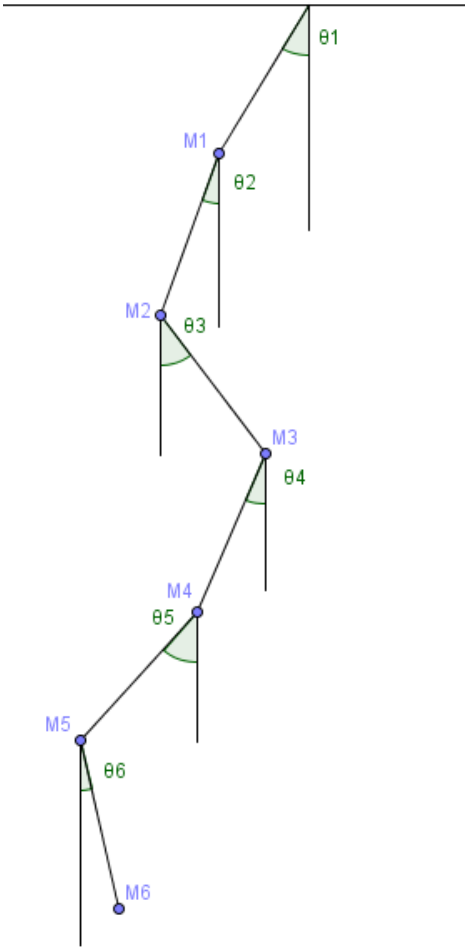
Pendules multiples : méthodes mises en œuvre pour la simulation du mouvement d'un pendule multiple

Problématique : Nous cherchons à simuler le mouvement d'un pendule multiple par informatique.

Cadre : Le système considéré est constitué de n pendules à la suite dans un champ de pesanteur uniforme. Chaque pendule est assimilé à une masse ponctuelle m et la tige qui relie deux pendules successifs est rigide de longueur l, de masse négligeable devant m. On limite notre étude à un mouvement plan. On repère la position du système par les angles θ_k des tiges par rapport à $\vec{e_z}$ unitaire vertical dirigé vers le bas. (Le sens positif est le sens trigonométrique.)



I- Les équations du mouvement

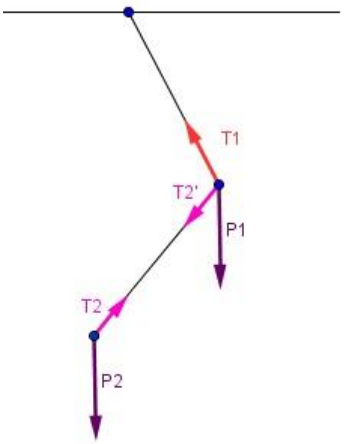
1) Bilan des forces

On s'intéresse au système $\{M_1(m), M_2(m), ..., M_n(m)\}$ dans le référentiel du laboratoire supposé galiléen.

Bilan des forces :

Poids de chaque masse ponctuelle noté $\overrightarrow{P_k}$.

Tensions exercées par les tiges rigides sur les masses ponctuelles notées $\overrightarrow{T_k}$.



On ne connaît pas l'expression de ces tensions. Les utiliser pour une mise en équations reviendrait à introduire de nouvelles inconnues. En outre, ces forces matérialisent la présence d'une contrainte qui lie les points du système : la distance entre M_k et M_{k+1} est toujours l. De telles forces sont appelées « *forces de liaison* ». Selon le *principe de d'Alembert*, elles ne travaillent pas.

On va donc utiliser des considérations énergétiques pour établir les équations du mouvement.

2) Équations d'Euler-Lagrange

Ici, les θ_k sont n variables indépendantes qui décrivent entièrement l'état du système. On les appelle *coordonnées généralisées du système*. Et, les poids dérivent d'une énergie potentielle $\mathrm{Ep}(\theta_1, \theta_2, \ldots, \theta_n)$. On note $\mathrm{Ec}\left(\theta_1, \theta_2, \ldots, \theta_n, \dot{\theta}_1, \dot{\theta}_2, \ldots, \dot{\theta}_n\right)$ l'énergie cinétique du système. Soit L=Ec-Ep le *lagrangien* du système. On montre alors que le Principe Fondamental de la Dynamique implique $\boxed{\forall i, \frac{d}{dt}\frac{\partial \mathrm{L}}{\partial \dot{\theta}_i} - \frac{\partial \mathrm{L}}{\partial \theta_i} = 0}$. Ces équations sont appelées *Équations d'Euler-Lagrange*.

3) Équations du mouvement d'un pendule multiple

On établit les équations du mouvement par récurrence sur le nombre de pendules n considérés. Pour cela, on note Ep_n, Ec_n et L_n respectivement l'énergie potentielle, l'énergie cinétique et le langrangien du système composé de n pendules. On note $ep_k = -mgl\cos\theta_k$ et $ec_k = \frac{1}{2}ml^2\dot{\theta}_k^{\,2}$. On choisit l'énergie potentielle nulle à l'altitude $z=0$ de O.

Pendule simple :

$\mathrm{Ep}_1 = -mgl\cos\theta_1$; $\mathrm{Ec}_1 = \frac{1}{2}ml^2\dot{\theta}_1^{\,2}$ donc $\mathrm{L}_1 = \frac{1}{2}ml^2\dot{\theta}_1^{\,2} + mgl\cos\theta_1$. L'équation d'Euler-Lagrange s'écrit donc : $\frac{d}{dt}\left(ml^2\dot{\theta}_1\right) + mgl\sin\theta_1 = 0$ c'est-à-dire : $\boxed{\ddot{\theta}_1 + \frac{g}{l}\sin\theta_1 = 0}$.

Pendule double :

L'énergie potentielle (respectivement cinétique) du système s'écrit comme la somme des énergies potentielles (respectivement cinétiques) de M_1 et de M_2. On remarque que l'énergie potentielle (respectivement cinétique) de M_1 est Ep_1 (respectivement Ec_1).

L'énergie cinétique de M_2 est $\frac{1}{2}mv_2^{\,2} = \frac{1}{2}ml^2\left(\dot{\theta}_1\overrightarrow{e_{\theta_1}} + \dot{\theta}_2\overrightarrow{e_{\theta_2}}\right)^2$.

Donc. $\mathrm{Ec}_2 = \mathrm{Ec}_1 + ec_1 + ec_2 + ml^2\dot{\theta}_1\dot{\theta}_2\cos(\theta_2-\theta_1) = 2ec_1 + ec_2 + ml^2\dot{\theta}_1\dot{\theta}_2\cos(\theta_2-\theta_1)$.

L'énergie potentielle de M_2 est $-mgl(\cos\theta_1 + \cos\theta_2)$.

Donc $\mathrm{Ep}_2 = \mathrm{Ep}_1 + ep_1 + ep_2 = 2ep_1 + ep_2$.

Le lagrangien s'écrit donc $\mathrm{L}_2 = 2(ec_1 - ep_1) + (ec_2 - ep_2) + ml^2\dot{\theta}_1\dot{\theta}_2\cos(\theta_2-\theta_1)$.

Après simplifications dans les équations d'Euler-Lagrange, on a :

$$\boxed{\begin{cases} 2\left(\ddot{\theta}_1 + \frac{g}{l}\sin\theta_1\right) + \ddot{\theta}_2\cos(\theta_2-\theta_1) - \dot{\theta}_2^{\,2}\sin(\theta_2-\theta_1) = 0 \\ \ddot{\theta}_2 + \frac{g}{l}\sin\theta_2 + \ddot{\theta}_1\cos(\theta_2-\theta_1) + \dot{\theta}_1^{\,2}\sin(\theta_2-\theta_1) = 0 \end{cases}}$$

Système de n pendules :

Par récurrence, on obtient :

$$\boxed{\begin{aligned} \forall k \in \{1,2,\ldots,n\}, (n+1-k)&\left(\ddot{\theta}_k + \frac{g}{l}\sin\theta_k\right) \\ &+ (n+1-k)\sum_{i=1}^{k-1}\ddot{\theta}_i\cos(\theta_k-\theta_i) + \dot{\theta}_i^{\,2}\sin(\theta_k-\theta_i) \\ &+ \sum_{j=k+1}^{n}(n+1-j)\left(\ddot{\theta}_j\cos(\theta_j-\theta_k) - \dot{\theta}_j^{\,2}\sin(\theta_j-\theta_k)\right) = 0 \end{aligned}}$$

Il s'agit d'un système de n équations différentielles non linéaires que l'on ne sait pas résoudre exactement, on va donc procéder à une résolution approchée.

