n
 end     % fonction Euler
```

(E)

La fonction 'Euler' retourne le vecteur $t = [t_0, \ldots, t_N]$ et un vecteur $y$ contenant les $y(t_n)$ à chacun des temps $t_n$ de l'intervalle $t \in [t_0, t_0 + T]$.

**A noter :** nous avons utilisé la fonction MatLab (Octave) 'feval'. La fonction 'feval("fcnt",x)' est une commande de Octave (MATLAB) qui évalue la fonction de nom 'fcnt' et d'argument x . **Exemple :** y=feval("sin",x) est équivalente à y=sin(x). L'utilisation de 'feval' nous permet d'écrire une fois pour toutes la méthode d'Euler et de passer le nom de la fonction à évaluer comme un argument de la fonction 'Euler'.

**Exemple numérique :**

Utilisons la même équation teste que dans l'exemple de la méthode de Picard : $\frac{dy}{dt} = t - y(t)$. Essayons d'abord avec $N = 20$ intervalles. En MatLab (Octave) entrez :

```
>>  f = inline('t - y','t','y') ;
>>  [t,yEul20] = Euler( f, 0, 2, 20, 1) ;
```

où la fonction 'inline' est une façon rapide de créer une fonction simple comme $f = t - y$ sans avoir à créer un fichier 'f.m' de cette fonction.

On peut générer les valeurs exactes de la solution de cette équation par les commandes suivantes :

```
>> yt = inline('(y0+1)*exp(-t) + t - 1','t','y0') ;
```

```
>> yexact = yt(t,y0) ;
```

Finalement, on peut comparer graphiquement la solution générée par la méthode d'Euler avec la solution exacte grâce aux commandes :

```
>> plot(t,yexact,t,yEul20,' :') ;
>> legend('y(t)',' Euler ') ;
```

Le vecteur $t$ est donc ici $t = [t_0 = 0, t_1, \ldots, t_N = 2]$ et 'yEul20' est la solution par la méthode d'Euler avec 20 intervalles.

Bien entendu, on peut améliorer la solution d'Euler en diminuant $h$ (i.e. en augmentant le nombre d'intervalles). On voit sur la figure (5) une comparaison des solutions de $y' = t - y$ par la méthode d'Euler pour $N = 20$ et $N = 50$ pour $t \in [0, 2]$.

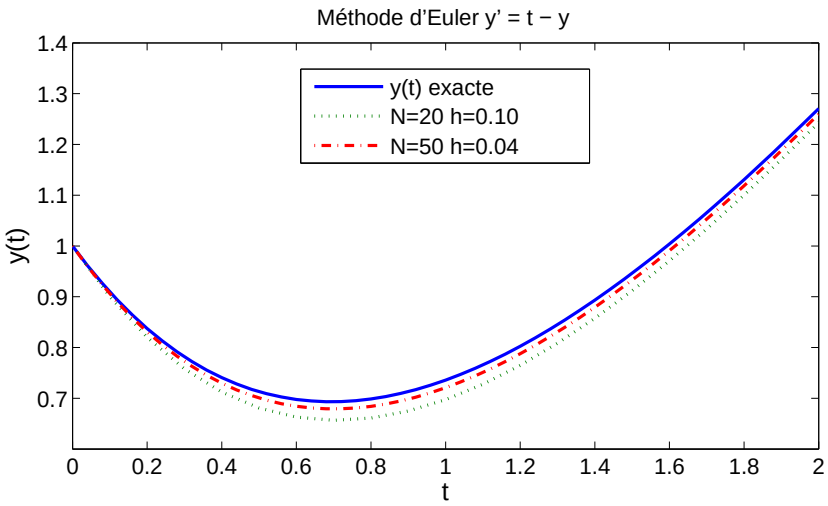


FIG. 5 – Méthode d'Euler. Comparaison des solutions de $\frac{d}{dt}y(t) = t - y(t)$ avec les approximations par méthode d'Euler.

Vous avez peut-être remarqué qu'ici la solution par Euler s'approche de la solution exacte pour les grands $t$. Ce phénomène est une propriété de l'équation différentielle particulièrement simple que nous étudions ici, et non une propriété générale des solutions par méthode d'Euler.

### 3.3.2 Méthodes de Taylor d'ordre plus élevés

En considérant l'équation (45), et en augmentant le nombre de termes inclus dan la série de Taylor on s'attend à une amélioration de la solution numérique. Pour ceci, nous avons besoin des dérivées d'ordre supérieur à 1. On peut les obtenir en dérivant l'équation différentielle de

base :

$$\begin{aligned}
\frac{dy}{dt} &\equiv y^{(1)} = f\left(t, y\left(t\right)\right) \\
\frac{d^2y}{dt^2} &\equiv y^{(2)} = \frac{d}{dt} f\left(t, y\left(t\right)\right) \\
y^{(3)} &= \frac{d^2}{dt^2} f\left(t, y\left(t\right)\right) \\
&\vdots \\
y^{(m)} &= \frac{d^{m-1}}{dt^{m-1}} f\left(t, y\left(t\right)\right)
\end{aligned} \tag{47}$$

Le développement de la série de Taylor (45), nous montre que l'erreur commise par une méthode de Taylor d'ordre $m$ est $O\left(h^{m+1}\right)$. Néanmoins, il est incorrect de penser que plus $m$ est grand mieux c'est, puisque les calculs à chaque étape deviennent plus lourds au fur et à mesure que $m$ augmente (voir (F) ci dessous). Il est communément admis qu'il n'est pas profitable de prendre $m > 4$.

Pour notre équation $y^{(1)} = t - y$ traitée dans les exemples précédents, ce procédé pour déterminer les $y^{(m)}$ avec $m > 1$ est facile :

$$\begin{aligned}
y^{(1)} &= t - y \\
y^{(2)} &= 1 - \frac{d}{dt} y\left(t\right) = 1 - t + y\left(t\right) \\
y^{(3)} &= \frac{d}{dt}\left(1 - t + y\left(t\right)\right) = -1 + t - y \\
y^{(4)} &= \frac{d}{dt}\left(-1 + t - y\left(t\right)\right) = 1 - t + y \\
&\vdots
\end{aligned} \tag{48}$$

La fonction Matlab suivante calcule les approximations à solution de $y^{(1)} = t - y$ faites avec la série de Taylor jusqu'à l'ordre 4.

