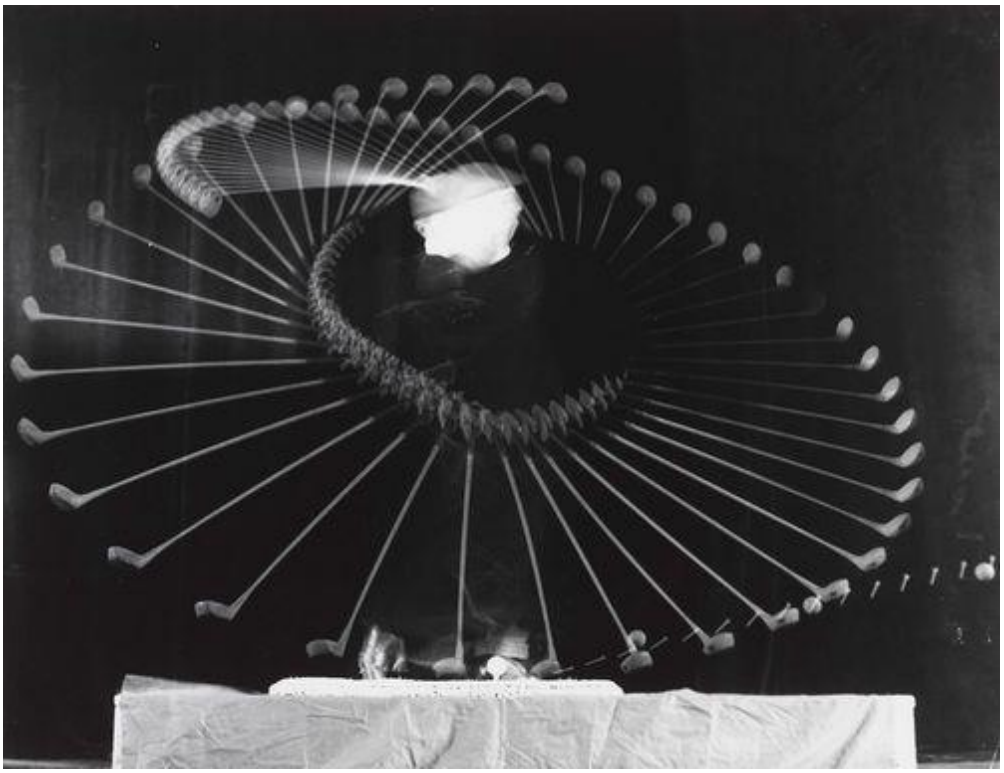
Daniel Yaacoub - 21605595
Angélo Charry - 21500316

L3 Physique
TP A11

COMPTE-RENDU

Projet Numérique



daniel.yaacoub@univ-tlse3.fr
angelo.charry@univ-tlse3.fr

UNIVERSITÉ TOULOUSE III PAUL SABATIER
Université de Toulouse

Année universitaire 2018-2019

Photographie de couverture : Harold Edgerton
https://www.pinterest.fr/pin/127648970660496404/

Daniel Yaacoub - 21605595
Angelo Charry - 21500316

L3 Physique
TP A11

Mouvement d'une balle de golf

Table des matières

I

Dynamique de vol, rebonds, glissements et simulations numériques

1. Modèle théorique initial – Forces en jeu

Nous nous proposons, en première modélisation, d'assimiler la balle de golf à un point matériel soumis à son poids, la résistance de l'air et la force de *Magnus* [1] et nous restreignons au vol de cette balle dans une portion d'atmosphère.

1.1. Force de pesanteur, approximations relatives, justifications

1.1.1. Force de pesanteur – approximations relatives

Nous identifions donc, durant son vol, la balle de golf à son centre de masse et lui attribuons la valeur de la masse de la balle. La force de pesanteur exprimée dans la base cartésienne rattachée au référentiel $(O, \hat{x}, \hat{y}, \hat{z})$ est :

$$\overrightarrow{f_p} = \begin{vmatrix} 0 \\ 0 \\ -mg \end{vmatrix}$$

Où g désigne la norme de l'accélération de la pesanteur : $g \cong 9{,}8\, m.s^{-2}$. On choisit dans la suite une balle de masse $= 0.045\ kg$.