II- Simulation

1) Deux méthodes de résolution numérique approchée d'équations différentielles

On raisonne à partir d'une équation différentielle du type $Y' = f(t,Y)$ où Y est un vecteur d'un ouvert U d'un espace euclidien et f est continue, localement k-lipschitzienne, ce qui assure d'après le théorème de Cauchy-Lipschitz, l'existence et l'unicité d'une solution maximale au *problème de Cauchy* suivant : $\begin{cases} Y' = f(t,Y) \\ Y(t=0) = Y_0 \end{cases}$ $(*)$.

Les deux méthodes présentées sont à pas constant, c'est-à-dire que l'on va calculer des valeurs approchées de $Y(hi)$ pour i variant entre 0 et n_{it} le nombre d'itérations souhaité, et où h est le pas. On notera $Y(hi)$ la valeur exacte de la solution de $(*)$ à l'instant hi et y_i la valeur calculée. On a alors une formule de calcul de la forme suivante : $y_{i+1} = y_i + h\Psi(t_i, y_i, h)$ où $t_i = hi$.

On dit qu'une méthode est *stable* lorsqu'il existe une constante positive S appelée *constante de stabilité* telle que pour toutes suites finies (y_i), (y'_i) et (ε_i) telles que $\forall i \in \{0,1,\dots,n_{it}-1\}, \begin{cases} y_{i+1} = y_i + h\Psi(hi, y_i, h) \\ y'_{i+1} = y'_i + h\Psi(hi, y'_i, h) + \varepsilon_i \end{cases}$

on ait $\forall i \in \{0,1,\dots,n_{it}\}, \|y_i - y'_i\| \le S\left(\|y_0 - y'_0\| + \sum_{j=0}^{n_{it}-1} \|\varepsilon_j\| \right)$

Théorème : Si la fonction Ψ est Λ-lipchitzienne en Y, alors la méthode est stable et $S = e^{\Lambda T}$ où $T = hn_{it}$ en est une constante de stabilité.

La quantité $e_i = Y(h(i+1)) - y_{i+1}$ est appelée *erreur de consistance* et on a $e_i = Y(h(i+1)) - Y(hi) - h\Psi(hi, Y(hi), h)$.

La méthode est dite d'*ordre* $\ge p$ lorsque pour toute solution exacte Y de l'équation différentielle $Y' = f(t,Y)$ où f est de classe C^p il existe une constante $C \ge 0$ telle que l'erreur de consistance vérifie $\forall i \in \{0,1,\dots,n_{it}-1\}, \|e_i\| \le Ch^{p+1}$.

Propriété : La méthode est d'ordre au moins p si et seulement si Ψ est p fois dérivable par rapport à sa troisième variable h et vérifie la propriété suivante :

$$\forall l \in \{0,1,\dots,p-1\}, \frac{\partial^l \Psi}{\partial h^l}(t,Y,0) = \frac{1}{l+1} f^{[l]}(t,Y)$$

avec $f^{[l]}$ défini par récurrence de la manière suivante : $\begin{cases} f^{[1]} = \frac{\partial f}{\partial t} + \frac{\partial f}{\partial y} f \\ f^{[l+1]} = \left(f^{[l]} \right)^{[1]} \end{cases}$.

Majoration de l'erreur globale E: On appelle *erreur globale* la quantité $E = \max_{0 \le i \le n_{it}} \| Y(hi) - y_i \|$. C'est l'erreur maximale entre la solution exacte et la solution approchée calculée. Si la méthode est d'ordre p (C constante associée) et est stable de constante de stabilité S, on a $E \le SCTh^p$ où $T = n_{it} h$.

Algorithme d'Euler :

On se donne f, ci, h le pas de la résolution et n_{it} le nombre d'itérations.

On renvoie un vecteur sol de taille $n_{it}+1$ qui contient en position i le vecteur $Y(t = h.i)$ calculé.

$sol(0) \leftarrow ci$

Pour i allant de 1 à n_{it} faire

$sol(i) \leftarrow sol(i-1) + h.f(h.(i-1), sol(i-1))$

Fin faire

Renvoyer sol.

Le temps de calcul en nombre d'affectations est en $\Theta(n_{it})$.

Le temps de calcul en nombre d'opérations dépend du temps de calcul en nombre d'opérations de f que nous notons $C_{op}(f)$. Le temps de calcul de sol en nombre d'opérations est alors en $C_{op}(f).O(n_{it})$.

Ici, $\Psi(t,Y,h) = f(t,Y)$.

On peut montrer que cette méthode est stable et d'ordre 1.

Algoritme de Runge-Kutta 4 :

On se donne f, ci, h le pas de la résolution et n_{it} le nombre d'itérations.

On renvoie un vecteur sol de taille $n_{it}+1$ qui contient en position i le vecteur $Y(t = hi)$ calculé.

$sol(0) \leftarrow ci$

Pour i allant de 1 à n_{it} faire

$k_1 \leftarrow hf(h(i-1), sol(i-1))$

$k_2 \leftarrow hf(h(i-1/2), sol(i-1)+\frac{1}{2}k_1)$
$k_3 \leftarrow hf(h(i-1/2), sol(i-1)+\frac{1}{2}k_2)$
$k_4 \leftarrow hf(hi, sol(i)+k_3)$
$sol(i) \leftarrow sol(i-1)+\frac{1}{6}(k_1+2k_2+2k_3+k_4)$
Fin faire
Renvoyer sol.

On peut montrer que la méthode est stable et d'ordre 4.

2) Mise sous forme réduite de l'équation différentielle : utilisation de l'algorithme de Crout de décomposition d'une matrice réelle symétrique en produit de matrices triangulaires

Il s'agit ici de calculer à partir des équations différentielles obtenues dans la partie I la fonction *f* de la partie précédente, avec

$$Y = \begin{pmatrix} \theta_1 \\ \dot{\theta}_1 \\ \theta_2 \\ \dot{\theta}_2 \\ \vdots \\ \theta_n \\ \dot{\theta}_n \end{pmatrix} \text{ de telle sorte que } \dot{Y} = \begin{pmatrix} \dot{\theta}_1 \\ \ddot{\theta}_1 \\ \dot{\theta}_2 \\ \ddot{\theta}_2 \\ \vdots \\ \dot{\theta}_n \\ \ddot{\theta}_n \end{pmatrix}.$$

En clair, on doit avoir l'expression des $\ddot{\theta}_i$ en fonction des données du problème, des $\dot{\theta}_i$ et des θ_i. Pour cela on doit résoudre le système linéaire suivant :

$$\underbrace{\begin{pmatrix} n & (n-1)\cos(\theta_1-\theta_2) & \cdots & 2\cos(\theta_1-\theta_{n-1}) & \cos(\theta_1-\theta_n) \\ (n-1)\cos(\theta_1-\theta_2) & n-1 & \cdots & 2\cos(\theta_2-\theta_{n-1}) & \cos(\theta_2-\theta_n) \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 2\cos(\theta_1-\theta_{n-1}) & 2\cos(\theta_2-\theta_{n-1}) & \cdots & 2 & \cos(\theta_{n-1}-\theta_n) \\ \cos(\theta_1-\theta_n) & \cos(\theta_2-\theta_n) & \cdots & \cos(\theta_{n-1}-\theta_n) & 1 \end{pmatrix}}_{M} \begin{pmatrix} \ddot{\theta}_1 \\ \ddot{\theta}_2 \\ \vdots \\ \ddot{\theta}_{n-1} \\ \ddot{\theta}_n \end{pmatrix} = -\begin{pmatrix} n\frac{g}{l}\sin\theta_1 + \sum_{j=2}^{n}(n+1-j)\dot{\theta}_j^{\,2}\sin(\theta_1-\theta_j) \\ (n-1)\frac{g}{l}\sin\theta_2 + (n-1)\dot{\theta}_1^{\,2}\sin(\theta_2-\theta_1) + \sum_{j=3}^{n}(n+1-j)\dot{\theta}_j^{\,2}\sin(\theta_2-\theta_j) \\ \vdots \\ (n+1-k)\frac{g}{l}\sin\theta_k + (n+1-k)\sum_{i=1}^{k-1}\dot{\theta}_i^{\,2}\sin(\theta_k-\theta_i) + \sum_{j=k+1}^{n}(n+1-j)\dot{\theta}_j^{\,2}\sin(\theta_k-\theta_j) \\ \vdots \\ 2\frac{g}{l}\sin\theta_{n-1} + 2\sum_{i=1}^{n-2}\dot{\theta}_i^{\,2}\sin(\theta_{n-1}-\theta_i) + \dot{\theta}_n^{\,2}\sin(\theta_{n-1}-\theta_n) \\ \frac{g}{l}\sin\theta_n + \sum_{i=1}^{n-1}\dot{\theta}_i^{\,2}\sin(\theta_n-\theta_i) \end{pmatrix}$$

Nous allons décomposer M en produit de deux matrices triangulaires supérieure et inférieure, on parle de *décomposition de Crout*. La résolution pratique du système revient alors à la résolution successive de deux systèmes triangulaires ce qui ne pose pas de problème.