```
function Taylor(tmin,tmax,Nint,y0)          % Approximations par série de Taylor
% Nint - nombre de sous intervalles N
% tmin - temps t_0   ;    tmax - temps t_0 + T
% On trait ici l'équation y' = t - y(t)
 h = (tmax-tmin)/Nint ;                      % valeur du pas
 t = linspace(tmin,tmax,Nint+1) ;            % vecteur de t discrétisé t=[tmin,tmax]
 yt = inline('(y0+1)*exp(-t) + t - 1','t','y0') ; % solution exact                    (F)
 texa = linspace(tmin,tmax,101) ;      % discrétisation de t pour l'affichage de la solution exacte
  yexact = yt(texa,y0) ;
 y1 = inline('t - y','t','y') ;                % y^(1)
 y2 = inline('1 - t + y','t','y') ;            % y^(2)
 y3 = inline('-1 + t - y','t','y') ;           % y^(3)
 y4 = inline('1 - t + y','t','y') ;            % y^(4)
 yT1(1) = y0 ;          % yT1 contient les solutions de y - Approximation d'Euler
 yT2(1) = y0 ;          % yT2 contient les solutions de y Approximation de Taylor ordre 2
 yT3(1) = y0 ;          % yT3 contient les solutions de y Approximation de Taylor ordre 3
```

```
 yT4(1) = y0;              % yT4 contient les solutions de y Approximation de Taylor ordre 4
  for n = 2 :Nint+1        % Calcul approximation d'Euler
       yT1(n) = yT1(n-1) + h*y1(t(n-1),yT1(n-1));
  end % for n
  for n = 2 :Nint+1          % Calcul de l'approximation de Taylor ordre 2
       yT2(n) = yT2(n-1) + h*y1(t(n-1),yT2(n-1)) + (h^2/2)*y2(t(n-1),yT2(n-1));
  end % for n
 for n = 2 :Nint+1           % Calcul de l'approximation de Taylor ordre 3
      yT3(n) = yT3(n-1) + h*y1(t(n-1),yT3(n-1)) + ...
      +(h^2/2)*y2(t(n-1),yT3(n-1))+(h^ 3/6)*y3(t(n-1),yT3(n-1));
   end % for n
for n = 2 :Nint+1           % Calcul de l'approximation de Taylor ordre 4
  yT4(n) = yT4(n-1) + h*y1(t(n-1),yT4(n-1)) + ...
   +(h^2/2)*y2(t(n-1),yT4(n-1))+(h^ 3/6)*y3(t(n-1),yT4(n-1)) + ...
   + (h^4/24)*y4(t(n-1),yT4(n-1));
end % for n
% plot : créer un graphique avec les résultats des 4 courbes
axes('FontSize',14);
plot(texa,yexact,t,yT1,t,yT2,t,yT3,t,yT4,'+','LineWidth',2)
xlabel(' t ','FontSize',16);
ylabel(' y(t) ','FontSize',16);
legend('y(t) exacte','Euler (ordre 1)','Taylor ordre 2','Taylor ordre 3','Taylor ordre 4');
tit = sprintf('Méthodes de Taylor y"=t-y : N=%d, h=%.2f',Nint,h);
title(tit);
end
```

On peut maintenant trouver la solution de $y^{(1)} = t - y$ avec $t \in [0, 2]$ et $N = 20$ avec la commande :

```
>> Taylor(0,2,20,1);
```

Le résultat apparaît en figure (6). On voit qu'à partir de l'ordre 2 de Taylor, on ne voit plus de différence entre les approximations basées sur la méthode de Taylor et la solution exacte à l'échelle de la figure, alors que la différence entre l'approximation d'Euler et la solution exacte est considérable. On peut essayer de faire apparaître les différences entre les calculs à différents ordres de la série de Taylor en augmentant le pas de $h = 0.1$ à $h = 0.4$ avec la commande :

```
>> Taylor(0,2,5,1);
```

Le résultat de cette commande apparaît sur la figure (7). On constate que même avec ce pas assez grand, les 5 valeurs calculées par la méthode de Taylor à l'ordre 4 reste sur la courbe de la solution exacte.

### 3.4 Runge Kutta

Ayant constaté les améliorations offertes par le développement de Taylor, on doit remarquer un problème important. Pour se servir de la méthode de Taylor, il faut calculer les fonctions $y^{(m)}(t)$ d'ordre $m > 1$ à la main en prenant les dérivées de l'équation de base $y' = f(t, y(t))$. Par conséquent, avec les méthodes de Taylor on doit recommencer cette étape chaque fois qu'on change la fonction $f$. Cette opération est lourde et loin d'être pratique.