Théorème : Toute matrice symétrique réelle inversible M peut s'écrire $\boxed{M = LD^tL}$ où L est triangulaire inférieure avec que des 1 sur la diagonale et D est diagonale.

Algorithme de décomposition de Crout :
On se donne une matrice réelle symétrique inversible M de taille n. On note D un vecteur qui contiendra les coefficients diagonaux de la matrice diagonale, et on note L la future matrice triangulaire inférieure que l'on cherche contenant initialement que des 0.

Pour k allant de 0 à $n-1$ faire
    $L_{k,k} \leftarrow 1$
    Pour i allant de $k+1$ à $n-1$ faire
        $L_{i,k} \leftarrow M_{i,k} / M_{k,k}$
        Pour j allant de $k+1$ à $n-1$ faire
            $M_{i,j} \leftarrow M_{i,j} - L_{i,k} M_{k,j}$
        Fin faire
    Fin faire
    $D_k \leftarrow M_{k,k}$
Fin faire
Renvoyer D et L

Le temps de calcul est en n^3.

3) Résultats

Nous avons écrit des programmes (en Camllight) qui à partir d'un nombre de pendules et d'une condition initiale donnés décrivent au mieux le mouvement du système. Nous avons élaboré trois programmes. Le premier dessine la trajectoire de chaque pendule. Le deuxième est animé : on y voit les tiges des pendules bouger. C'est certainement celui-ci qui donne une meilleure description du système réel. Le troisième fournit un tableau donnant les angles par rapport à la verticale en fonction du temps et permet ainsi des considérations numériques.

Les programmes que nous avons réalisés sont en annexe. Les angles sont exprimés en rad, les temps en s. Voici des résultats obtenus.

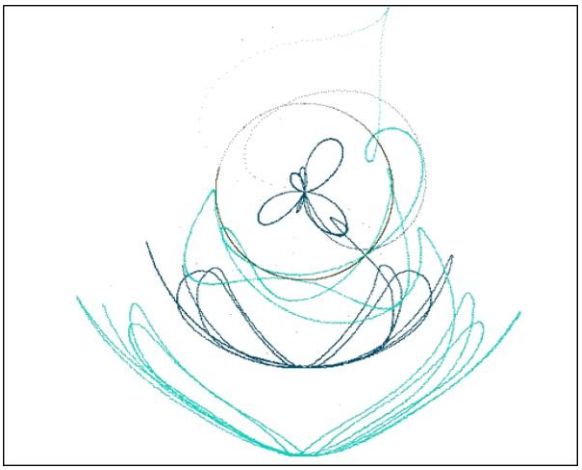
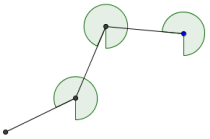


Figure 1

Résultat obtenu à partir de la fonction
`resolution_trajonly_rk_vnulle 0.001 100000 3 1. ;;`
La condition initiale était

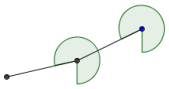
$$\begin{cases} \theta_1 = 4.62077762709 \\ \theta_2 = 5.89149857543 \\ \theta_3 = 5.16793634317 \end{cases}$$



On voit que la trajectoire est assez régulière au début. Elle diverge par la suite, c'est certainement dû à des imprécisions de calcul.

Avec un pendule double, on obtient des résultats un peu plus réguliers lorsque les conditions initiales sont éloignées des positions d'équilibre instable.
`resolution_trajonly_rk_vnulle 0.001 100000 2 1. ;;`
La condition initiale était

$$\begin{cases} \theta_1 = 5.16567739765 \\ \theta_2 = 4.93895293423 \end{cases}$$



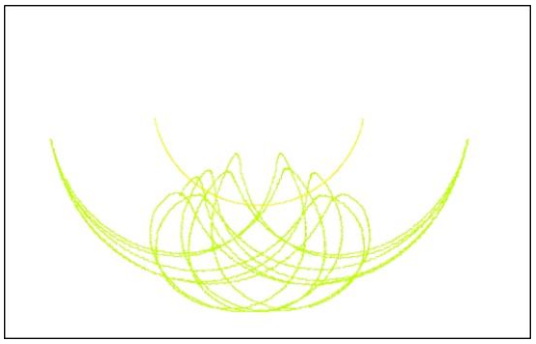


Figure 2

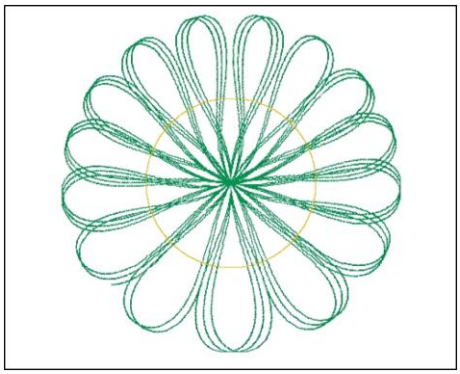
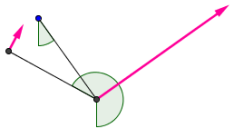


Figure 3

Résultat obtenu à partir de la fonction
`resolution_trajonly_rk 0.001 100000 2 1. ;;`
La condition initiale était

$$\begin{cases} \theta_1 = 0.621989847563 \\ \dot{\theta}_1 = 3.5971566492 \\ \theta_2 = 4.20412561222 \\ \dot{\theta}_2 = -0.680278195629 \end{cases}$$



Ce type de figure n'existe pas physiquement car en réalité, il y a des frottements.

Sur cette image, on voit deux pendules doubles lancés avec des positions initiales proches. On remarque que le pendule rose et vert prend très vite de l'avance sur le pendule jaune et bleu. Il a été obtenu à partir de

```
resolution_trajonlydouble_ci_rk        0.001
50000 0.1 ([
{couleur=yellow;theta=3.1415;thetapoint=0.
};{couleur=blue;theta=0.;thetapoint=0.}],[
{couleur=green;theta=3.1414;thetapoint=0.};
{couleur=magenta;theta=0.0001;thetapoint=0
.}]) ;;
```

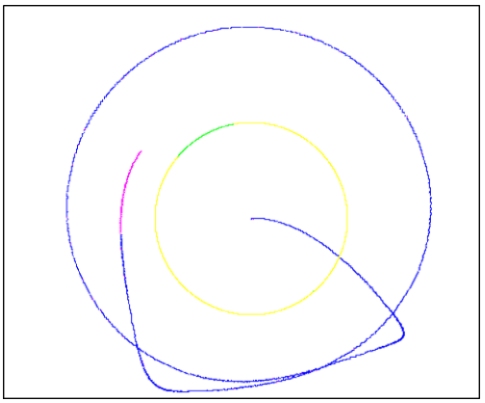
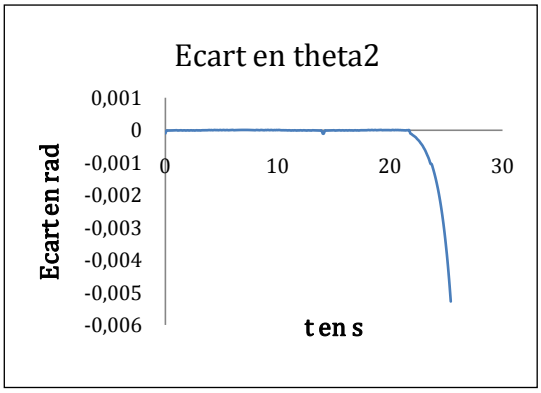


Figure 4



Ce graphique, obtenu à l'aide du programme `rkcopydouble` donne l'écart de θ_2 pour les deux pendules en fonction du temps. On constate qu'il augmente très vite.

Là aussi, on voit la trajectoire de deux pendules doubles lancés avec des conditions initiales proches. On l'a obtenu à partir de

```
resolution_trajonlydouble_ci_rk  0.001  50000
0.1     ([|{couleur=rgb     12     255     42;
theta=3.142;thetapoint=(0.001)};{couleur=rgb
175 50 12; theta=1.400; thetapoint=0.0001}|],
[|{couleur=rgb   128   128   0;   theta=3.141;
thetapoint=0.001};   {couleur=rgb   128   25
128;theta=1.401; thetapoint=(0.0001)}|]);;
```

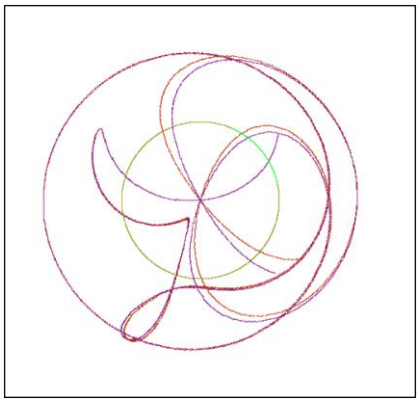
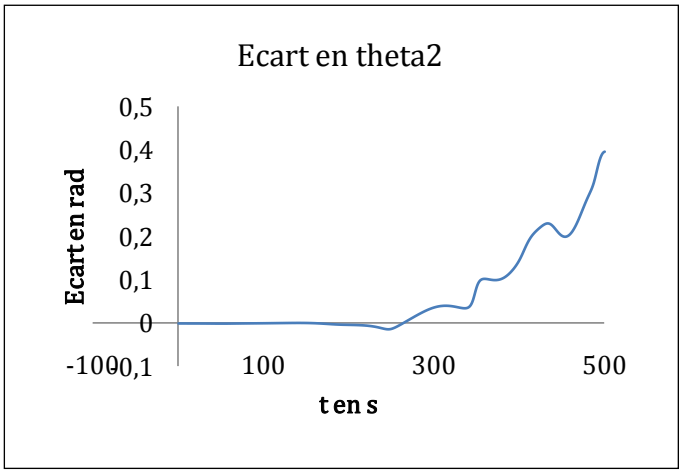


Figure 5



On constate de nouveau que l'écart de θ_2 augmente avec le temps.

Nous avons utilisé le programme `rktrajonly_em` pour confronter notre simulation à la réalité. Pour $n \geq 2$, et pour `marge` de l'ordre de 10% (`0.1`), le programme renvoyait une erreur dès la première itération. Il semblerait, comme on peut le voir sur la figure 1, l'énergie mécanique augmente dans notre simulation. Cependant, nous n'avons pas pu déterminer la cause de cette augmentation. Nous n'avons donc pas confronté nos résultats au pendule double qui nous avait été prêté.

Conclusion :

Nous avons constaté que bien que les équations du mouvement soient non-dissipatives, l'énergie mécanique de notre système simulé n'est pas une constante du mouvement (elle augmente !!), ce qui prouve que notre résolution des équations du mouvement n'est pas assez précise.

Même si nos résultats ont été surprenants, notre travail a été utile. Cela nous a permis d'étudier et de comprendre des algorithmes souvent utilisés pour résoudre numériquement d'une part des systèmes linéaires, d'autre part des équations différentielles. Nous avons été confrontés à une difficulté : celle de l'influence des erreurs d'arrondis. Dès lors, cela revalorise l'importance de la stabilité des algorithmes.

De plus, nous avons pu mettre en évidence le phénomène de chaos qui se traduit par une forte dépendance aux conditions initiales. C'est d'ailleurs ce type de phénomènes qui constitue l'intérêt principal de l'étude des pendules multiples. Comprendre comment se manifeste le chaos dans ce système apparemment simple peut nous aider à modéliser plus précisément les mouvements planétaires car ils sont eux-aussi chaotiques.

Contact :

H.R. Jauslin (ICB - Laboratoire Interdisciplinaire Carnot de l'Université de Bourgogne) enseignant-chercheur

Bibliographie :

Analyse Numérique et Equations Différentielles de Jean-Pierre DEMAILLY

Sources internet :

Interpolation polynomiale, intégration numérique, résolution numérique d'équations différentielles de Gilles LEBORGNE

Méthodes numériques de Laurent SIGNAC

Panorama des méthodes de résolution numérique approchée des problèmes différentiels de conditions initiales (Annales scientifiques de l'université de Clermont-Ferrand 2) de J. KUNTZMANN

Méthodes numériques de résolution d'équations différentielles (Université de Provence) de Brian STOUT

Équations différentielles (INP Toulouse) de J. GERGAUD

Décomposition LDtL=tUDU "incrémentale" d'une matrice symétrique de Y. RÉMION

Dynamique non linéaire et chaos (Laboratoire d'Hydrodynamique, École Polytechnique) de Paul MANNEVILLE

Annexe : Le code utilisé

Le langage de programmation utilisé est Camllight.

Type pendule et génération aléatoire de pendules :

```
type pendule={couleur:int;theta:float;thetapoint:float};;
type pendule_multiple == pendule vect;;

#open "random";;
#open "graphics";;

(* fonction qui génère un n-pendule avec des conditions initiales
(positions et vitesses) et des couleurs aléatoires (nocolor sert à obtenir
des couleurs autres que du marron) *)
let générer_pm n = let pm=make_vect n {couleur=0;theta=0.;thetapoint=0.} in
  for i=0 to (n-1) do
    let nocolor=int 3 in
      match nocolor with
        |0->pm.(i)<-{couleur=(rgb 0 (int 255) (int 255));theta=(float
7.);thetapoint=(-.(float 5.))}
        |1->pm.(i)<-{couleur=(rgb (int 255) 0 (int 255));theta=(float
7.);thetapoint=(float 5.)}
        |_->pm.(i)<-{couleur=(rgb (int 255) (int 255) 0);theta=(float
7.);thetapoint=(float 5.)}
  done;
  pm;;

(* fonction qui génère un n-pendule de vitesse initiale nulle  et des
couleurs aléatoires*)
let générer_vnulle n =
let pm=make_vect n {couleur=0;theta=0.;thetapoint=0.} in
  for i=0 to (n-1) do
    let nocolor=int 2 in
      match nocolor with
        |0->pm.(i)<-{couleur=(rgb 0 (int 255) (int 255));theta=(float
7.);thetapoint=0.}
        |1->pm.(i)<-{couleur=(rgb (int 255) 0 (int 255));theta=(float
7.);thetapoint=0.}
        |_->pm.(i)<-{couleur=(rgb (int 255) (int 255) 0);theta=(float
7.);thetapoint=0.}
  done;
pm;;

(* fonction qui génère deux n-pendules de conditions initiales proches
différentes de epsilon qui est un vecteur de taille 2n telle que theta_i
est en position 2i et thetapoint_i est en position 2i+1 *)
let générer_pm_double n epsilon = let pm=make_vect n
{couleur=0;theta=0.;thetapoint=0.} and pmbis=make_vect n
{couleur=0;theta=0.;thetapoint=0.} in
  for i=0 to (n-1) do
    let nocolor=int 2 in
      match nocolor with
        |0->pm.(i)<-{couleur=(rgb 0 (int 255) (int 255));theta=(float
7.);thetapoint=(-.(float 5.))};
            pmbis.(i)<-{couleur=(rgb (int 255) 0 (int
255));theta=pm.(i).theta +. epsilon.(2*i);thetapoint=pm.(i).thetapoint +.
epsilon.(2*i+1)};
        |1->pm.(i)<-{couleur=(rgb (int 255) 0 (int 255));theta=(float
7.);thetapoint=(float 5.)};
```

```
                pmbis.(i)<-{couleur=(rgb (int 255) (int 255)
0);theta=pm.(i).theta +. epsilon.(2*i);thetapoint=pm.(i).thetapoint +.
epsilon.(2*i+1)};
          |_->pm.(i)<-{couleur=(rgb (int 255) (int 255) 0);theta=(float
7.);thetapoint=(float 5.)};
                pmbis.(i)<-{couleur=(rgb 0 (int 255) (int
255));theta=pm.(i).theta +. epsilon.(2*i);thetapoint=pm.(i).thetapoint +.
epsilon.(2*i+1)};
   done;
   (pm,pmbis);;

(* idem sans vitesse initiale *)
let générer_pm_double_vnulle n epsilon = let pm=make_vect n
{couleur=0;theta=0.;thetapoint=0.} and pmbis=make_vect n
{couleur=0;theta=0.;thetapoint=0.} in
   for i=0 to (n-1) do
      let nocolor=int 2 in
         match nocolor with
           |0->pm.(i)<-{couleur=(rgb 0 (int 255) (int 255));theta=(float
7.);thetapoint=0.};
                pmbis.(i)<-{couleur=(rgb (int 255) 0 (int
255));theta=pm.(i).theta +. epsilon.(2*i);thetapoint=epsilon.(2*i+1)};
          |1->pm.(i)<-{couleur=(rgb (int 255) 0 (int 255));theta=(float
7.);thetapoint=0.};
                pmbis.(i)<-{couleur=(rgb (int 255) (int 255)
0);theta=pm.(i).theta +. epsilon.(2*i);thetapoint=epsilon.(2*i+1)};
          |_->pm.(i)<-{couleur=(rgb (int 255) (int 255) 0);theta=(float
7.);thetapoint=0.};
                pmbis.(i)<-{couleur=(rgb 0 (int 255) (int
255));theta=pm.(i).theta +. epsilon.(2*i);thetapoint=epsilon.(2*i+1)};
   done;
   (pm,pmbis);;
```