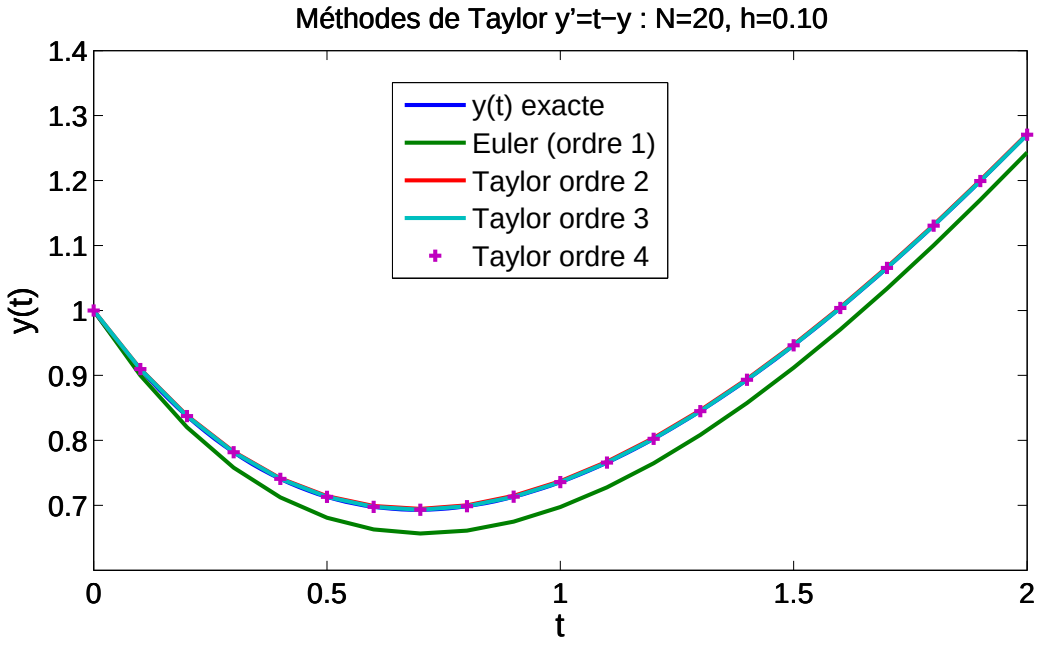


FIG. 6 – Méthodes basées sur la série de Taylor avec $h = 0.1$.

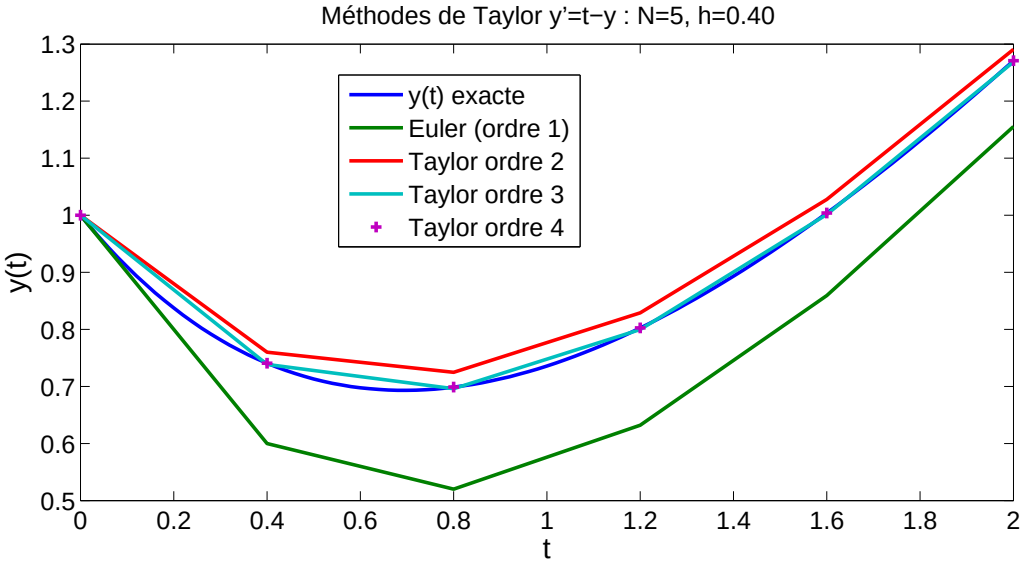


FIG. 7 – Méthodes basées sur la série de Taylor avec $h = 0.4$.

La méthode Runge Kutta tire les avantages des méthodes de Taylor tout en gardant une simplicité d'exécution de la méthode d'Euler. En pratique, Runge Kutta remplace l'évaluation analytique des ordres $y^{(m)}$, $m > 1$ par des dérivées numériques obtenues en évaluant la fonction $f(t, y(t))$ à différents endroits afin d'obtenir presque les mêmes résultats que ceux obtenu avec la méthode de Taylor.

### 3.4.1 Runge Kutta d'ordre 2

Rappelons la méthode d'Euler

$$y(t_{n+1}) = y(t_n) + hf(t_n, y(t_n)) \tag{49}$$

La méthode d'Euler permet ainsi de calculer $y(t_n + h)$ en fonction de $y(t_n)$ et la dérivée en $y(t_n)$. Cette méthode n'est pas symétrique par rapport à l'intervalle puisqu'il ne fait pas intervenir l'information sur la dérivée en fin d'intervalle, i.e. $f(t_n, y(t_{n+1}))$ n'intervient pas.

La méthode Runge Kutta est plus symétrique puisqu'elle revient à évaluer numériquement la dérivée au centre de l'intervalle. Les équations de cette méthode sont :

$$\begin{aligned} k_1 &\equiv hf(t_n, y(t_n)) \\ k_2 &\equiv hf\left(t_n + \frac{1}{2}h, y(t_n) + \frac{1}{2}k_1\right) \\ y(t_{n+1}) &= y(t_n) + k_2 + O\left(h^3\right) \end{aligned} \tag{50}$$

Une façon de programmer la méthode Runge Kutta d'ordre 2 est la suivante :

```
function [t,y] = RK2(f,tmin,tmax,Nint,y0)   % Méthode de Runge Kutta d'ordre 2
% Nint - nombre de sous intervalles
% tmin - temps t0
% tmax - temps t0 + T
% f est une fonction avec comme arguments t et y : f(t,y(t))
% y0 contient les valeurs des conditions limites
 h = (tmax-tmin)/Nint ;                % valeur du pas                          (G)
 t = linspace(tmin,tmax,Nint+1) ;      % vecteur de t discrétisé t=[tmin,tmax]
 y(1) = y0 ;                           % y contient les solutions de y(t_n)n = 1, ..., Nint + 1
 for n = 2 :Nint+1
    k1 = h*feval(f,t(n-1),y(n-1)) ;
    k2 = h*feval(f,t(n-1)+h/2,y(n-1)+k1/2) ;
    y(n) = y(n-1) + k2 ;
 end % for n
end
```

Comme la méthode d'Euler, les méthodes de Runge Kutta peuvent être appliquées à une fonction arbitraire. La quasi-équivalence de la méthode Runge Kutta d'ordre 2 avec la méthode de Taylor d'ordre 2 n'est pas évidente sans quelques manipulations mathématiques (voir ci-dessous). Néanmoins, on peut rapidement vérifier que leur comportement au niveau numérique est similaire. Pour notre exemple habituel, $y' = t - y$, nous comparons en figure (8) les résultats des deux méthodes en utilisant un grand pas, $h = 0.4$.