Implémentation de l'algorithme d'Euler :

```
(* ci est de la forme [|THETA1;THETA1POINT;THETA2;THETA2POINT;...|]
Cette fonction calcule f(pas*i) par l'algorithme d'Euler *)
let euler pas nbit f ci =
  let y=make_matrix (nbit+1) (vect_length ci) 0. in
    for i=0 to (vect_length ci)-1 do
      y.(0).(i)<-ci.(i)
    done;
    for t=1 to nbit do
      for k=0 to (vect_length ci)-1 do
        y.(t).(k)<-y.(t-1).(k)+.pas*.(f y.(t-1)).(k)
      done
    done;
  y;;
```

Implémentation de l'algorithme de Runge-Kutta 4 :

```
  (* multiplication d'un vecteur par un scalaire *)
  let mscal scal v =
    let rep = make_vect (vect_length v) 0. in
      for i=0 to ((vect_length v)-1) do
        rep.(i)<-v.(i)*.scal
      done;
    rep;;

  (* addition de deux vecteurs *)
```

```
  let plus_vect v1 v2 =
    let rep = make_vect (vect_length v1) 0. in
      for i=0 to ((vect_length v1)-1) do
        rep.(i) <- (v1.(i) *. v2.(i))
      done;
    rep;;

(* calcul de f par l'algorithme de runge-kutta 4 ; renvoie une matrice
contenant les positions et les vitesses angulaires (lignes) en fonction de
t=h*i (colonnes) *)
let rk pas nbit f ci =
  let m=(vect_length ci)-1 in
    let y=make_matrix (nbit+1) (m+1) 0.
    and k1=ref (make_vect (m+1) 0.)
    and k2=ref (make_vect (m+1) 0.)
    and k3=ref (make_vect (m+1) 0.)
    and k4=ref (make_vect (m+1) 0.) in
      for i=0 to m do
        y.(0).(i)<-ci.(i)
      done;
      for t=1 to nbit do
        k1:=mscal pas (f y.(t-1));
        k2:=mscal pas (f (plus_vect y.(t-1) (mscal 0.5 !k1)));
        k3:=mscal pas (f (plus_vect y.(t-1) (mscal 0.5 !k2)));
        k4:=mscal pas (f (plus_vect y.(t-1) !k3));
        y.(t)<-plus_vect y.(t-1) (mscal (1./.6.) (plus_vect !k1
(plus_vect !k4 (mscal 2. (plus_vect !k3 !k2)))));
      done;
y;;
```

Implémentation de la résolution de systèmes triangulaires supérieur puis inférieur :

```
(*résolution de PX=B avec P triangulaire supérieure ou inferieure*)

   (*calcul la somme des produits des coordonnées de c par celle de la
   colonne i de M*)
   let somme_pcl M i c = let n = vect_length c and r=ref 0. in
     for k = 0 to n-1 do
       r:=!r+.(M.(i)).(k)*.c.(k);
     done;
     !r
   ;;

(* resolution d'un système triangulaire supérieur *)
let solve_tri_sup P B = let n = vect_length P in
   let X = make_vect n 0. in
     for i = n-1 downto 0 do
       X.(i)<-(B.(i)-.(somme_pcl P i X))/.(P.(i)).(i);
     done;
   X
;;

(* resolution d'un système triangulaire inférieur *)
let solve_tri_inf P B = let n = vect_length P in
   let X = make_vect n 0. in
     for i = 0 to n-1 do
       X.(i)<-(B.(i)-.(somme_pcl P i X))/.(P.(i)).(i);
     done;
   X
;;
```

Implémentation de l'algorithme de Crout :

```
(*résolution d'un système linéaire symétrique : factorisation de Crout
M=LDtL avec L triangulaire inférieure*)
let crout m = let n=vect_length m in
  let d=make_vect n 0. and l=make_matrix n n 0. in
    for k=0 to n-1 do
      l.(k).(k)<-1.;
      for i=k+1 to n-1 do
        l.(i).(k)<-m.(i).(k)/.m.(k).(k);
        for j=k+1 to n-1 do
          m.(i).(j)<-m.(i).(j)-.l.(i).(k)*.m.(k).(j)
        done;
      done;
      d.(k)<-m.(k).(k)
    done;
  (d,l)
;;
```

Implémentation du calcul de *f* :

```
(* transpose une matrice carrée *)
let transpose M = let n = vect_length M  and p = vect_length M.(0) in
  let N = make_matrix p n (M.(0)).(0) in
    for i = 0 to p-1 do
      for j = 0 to n-1 do
        (N.(i)).(j)<-(M.(j)).(i);
      done;
    done;
  N
;;

(* calcule la matrice du système des theta deux points à partir du rang n *)
(* v est de la forme [|THETA1;THETA1POINT;THETA2;THETA2POINT;...|] *)
(* on ne remplit que la moitié supérieur car nous n'avons besoin de celle-
ci *)
let create_matrix n v =
  let m=make_matrix n n 0. in
    for i=0 to (n-1) do
      for j=i to (n-1) do
        m.(i).(j) <- (float_of_int (n-(max i j)))*.(cos (v.(2*i) -.
v.(2*j)));
        m.(j).(i) <- (float_of_int (n-(max i j)))*.(cos (v.(2*i) -.
v.(2*j)))
      done;
    done;
  m
;;

(* calcule le vecteur constant *)
(* v est de la forme [|THETA1;THETA1POINT;THETA2;THETA2POINT;...|] *)
(* q est égal au rapport g/l *)
let create_b n q v =
  let b=make_vect n 0. in
    for k=1 to n do
      for i=1 to (k-1) do
        b.(k-1) <- (v.(2*i-1) *. v.(2*i-1)) *. (sin (v.(2*k-2) -. v.(2*i-
2)))+. b.(k-1)
      done;
      b.(k-1) <- (float_of_int (n+1-k))*.(b.(k-1)) ;
```

```
        for j=k+1 to n do
          b.(k-1) <- b.(k-1) +. (float_of_int (n+k-j))*.(v.(2*j-1) *.
v.(2*j-1))*. (sin (v.(2*j-2) -. v.(2*k-2)))
        done;
        b.(k-1) <- -. (b.(k-1) +. (float_of_int (n+1-k)) *. q *. (sin
(v.(2*k-2))))
     done;
   b
;;

(* calcule f *)
let create_f n q v =
   let m=create_matrix n v and b=create_b n q v in
     let (d,l) = crout m in
       let Y = solve_tri_inf l b in
         let U=transpose l in
           for i=0 to n-1 do
             for j=0 to i do
               U.(i).(j)<-U.(i).(j)*.d.(i)
             done;
           done;
           let rep=solve_tri_sup U b and fv=make_vect (2*n) 1. in
             for i=0 to n-1 do
               fv.(2*i)<-v.(2*i+1);
               fv.(2*i+1)<-rep.(i);
             done;
             fv
;;
```

Dessin de la trajectoire des pendules :

```
(* dessine la trajectoire de chaque pendule dans sa couleur calculée par
l'algorithme de runge-kutta 4 au fur et à mesure *)
let rktrajonly pas nbit f pendule = let n = (vect_length pendule) in
let ci=make_vect (2*n) 0. in
   for i=0 to n-1 do
     ci.(2*i)<-pendule.(i).theta;
     ci.(2*i+1)<-pendule.(i).thetapoint
   done;
   open_graph "1600x900";
   let y=make_matrix 2 (2*n) 0.
   and centre1=ref 0
   and centre2=ref 0
   and k1=ref (make_vect n 0.)
   and k2=ref (make_vect n 0.)
   and k3=ref (make_vect n 0.)
   and k4=ref (make_vect n 0.) in
     for i=0 to 2*n-1 do
       y.(0).(i)<-ci.(i)
     done;
     for t=1 to nbit do
       centre1:=800;
       centre2:=450;
       k1:=mscal pas (f y.((t-1) mod 2));
       k2:=mscal pas (f (plus_vect y.((t-1) mod 2) (mscal 0.5 !k1)));
       k3:=mscal pas (f (plus_vect y.((t-1) mod 2) (mscal 0.5 !k2)));
       k4:=mscal pas (f (plus_vect y.((t-1) mod 2) !k3));
       y.(t mod 2)<-plus_vect y.((t-1) mod 2) (mscal (1./.6.)
(plus_vect !k1 (plus_vect !k4 (mscal 2. (plus_vect !k3 !k2)))));
       for k=0 to n-1 do
         set_color pendule.(k).couleur;
```

```
          centre1:=((int_of_float (100.*.(sin y.(t mod
2).(2*k))))+(!centre1));
          centre2:=(-(int_of_float (100.*.(cos y.(t mod
2).(2*k))))+(!centre2));
          plot !centre1 !centre2;
        done;
      done;
   pendule;;

(* à partir d'un n-pendule choisi aléatoirement *)
let resolution_trajonly_rk pas nbit n q = rktrajonly pas nbit (create_f n q)
(générer_pm n);;

let resolution_trajectoire_rk pas nbit n q = rktrajectoire pas nbit
(create_f n q) (générer_pm n);;

(* à partir d'un n-pendule choisi aléatoirement sans vitesse initiale *)
let resolution_trajectoire_vnulle_rk pas nbit n q = rktrajectoire pas nbit
(create_f n q) (générer_vnulle n);;

let resolution_trajonly_vnulle_rk pas nbit n q = rktrajonly pas nbit
(create_f n q) (générer_vnulle n);;

(* à partir de conditions initiales données : « pendules » de type pendules
*)
let resolution_trajonly_ci_rk pas nbit q pendules = rktrajonly pas nbit
(create_f (vect_length pendules) q) pendules;;

let resolution_trajectoire_ci_rk pas nbit q pendules = rktrajectoire pas
nbit (create_f (vect_length pendules) q) pendules;;
```

Dessin de la trajectoire en dégradé :

```
type qd = { couleur_queue : color ;
  taille : int ;
  mutable decalage : int ;
  couleurs : color vect ;
  points : (int * int) vect }
;;

type queues == qd vect;;

(* Crée une nouvelle queue dégradée, te couleur de tête définie par ses
composantes RGB : (coul_tete_r, coul_tete_g, coul_tete_b), et de couleur de
queue définie de même par : (coul_queue_r, coul_queue_g, coul_queue_b). On
pourra utiliser à bon escient la fonction palir pour obtenir une version
plus pâle de la couleur de tête pour la queue.
À propos de la taille, l'ajout de point avec addqd est en thêta(taille). *)
let newqd (coul_queue_r, coul_queue_g, coul_queue_b)
  (coul_tete_r, coul_tete_g, coul_tete_b) taille =
  let couleurs = make_vect taille white in
    for i = 0 to taille - 1  do
      (* On calcule une bonne fois pour toutes les couleurs qui vont
constituer le dégradé, en mélangeant en proportions variant linéairement
les couleurs de tête et de queue. *)
      couleurs.(i) <- rgb (coul_queue_r - ((coul_queue_r - coul_tete_r) * i)
/ (taille - 1))
        (coul_queue_g - ((coul_queue_g - coul_tete_g) * i) / (taille - 1))
        (coul_queue_b - ((coul_queue_b - coul_tete_b) * i) / (taille - 1))
    done ;
```

```
  {couleur_queue = rgb coul_queue_r coul_queue_g coul_queue_b ; couleurs =
couleurs ;
   taille = taille ; decalage = 0 ; points = make_vect taille (-1, -1)}
;;

(* Dessine la queue en entier, en utilisant les couleurs précalculées et
les points déjà ajoutés. Ne pas appeler directement cette fonction, mais
plutôt addqd. *)
let redraw qd =
  for i = qd.taille - 1 downto 0 do
    let (x, y) = qd.points.((i + qd.decalage) mod qd.taille) in
      set_color qd.couleurs.(i) ;
      plot x y
  done
;;

(* Ajoute un nouveau point à la queue qd, puis dessine la queue agrémentée
de ce nouveau point. *)
let addqd qd x y = if (qd.points.(qd.decalage) <> (x, y)) then begin
  set_color qd.couleur_queue ;
  let (a, b) = qd.points.((qd.decalage + 1) mod qd.taille) in
    plot a b ; (* erase last point *)
    qd.points.((qd.decalage + 1) mod qd.taille) <- (x, y) ;
    qd.decalage <- (qd.decalage + 1) mod qd.taille ;
    redraw qd
  end
;;

(* dessin de la trajectoire dégradée à partir du calcul de f par
l'algorithme de runge-kutta 4 *)
let rktrajnet pas nbit f pendule taille_queue = let n = (vect_length
pendule) in
let ci=make_vect (2*n) 0.
and queues = make_vect n (newqd (palir 8 (rgb_of_color pendule.(0).couleur))
(rgb_of_color pendule.(0).couleur) taille_queue) in
   for i=0 to n-1 do
      ci.(2*i)<-pendule.(i).theta;
      ci.(2*i+1)<-pendule.(i).thetapoint;
      queues.(i)<-newqd (palir 10 (rgb_of_color pendule.(i).couleur))
(rgb_of_color pendule.(i).couleur) taille_queue
   done;
   open_graph "1600x900";
   let y=make_matrix 2 (2*n) 0.
   and centre1=ref 0
   and centre2=ref 0
   and k1=ref (make_vect n 0.)
   and k2=ref (make_vect n 0.)
   and k3=ref (make_vect n 0.)
   and k4=ref (make_vect n 0.) in
      for i=0 to 2*n-1 do
        y.(0).(i)<-ci.(i)
      done;
      for t=1 to nbit do
        centre1:=800;
        centre2:=450;
        k1:=mscal pas (f y.((t-1) mod 2));
        k2:=mscal pas (f (plus_vect y.((t-1) mod 2) (mscal 0.5 !k1)));
        k3:=mscal pas (f (plus_vect y.((t-1) mod 2) (mscal 0.5 !k2)));
        k4:=mscal pas (f (plus_vect y.((t-1) mod 2) !k3));
        y.(t mod 2)<-plus_vect y.((t-1) mod 2) (mscal (1./.6.)
(plus_vect !k1 (plus_vect !k4 (mscal 2. (plus_vect !k3 !k2)))));
```

```
        for k=0 to n-1 do
          centre1:=((int_of_float (100.*.(sin y.(t mod
2).(2*k))))+(!centre1));
          centre2:=(-(int_of_float (100.*.(cos y.(t mod
2).(2*k))))+(!centre2));
          addqd queues.(k) !centre1 !centre2;
        done;
      done;
   pendule
;;

(* positions initiales quelconques, vitesses initiales nulles *)
let resolution_trajnet_rk_vnulle pas nbit n q taille = rktrajnet pas nbit
(create_f n q) (générer_vnulle n) taille;;

(* conditions initiales quelconques *)
let resolution_trajnet_rk pas nbit n q taille = rktrajnet pas nbit
(create_f n q) (générer_pm n) taille;;

(* conditions initiales données de type pendules *)
let resolution_trajnet_ci_rk pas nbit q pendules taille = rktrajnet pas
nbit (create_f (vect_length pendules) q) pendules taille;;
```