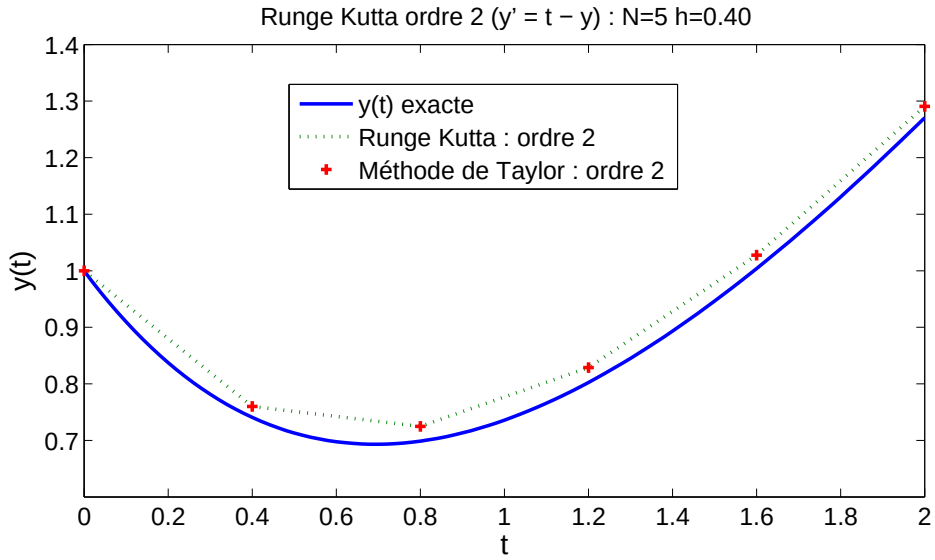


FIG. 8 – Comparaison de la méthode de Runge Kutta d'ordre 2 avec la série de Taylor d'ordre 2 avec $h = 0.4$.

**Dérivation de Runge Kutta d'ordre 2** Avec un peu d'effort, on peut dériver les formules de Runge Kutta à partir des formules de Taylor et les formules des dérivées par différences finies. Prenons un développement en série de Taylor autour du point $(\widetilde{t},\widetilde{y})$ au centre de l'intervalle $t_n, t_{n+1}$.

$$\widetilde{t} \equiv t_n + \frac{h}{2} \qquad \widetilde{y} \equiv y\left(t_n + \frac{h}{2}\right) \tag{51}$$

La série de Taylor développée autour de ce point s'écrit en remplaçant $t_n \to t_n + \frac{h}{2}$, et $h \to \frac{h}{2}$ dans le développement de Taylor de l'éq.(45) :

$$\begin{aligned} y\left(t_n + h\right) &= y\left(t_n + \frac{h}{2}\right) + \frac{h}{2} y^{(1)}\left(t_n + \frac{h}{2}\right) + \frac{h^2}{8} y^{(2)}\left(t_n + \frac{h}{2}\right) + O(h^3) \\ &= \widetilde{y} + \frac{h}{2} f\left(t_n + \frac{h}{2}, \widetilde{y}\right) + \frac{h^2}{8} y^{(2)}\left(t_n + \frac{h}{2}\right) + O(h^3) \end{aligned} \tag{52}$$

Maintenant, on remplace $y^{(2)}\left(t_n + \frac{h}{2}\right)$ par sa dérivée par différences discrètes :

$$\begin{aligned} y^{(2)}\left(t_n + \frac{h}{2}\right) &= \frac{4}{h^2}\left[y\left(t_n + h\right) - 2y\left(t_n + \frac{h}{2}\right) + y\left(t_n\right)\right] + O\left(h\right) \\ &= \frac{4}{h^2}\left[y\left(t_n + h\right) - 2\widetilde{y} + y\left(t_n\right)\right] + O\left(h\right) \end{aligned} \tag{53}$$

Insérant ce résultat dans l'équation (52), on obtient :

$$\begin{aligned} y\left(t_n + h\right) &= \widetilde{y} + \frac{h}{2} f\left(t_n + \frac{h}{2}, \widetilde{y}\right) + \frac{1}{2}\left[y\left(t_n + h\right) - 2\widetilde{y} + y\left(t_n\right)\right] + O(h^3) \\ &= \frac{h}{2} f\left(t_n + \frac{h}{2}, \widetilde{y}\right) + \frac{1}{2} y\left(t_n + h\right) + \frac{1}{2} y\left(t_n\right) + O(h^3) \end{aligned} \tag{54}$$

ce qui peut s'écrire :

$$y\left(t_n+h\right)=y\left(t_n\right)+hf\left(t_n+\frac{h}{2},\widetilde{y}\right)+O(h^3) \tag{55}$$

mais la série de Taylor d'ordre un nous donne le résultat suivant :

$$\begin{aligned}\widetilde{y}\equiv y\left(t_n+\frac{h}{2}\right)&=y\left(t_n\right)+\frac{h}{2}f\left(t_n,y\left(t_n\right)\right)+O(h^2)\\&\equiv y\left(t_n\right)+\frac{h}{2}k_1+O(h^2)\end{aligned} \tag{56}$$

où nous avons utilisé la définition de $k_1$ (voir l'éq.(50) ). Avec ce résultat, on peut faire l'approximation :

$$\begin{aligned}hf\left(t_n+\frac{h}{2},\widetilde{y}\right)&=hf\left(t_n+\frac{h}{2},y\left(t_n\right)+\frac{h}{2}k_1\right)+O(h^3)\\&\equiv hk_2+O(h^3)\end{aligned} \tag{57}$$

et l'équation (55) devient finalement la formule de Runge Kutta d'ordre 2 :

$$y\left(t_n+h\right)=y\left(t_n\right)+hk_2+O(h^3). \tag{58}$$

### 3.4.2 Runge Kutta : ordres 3 et 4

Les dérivations de Runge Kutta aux ordres 3 et 4 sont fastidieuses. Heureusement leurs formules sont faciles à programmer et nous nous contentons d'utiliser les résultats.

La formule Runge-Kutta à l'ordre 3 est :

$$\begin{aligned}k_1&=hf\left(t_n,y\left(t_n\right)\right)\\k_2&=hf\left(t_n+\frac{1}{2}h\ ,\ y\left(t_n\right)+\frac{1}{2}k_1\right)\\k_3&=hf\left(t_n+h\ ,\ y\left(t_n\right)+2k_2-k_1\right)\\y\left(t_{n+1}\right)&=y\left(t_n\right)+\frac{1}{6}\left(k_1+4k_2+k_3\right)+O\left(h^4\right)\end{aligned} \tag{59}$$

La formule Runge-Kutta à l'ordre 4 est de loin la plus utilisée. Elle a une forme assez symétrique :

$$\begin{aligned}k_1&=hf\left(t_n,y\left(t_n\right)\right)\\k_2&=hf\left(t_n+\frac{1}{2}h\ ,\ y\left(t_n\right)+\frac{1}{2}k_1\right)\\k_3&=hf\left(t_n+\frac{1}{2}h\ ,\ y\left(t_n\right)+\frac{1}{2}k_2\right)\\k_4&=hf\left(t_n+h\ ,\ y\left(t_n\right)+k_3\right)\\y\left(t_{n+1}\right)&=y\left(t_n\right)+\frac{1}{6}\left(k_1+2k_2+2k_3+k_4\right)+O\left(h^5\right)\end{aligned} \tag{60}$$

La technique Runge-Kutta à l'ordre 4 intervient dans la plupart des programmes ODE (Ordinary Differential Equations) comme ceux utilisés par Matlab et Octave.

On montre maintenant en figure (9) qu'en pratique Runge-Kutta à l'ordre 4 donne presque les mêmes résultats que la technique de Taylor d'ordre 4.

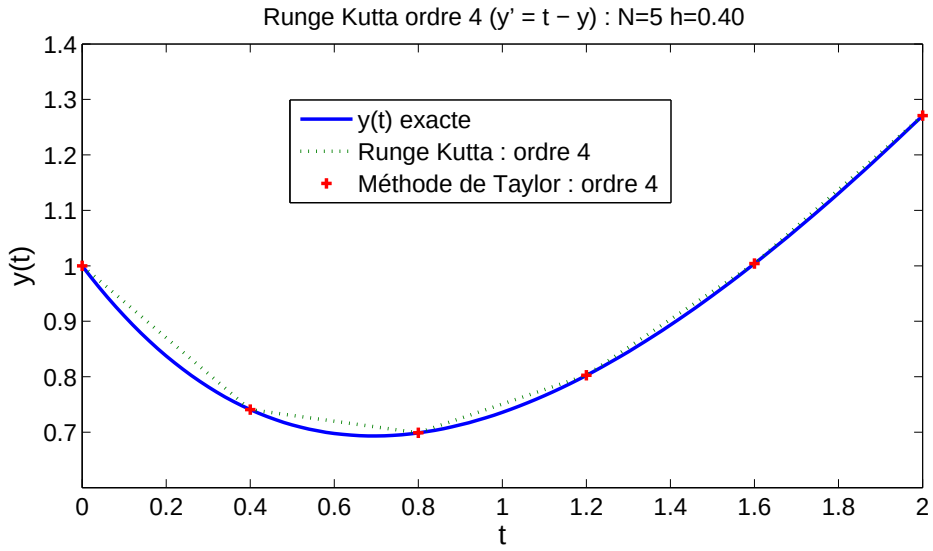


FIG. 9 – Comparaison de la méthode Runge Kutta d'ordre 4 avec la série de Taylor ordre 4 avec $h = 0.4$.

On remarque que Runge-Kutta d'ordre 4 fait deux fois plus d'évaluations de la fonction $f$ que Runge-Kutta d'ordre 2. Par conséquent RK4 n'est vraiment meilleur que RK2 que si RK4 donne de meilleurs résultats que RK2 avec un pas deux fois plus grand. Ceci est souvent le cas, mais pas toujours.

### 3.4.3 Runge Kutta à pas adaptatif et méthodes prédiction correction

Même si l'erreur produite par la méthode Runge Kutta est de l'ordre de $O\left(h^5\right)$, il reste la question du choix de $h$ afin d'avoir le meilleur compromis entre précision et temps de calcul. Il est intuitivement clair que l'on peut choisir un $h$ relativement grand dans une région où $y\left(t\right)$ varie plutôt lentement, mais un petit $h$ est nécessaire quand il y a de fortes variations de $y\left(t\right)$. Matlab utilise une méthode Runge Kutta d'ordre 4 à pas adaptatif dans ODE45 pour résoudre les équations différentielles. Dans les situations où une grande précision est nécessaire, des méthode dites prédiction correction sont souvent plus efficace que même Runge Kutta d'ordre 4. Néanmoins, les méthodes prédiction correction ont besoin d'une première solution approximative pour démarrer leurs algorithmes. Souvent donc, Runge Kutta est invoqué par les algorithmes prédiction correction pour fournir une première solution approximative.

## 3.5 Fonctions Euler et Runge Kutta adaptée à $y \in \mathbb{R}^m$

On peut facilement écrire les fonctions d'Euler et de Runge-Kutta afin qu'elles marchent pour un nombre de fonctions inconnues, $y_i\left(t\right)$, arbitraires. Pour faciliter les comparaisons, on écrira ces fonctions afin qu'elles retournent des valeurs dans le même format que 'ODE45' (de MatLab), ou ('lsode' d'Octave) c'est-à-dire dans une matrice composée de $N+1$ lignes et une colonne pour chaque fonction inconnue $y_i\left(t\right)$, $i \in [1, m]$.