Dessin des pendules :

```
(* dessine un n-pendule sur une fenêtre graphique déjà ouverte
La couleur est celle du pendule donné en condition initiale « pendule » ;
La position est donnée par « y » vecteur de taille 2n contenant les theta
et les thetapoints *)
let tracer_pendules n y pendule = let centre1=ref 800 and centre2=ref 450
in
   for i=0 to n-1 do
     moveto !centre1 !centre2;
     set_color pendule.(i).couleur;
     centre1:=((int_of_float (100.*.(sin y.(2*i))))+(!centre1));
     centre2:=(-(int_of_float (100.*.(cos y.(2*i))))+(!centre2));
     lineto !centre1 !centre2;
   done;;

(* colorie un pendule en blanc (fonctionne comme la fonction précédente) *)
let effacer_pendules n y = let centre1=ref 800 and centre2=ref 450 in
   for i=0 to n-1 do
     moveto !centre1 !centre2;
     set_color white;
     centre1:=((int_of_float (100.*.(sin y.(2*i))))+(!centre1));
     centre2:=(-(int_of_float (100.*.(cos y.(2*i))))+(!centre2));
     lineto !centre1 !centre2;
   done;;

(* dessine les pendules à chaque iterations *)
let rkp pas nbit f pendule = let n = (vect_length pendule) in
let ci=make_vect (2*n) 0. in
   for i=0 to n-1 do
     ci.(2*i)<-pendule.(i).theta;
     ci.(2*i+1)<-pendule.(i).thetapoint
   done;
   open_graph "1600x900";
   set_line_width 5;
   let y=make_matrix 2 (2*n) 0.
   and k1=ref (make_vect n 0.)
   and k2=ref (make_vect n 0.)
```

```
  and k3=ref (make_vect n 0.)
  and k4=ref (make_vect n 0.) in
    for i=0 to 2*n-1 do
      y.(0).(i)<-ci.(i)
    done;
    for t=1 to nbit do
      k1:=mscal pas (f y.((t-1) mod 2));
      k2:=mscal pas (f (plus_vect y.((t-1) mod 2) (mscal 0.5 !k1)));
      k3:=mscal pas (f (plus_vect y.((t-1) mod 2) (mscal 0.5 !k2)));
      k4:=mscal pas (f (plus_vect y.((t-1) mod 2) !k3));
      y.(t mod 2)<-plus_vect y.((t-1) mod 2) (mscal (1./.6.)
(plus_vect !k1 (plus_vect !k4 (mscal 2. (plus_vect !k3 !k2)))));
      effacer_pendules n y.((t-1) mod 2);
      tracer_pendules n y.(t mod 2) pendule;
    done;
  pendule;;

let resolution_p_rk_vnulle pas nbit n q = rkp pas nbit (create_f n q)
(générer_vnulle n);;

let resolution_p_rk pas nbit n q = rkp pas nbit (create_f n q) (générer_pm
n);;

let resolution_p_ci_rk pas nbit q pendules = rkp pas nbit (create_f
(vect_length pendules) q) pendules;;
```

Dessin de deux pendules superposés :

```
(* superpose la trajectoire de deux pendules *)
let rktrajonlydouble pas nbit f (pendule,pendulebis) = let n = (vect_length
pendule) in
let ci=make_vect (2*n) 0.
and cibis=make_vect (2*n) 0. in
  for i=0 to n-1 do
    ci.(2*i)<-pendule.(i).theta;
    ci.(2*i+1)<-pendule.(i).thetapoint;
    cibis.(2*i)<-pendulebis.(i).theta;
    cibis.(2*i+1)<-pendulebis.(i).thetapoint
  done;
  open_graph "1600x900";
  let y=make_matrix 2 (2*n) 0.
  and centre1=ref 0
  and centre2=ref 0
  and k1=ref (make_vect n 0.)
  and k2=ref (make_vect n 0.)
  and k3=ref (make_vect n 0.)
  and k4=ref (make_vect n 0.)
  and ybis=make_matrix 2 (2*n) 0.
  and centre1bis=ref 0
  and centre2bis=ref 0
  and k1bis=ref (make_vect n 0.)
  and k2bis=ref (make_vect n 0.)
  and k3bis=ref (make_vect n 0.)
  and k4bis=ref (make_vect n 0.) in
    for i=0 to 2*n-1 do
      y.(0).(i)<-ci.(i);
      ybis.(0).(i)<-cibis.(i)
    done;
    for t=1 to nbit do
      centre1:=800;
      centre2:=450;
```

```
        k1:=mscal pas (f y.((t-1) mod 2));
        k2:=mscal pas (f (plus_vect y.((t-1) mod 2) (mscal 0.5 !k1)));
        k3:=mscal pas (f (plus_vect y.((t-1) mod 2) (mscal 0.5 !k2)));
        k4:=mscal pas (f (plus_vect y.((t-1) mod 2) !k3));
        y.(t mod 2)<-plus_vect y.((t-1) mod 2) (mscal (1./.6.)
(plus_vect !k1 (plus_vect !k4 (mscal 2. (plus_vect !k3 !k2)))));
        centre1bis:=800;
        centre2bis:=450;
        k1bis:=mscal pas (f ybis.((t-1) mod 2));
        k2bis:=mscal pas (f (plus_vect ybis.((t-1) mod 2) (mscal
0.5 !k1bis)));
        k3bis:=mscal pas (f (plus_vect ybis.((t-1) mod 2) (mscal
0.5 !k2bis)));
        k4bis:=mscal pas (f (plus_vect ybis.((t-1) mod 2) !k3bis));
        ybis.(t mod 2)<-plus_vect ybis.((t-1) mod 2) (mscal (1./.6.)
(plus_vect !k1bis (plus_vect !k4bis (mscal 2. (plus_vect !k3bis !k2bis)))));
        for k=0 to n-1 do
          set_color pendule.(k).couleur;
          centre1:=((int_of_float (100.*.(sin y.(t mod
2).(2*k))))+(!centre1));
          centre2:=(-(int_of_float (100.*.(cos y.(t mod
2).(2*k))))+(!centre2));
          plot !centre1 !centre2;
        done;
        for k=0 to n-1 do
          set_color pendulebis.(k).couleur;
          centre1bis:=((int_of_float (100.*.(sin ybis.(t mod
2).(2*k))))+(!centre1bis));
          centre2bis:=(-(int_of_float (100.*.(cos ybis.(t mod
2).(2*k))))+(!centre2bis));
          plot !centre1bis !centre2bis;
        done;
     done;
(pendule,pendulebis);;

let resolution_trajonlydouble_rk_vnulle pas nbit n q epsilon =
rktrajonlydouble pas nbit (create_f n q) (générer_pm_double_vnulle n
epsilon) ;;

let resolution_trajonlydouble_rk pas nbit n q epsilon = rktrajonlydouble
pas nbit (create_f n q) (générer_pm_double n epsilon) ;;

let resolution_trajonlydouble_ci_rk pas nbit q (pendule,pendulebis) =
rktrajonlydouble pas nbit (create_f (vect_length pendule) q)
(pendule,pendulebis) ;;
```