```
function [t,y] = Euler(f,tmin,tmax,Nint,y0)   % Méthode d'Euler
% Nint - nombre de sous intervalles
% tmin - temps t_0
```

```
% tmax - temps t0 + T
% f est une fonction avec comme arguments t et un vecteur y : f(t,y(t))
% y0 contient les valeurs des conditions initiales
 h = (tmax-tmin)/Nint;                                                    (H)
 t =linspace(tmin,tmax,Nint+1);          % vecteur de t discrétisé t=[tmin,tmax]
 t = t';
 nc = length(y0);
 y = zeros(Nint+1,nc);     % initialisation d'une matrice de solution
 y(1, :) = y0;       % la ligne y(1, :) contient les conditions initiales y0
 for n = 2 :Nint+1
       y(n, :) = y(n-1, :) + h*feval(f,t(n-1),y(n-1, :))'; % Calcul d'Euler
 end    % for n
end
```

On peut facilement modifier les fonctions Runge Kutta de la même manière afin d'obtenir des fonctions qui acceptent $y_0$ comme un vecteur et donnent les solutions pour $y_i(t)$, $i \in [1, m]$.

# 4 Applications

## 4.1 Mécanique des points solides

L'Equation fondamentale de la dynamique $\sum \overrightarrow{\boldsymbol{F}} = m\overrightarrow{\boldsymbol{a}}$, nous donne typiquement des équations différentielles de deuxième ordre.

### 4.1.1 Mouvement d'un point solide avec forces de frottement :

Soit une particule de masse $m$, de poids $\overrightarrow{\boldsymbol{P}} = m\overrightarrow{\boldsymbol{g}}$ qui subit le frottement $\overrightarrow{\boldsymbol{F}}_f = -\gamma\overrightarrow{\boldsymbol{v}}$ dans le milieu (figure 10). Son équation de mouvement s'écrit :

$$m\frac{d^2\overrightarrow{\boldsymbol{x}}}{dt^2} = m\overrightarrow{\boldsymbol{g}} - \gamma\frac{d\overrightarrow{\boldsymbol{x}}}{dt} \tag{61}$$

ce qui se réduit au système d'équations découplées :

$$\begin{cases} \frac{d^2x}{dt^2} - \frac{\gamma}{m}\frac{dx}{dt} = 0 \\ \frac{d^2z}{dt^2} - \frac{\gamma}{m}\frac{dz}{dt} = -g \end{cases} \tag{62}$$

Les mêmes substitutions que celles proposées en équations (19) et (21) nous permettent de mettre les équations (62) sous la forme d'une équation matricielle d'ordre un. Même si on peut résoudre ce problème analytiquement, la solution pour $\gamma \neq 0$ est un peu pénible et nécessite l'utilisation de nombres complexes. Par contre, l'extension de la solution numérique au cas de $\gamma \neq 0$ est triviale et nécessite seulement une petite modification de la fonction 'fprojectile' trouvée en (A) de section 2.3).

### 4.1.2 Orbite d'un satellite :

Un satellite subit une force gravitationnelle dirigée vers le centre de la terre. Son mouvement s'effectue donc dans le plan qui contient le vecteur de vitesse initiale $\overrightarrow{\boldsymbol{v}_0}$ et le centre de la terre.

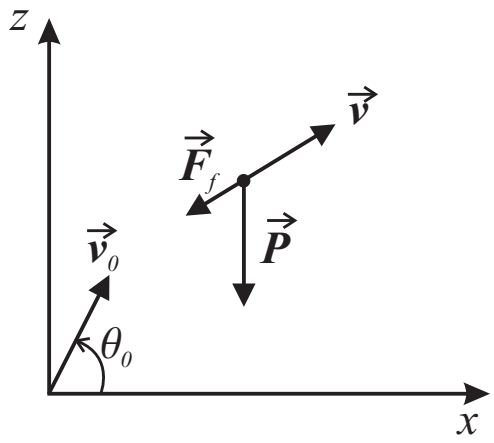


FIG. 10 – Mouvement d'un point de masse qui subit la force de pesanteur et la force de frottement dans le milieu.

Donc on peut représenter son mouvement par deux variables dans ce plan, $x$ et $y$ avec le centre de la terre comme origine.

Le mouvement du satellite est gouverné par la force gravitationnelle :

$$\vec{\boldsymbol{F}} = m\vec{\boldsymbol{a}} = -\frac{GMm}{r^2}\widehat{\boldsymbol{r}} = -GMm\frac{\vec{\boldsymbol{r}}}{r^3} = -GMm\frac{x\widehat{\boldsymbol{x}} + y\widehat{\boldsymbol{y}}}{(x^2+y^2)^{3/2}} \tag{63}$$

où $G = 6,67\,10^{-11}\mathrm{m}^3\mathrm{kg}^{-1}\mathrm{s}^{-2}$ est la constante gravitationnelle et $M \simeq 5,976\,10^{24}\mathrm{kg}$ la masse de la terre. L'accélération du satellite s'écrit donc :

$$\boldsymbol{a} = -C\frac{\boldsymbol{r}}{r^3} = -C\frac{x\widehat{\boldsymbol{x}} + y\widehat{\boldsymbol{y}}}{(x^2+y^2)^{3/2}} \tag{64}$$

où $C \equiv GM = 3,986\,10^{14}\mathrm{m}^3\mathrm{s}^{-2} = 3,986\,10^{5}\mathrm{km}^3\mathrm{s}^{-2}$ est une constante. On remarque que les équations différentielles en $x$ et $y$ sont couplées :

$$\begin{aligned}\frac{d^2x}{dt^2} &= -C\frac{x}{(x^2+y^2)^{3/2}}\\ \frac{d^2y}{dt^2} &= -C\frac{y}{(x^2+y^2)^{3/2}}\end{aligned} \tag{65}$$

Bien qu'on puisse résoudre les équations, (65), avec Matlab (Octave), l'analyse est plus efficace si on utilise des coordonnées circulaires $r$ et $\theta$ d'un point $M$ dont les relations avec les coordonnées $x$ et $y$ de $M$ sont données par (voir figure 11) :

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} r\cos\theta \\ r\sin\theta \end{bmatrix} \tag{66}$$

On rappelle qu'en coordonnées circulaires on définit de nouveaux vecteurs unitaires $\widehat{\boldsymbol{r}}$ et $\widehat{\boldsymbol{\theta}}$ (voir figure 11 ) :

$$\widehat{\boldsymbol{r}} = \begin{bmatrix} \cos\theta \\ \sin\theta \end{bmatrix} \qquad \widehat{\boldsymbol{\theta}} = \begin{bmatrix} -\sin\theta \\ \cos\theta \end{bmatrix} \tag{67}$$

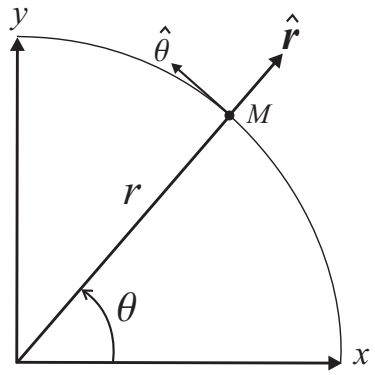


FIG. 11 – Coordonnées circulaires.

dont les orientations dépendent sur l'angle $\theta$. Par conséquent, quand on effectue des dérivées temporelles de la position d'un point solide en coordonnées circulaires, il faut tenir compte du fait que les orientations des vecteurs $\widehat{\boldsymbol{r}}$ et $\widehat{\boldsymbol{\theta}}$ changent elles aussi avec le temps.

Les dérivées temporelles de $\widehat{\boldsymbol{r}}$ et $\widehat{\boldsymbol{\theta}}$ sont :

$$\frac{d\widehat{\boldsymbol{r}}}{dt} = \begin{bmatrix} -\overset{\bullet}{\theta}\sin\theta \\ \overset{\bullet}{\theta}\cos\theta \end{bmatrix} = \overset{\bullet}{\theta}(t)\ \widehat{\boldsymbol{\theta}} \qquad \frac{d\widehat{\boldsymbol{\theta}}}{dt} = -\begin{bmatrix} \overset{\bullet}{\theta}\cos\theta \\ \overset{\bullet}{\theta}\sin\theta \end{bmatrix} = -\overset{\bullet}{\theta}(t)\ \widehat{\boldsymbol{r}} \tag{68}$$

En coordonnées circulaires, la vitesse $\overrightarrow{\boldsymbol{v}}$ du point s'exprime donc :

$$\begin{aligned} \overrightarrow{\boldsymbol{v}} \equiv \frac{d\overrightarrow{\boldsymbol{r}}}{dt} &= \widehat{\boldsymbol{r}}\frac{dr}{dt} + r\frac{d\widehat{\boldsymbol{r}}}{dt} = \overset{\bullet}{r}\widehat{\boldsymbol{r}} + r\overset{\bullet}{\theta}\widehat{\boldsymbol{\theta}} \\ &= v_r\widehat{\boldsymbol{r}} + v_t\widehat{\boldsymbol{\theta}} \end{aligned} \tag{69}$$

où $v_r \equiv \overset{\bullet}{r}$ est la *vitesse radiale* et $v_t \equiv r\overset{\bullet}{\theta}$ la *vitesse tangentielle*. De la même manière, l'accélération du satellite est donnée par :

$$\begin{aligned} \overrightarrow{\boldsymbol{a}} \equiv \frac{d\overrightarrow{\boldsymbol{v}}}{dt} &= \frac{d}{dt}\left(\overset{\bullet}{r}\widehat{\boldsymbol{r}}\right) + \frac{d}{dt}\left(r\overset{\bullet}{\theta}\widehat{\boldsymbol{\theta}}\right) \\ &= \overset{\bullet\bullet}{r}\widehat{\boldsymbol{r}} + 2\overset{\bullet}{r}\overset{\bullet}{\theta}\widehat{\boldsymbol{\theta}} + r\overset{\bullet\bullet}{\theta}\widehat{\boldsymbol{\theta}} - r\overset{\bullet}{\theta}^2\widehat{\boldsymbol{r}} \end{aligned} \tag{70}$$

On rappelle que la force gravitationnelle agit seulement dans la direction radiale, $\overrightarrow{\boldsymbol{F}} = -\frac{mC}{r^2}\widehat{\boldsymbol{r}} \equiv F_r\widehat{\boldsymbol{r}} + F_\theta\widehat{\boldsymbol{\theta}}$. Prises ensemble avec l'expression de $\overrightarrow{\boldsymbol{a}}$ de l'éq.(70), les équations du mouvement, $\overrightarrow{\boldsymbol{F}} = m\overrightarrow{\boldsymbol{a}}$, du satellite sont :

$$\begin{aligned} \overset{\bullet\bullet}{r} - r\overset{\bullet}{\theta}^2 &= \frac{F_r}{m} = -\frac{C}{r^2} \\ \overset{\bullet\bullet}{\theta} + 2\frac{\overset{\bullet}{r}}{r}\overset{\bullet}{\theta} &= \frac{F_\theta}{m} = 0 \end{aligned} \tag{71}$$

On transforme ce système d'équations différentielles couplées en système de Cauchy par les définitions habituelles :

$$\begin{aligned} u_1(t) &\equiv r(t) \qquad & u_2(t) &\equiv \overset{\bullet}{r}(t) \\ u_3(t) &\equiv \theta(t) \qquad & u_4(t) &\equiv \overset{\bullet}{\theta}(t) \end{aligned} \tag{72}$$

Le système d'équations différentielles de première ordre s'écrit alors :

$$\begin{aligned}
\frac{d}{dt}u_1 &= u_2 \\
\frac{d}{dt}u_2 &= -\frac{C}{u_1^2} + u_1 u_4^2 \\
\frac{d}{dt}u_3 &= u_4 \\
\frac{d}{dt}u_4 &= -2\frac{\dot{r}}{r}\dot{\theta} = -2\frac{u_2 u_4}{u_1}
\end{aligned} \tag{73}$$

i.e. $\frac{d}{dt}u(t) = f(t, u(t))$ avec

$$f(t, u(t)) = \begin{bmatrix} u_2 \\ -\frac{C}{u_1^2} + u_1 u_4^2 \\ u_4 \\ -2\frac{u_2 u_4}{u_1} \end{bmatrix} \tag{74}$$

On remarque que $\frac{d}{dt}u(t) = f(t, u(t))$ ne prend pas la forme d'une équation matricielle, mais on peut toujours résoudre ce système avec 'ODE45', 'lsode', Méthode d'Euler, Runge Kutta etc. Une fois qu'on a trouvé la solution des $u(t) = (u_1, ..., u_4)$, on peut tracer l'orbite du satellite en rappelant que

$$\begin{aligned}
x(t) &= r\cos\theta = u_1(t)\cos u_3(t) \\
y(t) &= r\cos\theta = u_1(t)\sin u_3(t)
\end{aligned} \tag{75}$$

et en faisant plot(x,y,'-o').