Récupération des données :

```
(* transforme un float x.y "y" *)
let tl_nb flottant = let reponse=string_of_float (flottant -. float_of_int
(int_of_float flottant)) in
  let n= string_length reponse in
    sub_string reponse 2 (n-2) ;;

(* transforme un float x.y en "y,x" (format des flottants pour excel) *)
let rec copy_nb nb = if nb>=0. then ((string_of_int (int_of_float
nb))^","^(tl_nb nb))
  else ("-"^(copy_nb (-.nb)));;

(* dessinne et copie dans un fichier de nom « nf » (string) les positions
et vitesses de chaque pendules en fonction de t *)
```

```
let rkcopy pas nbit f pendule nf q= let n = (vect_length pendule) in
let ci=make_vect (2*n) 0. in
  for i=0 to n-1 do
    ci.(2*i)<-pendule.(i).theta;
    ci.(2*i+1)<-pendule.(i).thetapoint
  done;
  open_graph "1600x900";
  let y=make_matrix 2 (2*n) 0.
  and centre1=ref 0
  and centre2=ref 0
  and k1=ref (make_vect n 0.)
  and k2=ref (make_vect n 0.)
  and k3=ref (make_vect n 0.)
  and k4=ref (make_vect n 0.)
  and out=(open_out (nf^".csv")) in
    output_string out ("h="^(string_of_float pas)^";q="^(string_of_float
q)^"\n") ;
    for i=1 to n do
      output_string out (";theta"^(string_of_int
i)^";thetapoint"^(string_of_int i))
    done;
    output_string out "\n" ;
    output_string out "0," ;
    for i=0 to 2*n-1 do
      y.(0).(i)<-ci.(i) ;
      output_string out (";"^(copy_nb y.(0).(i))) ;
    done;
    output_string out "\n" ;
    for t=1 to nbit do
      centre1:=800;
      centre2:=450;
      k1:=mscal pas (f y.((t-1) mod 2));
      k2:=mscal pas (f (plus_vect y.((t-1) mod 2) (mscal 0.5 !k1)));
      k3:=mscal pas (f (plus_vect y.((t-1) mod 2) (mscal 0.5 !k2)));
      k4:=mscal pas (f (plus_vect y.((t-1) mod 2) !k3));
      y.(t mod 2)<-plus_vect y.((t-1) mod 2) (mscal (1./.6.)
(plus_vect !k1 (plus_vect !k4 (mscal 2. (plus_vect !k3 !k2)))));
      output_string out (string_of_float ((float_of_int t)*.pas)) ;
      for k=0 to n-1 do
        set_color pendule.(k).couleur;
        centre1:=((int_of_float (100.*.(sin y.(t mod
2).(2*k))))+(!centre1));
        centre2:=(-(int_of_float (100.*.(cos y.(t mod
2).(2*k))))+(!centre2));
        plot !centre1 !centre2;
        output_string out (";"^(copy_nb y.(t mod 2).(2*k))) ;
        output_string out (";"^(copy_nb y.(t mod 2).(2*k+1)))
      done;
    output_string out "\n" ;
    done;
  close_out out ;
  pendule;;

let resolution_copy_rk pas nbit q pendule nf = rkcopy pas nbit (create_f
(vect_length pendule) q) pendule nf q ;;

(* idem avec deux pendules *)
let rkcopydouble pas nbit f pendule pendulebis nf q = let n = (vect_length
pendule) in
let ci=make_vect (2*n) 0. and cibis=make_vect (2*n) 0. in
  for i=0 to n-1 do
```

```
      ci.(2*i)<-pendule.(i).theta;
      ci.(2*i+1)<-pendule.(i).thetapoint;
      cibis.(2*i)<-pendulebis.(i).theta;
      cibis.(2*i+1)<-pendulebis.(i).thetapoint;
   done;
   let y=make_matrix 2 (2*n) 0.
   and ybis=make_matrix 2 (2*n) 0.
   and k1=ref (make_vect n 0.)
   and k2=ref (make_vect n 0.)
   and k3=ref (make_vect n 0.)
   and k4=ref (make_vect n 0.)
   and k1bis=ref (make_vect n 0.)
   and k2bis=ref (make_vect n 0.)
   and k3bis=ref (make_vect n 0.)
   and k4bis=ref (make_vect n 0.)
   and out=(open_out (nf^".csv")) in
      output_string out ("h="^(copy_nb pas)^";q="^(copy_nb q)^"\n") ;
      for i=1 to n do
        output_string out (";theta"^(string_of_int
i)^";thetabis"^(string_of_int i)^";erreur"^
          ";thetapoint"^(string_of_int i)^";thetapointbis"^(string_of_int
i)^";erreur")
      done;
      output_string out "\n" ;
      output_string out "0" ;
      for i=0 to 2*n-1 do
        y.(0).(i)<-ci.(i);
        ybis.(0).(i)<-cibis.(i);
        output_string out (";"^(copy_nb ci.(i))^";"^(copy_nb cibis.(i))^";")
      done;
      output_string out "\n" ;
      for t=1 to nbit do
        k1:=mscal pas (f y.((t-1) mod 2));
        k2:=mscal pas (f (plus_vect y.((t-1) mod 2) (mscal 0.5 !k1)));
        k3:=mscal pas (f (plus_vect y.((t-1) mod 2) (mscal 0.5 !k2)));
        k4:=mscal pas (f (plus_vect y.((t-1) mod 2) !k3));
        y.(t mod 2)<-plus_vect y.((t-1) mod 2) (mscal (1./.6.)
(plus_vect !k1 (plus_vect !k4 (mscal 2. (plus_vect !k3 !k2)))));
        k1bis:=mscal pas (f ybis.((t-1) mod 2));
        k2bis:=mscal pas (f (plus_vect ybis.((t-1) mod 2) (mscal
0.5 !k1bis)));
        k3bis:=mscal pas (f (plus_vect ybis.((t-1) mod 2) (mscal
0.5 !k2bis)));
        k4bis:=mscal pas (f (plus_vect ybis.((t-1) mod 2) !k3bis));
        ybis.(t mod 2)<-plus_vect ybis.((t-1) mod 2) (mscal (1./.6.)
(plus_vect !k1bis (plus_vect !k4bis (mscal 2. (plus_vect !k3bis !k2bis)))));
        output_string out (copy_nb ((float_of_int t)*.pas)) ;
        for k=0 to n-1 do
          output_string out (";"^(copy_nb y.(t mod 2).(2*k))^";"^(copy_nb
ybis.(t mod 2).(2*k))^";");
          output_string out (";"^(copy_nb y.(t mod 2).(2*k+1))^";"^(copy_nb
ybis.(t mod 2).(2*k+1))^";")
        done;
      output_string out "\n" ;
      done;
   close_out out ;
   "Travail fini !";;

let resolution_copy_double_rk pas nbit q pendule pendulebis nf =
rkcopydouble pas nbit (create_f (vect_length pendule) q) pendule pendulebis
nf q ;;
```

Vérification par le calcul de l'énergie mécanique :

```
(* fonction qui calcule l'énergie cinétique du système *)
let ec m q y =
   let l=9.814/.q in
   let n=((vect_length y)/2 -1) in
   let v=ref 0. in
      for i=0 to n do
         let a=ref 0. in
            for k=0 to i do
               a:= !a +. (y.(2*k+1))*.(y.(2*k+1))
            done;
         let b=ref 0. in
            for j=0 to n do
               for k=j+1 to i do
                  b:= !b +. (y.(2*j+1))*.(y.(2*k+1))*.(cos(y.(2*j)-.y.(2*k)))
               done;
            done;
         v:= !v +. 0.5*.m*.l*.l*.( !a +. !b*.2. ) ;
      done;
   !v
;;

(* fonction qui calcule l'énergie potentielle du système *)
let ep m q y =
  let l=9.814/.q
  and n=((vect_length y)/2 -1)
  and e=ref 0. in
    for i=0 to n do
      e:= !e -. (float_of_int (n+1-i))*.m*.l*.9.814*.cos(y.(2*i))
    done;
  !e
;;

(* fonction qui calcule l'énergie mécanique du système *)
let em m q y = (ec m q y)+.(ep m q y) ;;

(* fonction qui au fur et à mesure du calcul de la position du système
vérifie la constance de l'énergie mécanique du système et s'arrête lorsque
la valeur absolue de la différence de l'énergie mécanique à l'instant t et
de l'énergie mécaninique initiale dépasse marge*emi *)
let rktrajonly_em pas nbit f pendule m q marge= let n = (vect_length
pendule) in
let ci=make_vect (2*n) 0. in
   for i=0 to n-1 do
      ci.(2*i)<-pendule.(i).theta;
      ci.(2*i+1)<-pendule.(i).thetapoint
   done;
   let emi=em m q ci in
   open_graph "1600x900";
   let y=make_matrix 2 (2*n) 0.
   and centre1=ref 0
   and centre2=ref 0
   and k1=ref (make_vect n 0.)
   and k2=ref (make_vect n 0.)
   and k3=ref (make_vect n 0.)
   and k4=ref (make_vect n 0.) in
      for i=0 to 2*n-1 do
         y.(0).(i)<-ci.(i)
      done;
      let t=ref 1 in
```

```
      while !t<=nbit & (abs_float((em m q y.(!t mod 2))-.emi)<=marge*.emi)
do
        centre1:=800;
        centre2:=450;
        k1:=mscal pas (f y.((!t-1) mod 2));
        k2:=mscal pas (f (plus_vect y.((!t-1) mod 2) (mscal 0.5 !k1)));
        k3:=mscal pas (f (plus_vect y.((!t-1) mod 2) (mscal 0.5 !k2)));
        k4:=mscal pas (f (plus_vect y.((!t-1) mod 2) !k3));
        y.(!t mod 2)<-plus_vect y.((!t-1) mod 2) (mscal (1./.6.)
(plus_vect !k1 (plus_vect !k4 (mscal 2. (plus_vect !k3 !k2)))));
        for k=0 to n-1 do
          set_color pendule.(k).couleur;
          centre1:=((int_of_float (100.*.(sin y.(!t mod
2).(2*k))))+(!centre1));
          centre2:=(-(int_of_float (100.*.(cos y.(!t mod
2).(2*k))))+(!centre2));
          plot !centre1 !centre2;
        done;
        incr t
     done;
     match ((em m q y.(!t mod 2))-.emi) with
        |a when a>marge*.emi->("em>emi",(float_of_int !t)*.pas)
        |b when b<(-.marge*.emi)->("em<emi",(float_of_int !t)*.pas)
        |_->("Réussi",(float_of_int !t)*.pas);;

let resolution_trajonly_em_ci_rk pas nbit q pendules m marge =
rktrajonly_em pas nbit (create_f (vect_length pendules) q) pendules m q
marge;;
```