Vous pouvez imaginer que de telles méthodes sont employées de manière intensive par des ingénieurs de l'ESA. Imaginez un instant les calculs nécessaires pour envoyer une sonde dans le système solaire en tenant compte de la poussé des moteurs, des effets gravitationnels du soleil et des planètes, tout en cherchant à minimiser la consommation de carburant.

## 4.2 Circuits électriques

On rappelle la relation entre la charge $Q$ sur un condensateur et le potentiel $v$ à ces bornes :

$$Q = vC \tag{76}$$

On rappelle qu'en convention récepteur le potentiel aux bornes d'une bobine est $u_L = L\frac{di}{dt}$ et aux bornes d'une résistance $u_R = Ri$. La loi des mailles pour ce circuit RLC simple est :

$$L\frac{di}{dt} + Ri + v - u = 0 \tag{77}$$

Utilisant la définition du courant :

$$i = \frac{dQ}{dt} = C\frac{dv}{dt} \tag{78}$$

on obtient une équation différentielle de deuxième ordre pour la tension aux bornes du condensateur :

$$LC\frac{d^2v}{dt^2} + RC\frac{dv}{dt} + v = u \tag{79}$$

La même équation différentielle peut gouverner de bien différentes situations physiques. Dans un premier temps imaginer un circuit avec un interrupteur ouvert, sans générateur de tension (i.e. $u = 0$), et un condensateur chargé au potentiel $v_0 = 10\text{V}$. On ferme le circuit au temps $t_0 = 0$. L'équation de mouvement est donc homogène :

$$LC\frac{d^2v}{dt^2} + RC\frac{dv}{dt} + v = 0 \tag{80}$$

On fait les définitions habituelles :

$$\begin{aligned} y_1(t) &= v(t) \\ y_2(t) &= \frac{dv(t)}{dt} \end{aligned} \tag{81}$$

afin d'obtenir des équations d'ordre 1 :

$$\begin{aligned} \frac{d}{dt}y_1(t) &= y_2(t) \\ \frac{d}{dt}y_2(t) &= -\frac{1}{LC}y_1(t) - \frac{R}{L}y_2(t) \end{aligned} \tag{82}$$

La première condition initiale est simplement $y_1(t_0) = v(t_0) = v_0$. Pour la seconde condition initiale, il faut se rappeler que le courant à travers une bobine doit être une fonction continue du temps. Puisque dans ce problème $i(t_0 < 0) = 0$, la continuité du courant impose la condition initiale $i(t_0) = 0$, donc $y_2(t_0) = 0$.

Maintenant, on peut trouver la solution avec une fonction $f(t, y(t))$ appropriée et 'lsode' ('ODE45') et $y_0 = y(t_0) = (v_0, 0)$.

### 4.3 Evolution temporelle des populations

Les biologistes ont remarqué que si une population est suffisamment grande, la croissance de la population est proportionnelle à la population. Ceci revient à l'observation simple que la croissance de la population est proportionnelle au nombre de couples possible dans la population. Ces observations se traduisent par une équation différentielle simple :

$$\frac{dp}{dt} = ap(t) \tag{83}$$

qui a une solution analytique de

$$p(t) = \exp(at) + Cte \tag{84}$$

Bien entendu, on sait que ce modèle est trop simpliste puisque qu'au fur et à mesure que la population grandit ses membres vont entrer en compétition les uns avec les autres pour des ressources limités. Un meilleur modèle qui tient compte de cette compétition est donc la suivante :

$$\frac{dp}{dt} = ap(t) - bp^2(t) \tag{85}$$

La population humaine sur terre semble avoir suivit une évolution semblable à celle prédite par ce modèle pendant une bonne partie du 20ème siècle. Les biologistes estiment que la valeur de $a$ vaut $0,029$. On sait que la population mondiale en $t_0 = 1965$ valait $p_0 \equiv p(t_0) = 3,34\,10^9$

et qu'elle augmentait de 2% par an à cette époque, i.e. $\frac{1}{p}\frac{dp}{dt}\Big|_{t=t_0} = 0,02$. On peut donc déduire la valeur de $b$ de l'équation différentielle et des valeurs de $p_0$ et de $\frac{1}{p}\frac{dp}{dt}\Big|_{t=t_0}$ :

$$\frac{1}{p}\frac{dp}{dt}\Big|_{t_0} = a - bp\left(t_0\right) \Rightarrow 0,02 = 0,029 - b \times 3,34\,10^9 \tag{86}$$

soit

$$b = 2,695\,10^{-12} \tag{87}$$

Utiliser 'lsode' ('ODE45') afin de modéliser la population mondiale avec cette équation. Quelle est de la population mondiale prévue par cette équation pour 2007 ? Est-ce que cette équation prévoit une limite à la population mondiale ? et si c'est le cas, quelle limite prévoit-elle ?

**Note :** On peut comparer nos résultats avec la solution analytique de l'éq.(85) :

$$p\left(t\right) = \frac{ap_0}{bp_0 + \left(a - bp_0\right)\exp\left(-a\left(t - t_0\right)\right)} \tag{88}$$

On voit ici que la valeur limite de la population mondiale prévu par le modèle est $p\left(t \to \infty\right) = \frac{a}{b}n$.