1.1.2. Justifications pour le golf

Pour un objet matériel spatialement étendu de distribution de masse homogène, à symétrie sphérique, considérer le mouvement de son centre de masse auquel est attribuée la valeur de la masse de l'objet n'influe nullement sur les trajectoires, s'agissant ici d'une simple chute libre.

[1] Ou Robin

1.2. Force de trainée, approximations relatives

Bien qu'intéressés par le mouvement du centre de masse de la balle de golf, nous ajoutons la force de trainée ou résistance de l'air au bilan des forces, feignant de considérer la surface de la balle en interaction avec les molécules d'air. Nous considérerons une densité volumique de masse de l'air constante *i.e.* aucun changement barométrique ou thermique de la couche atmosphérique considérée. L'information sur la forme de la surface - et donc de la balle - étant incluse dans le choix du coefficient de trainée. On choisit un coefficient de trainée de l'ordre de 2.10^{-1}. La densité de l'air est prise constante égale à un.

$$\overrightarrow{f_a} = -\frac{1}{2}\rho C S \begin{vmatrix} (v_{x,b} - v_{x,v})^2 \\ (v_{y,b} - v_{y,v})^2 \\ (v_{z,b} - v_{z,v})^2 \end{vmatrix}$$

La composante suivant l'axe des z caractérise la portance et les deux autres, la trainée. L'indice « b » caractérise la balle et l'indice « v », le vent. On choisit enfin un rayon de balle de l'ordre de 42 mm.

1.3. Force de *Magnus*

L'effet Magnus est dû à la rotation de la balle. Cette rotation induit un découlement de la couche d'air l'enveloppant. La présence d'alvéoles sur la surface de la balle implique une diminution drastique du coefficient de la force de Magnus : il passe d'un ordre 10^{-6} à 10^{-4} ; en effet, la couche limite de Prandtl se décolle à l'arrière de la balle par rapport au sens de vol.

$$\overrightarrow{f_m} = k\,\vec{\Omega} \wedge (\overrightarrow{v_b} - \overrightarrow{v_v})$$

2. Première simulation : Dynamique de vol

Ayant présenté le premier modèle, les forces en jeu et les approximations pour l'instant établies, il s'agit désormais d'écrire le premier programme, d'établir et de résoudre les équations du mouvement et enfin d'obtenir et de discuter les premiers résultats numériques concernant le simple vol de la balle de golf.

2.1. Equation différentielle – `ode45`

Découle de la première partie et de la loi fondamentale de la dynamique, l'équation différentielle du mouvement de la balle de golf de masse m, repérée par ses coordonnées cartésiennes (x, y, z) :

$$\begin{pmatrix} \ddot{x} \\ \ddot{y} \\ \ddot{z} \end{pmatrix} = \begin{pmatrix} -\frac{1}{2m} sgn(\dot{x}) \rho C \pi R^2 (\dot{x} - v_{x,v})^2 - \frac{k}{m}(\Omega_{0y}\dot{z} - \Omega_{0z}\dot{y}) \\ -\frac{1}{2m} sgn(\dot{y}) \rho C \pi R^2 (\dot{y} - v_{y,v})^2 - \frac{k}{m}(\Omega_{0z}\dot{x} - \Omega_{0x}\dot{z}) \\ -\frac{1}{2m} sgn(\dot{z}) \rho C \pi R^2 (\dot{z} - v_{z,v})^2 - \frac{k}{m}(\Omega_{0x}\dot{y} - \Omega_{0y}\dot{x}) - g \end{pmatrix}$$

Soit une équation différentielle ordinaire non linéaire d'ordre deux en t que l'on peut écrire sous la forme :

$$\frac{d^2Y}{dt^2} = F\left(t, Y, \frac{dY}{dt}\right)$$

Ou encore sous la forme

$$\frac{dY}{dt} = F(t, Y)$$

en posant

$$Y(t) = (x, y, z, \dot{x}, \dot{y}, \dot{z})$$

et

$$F(t,Y) = \begin{pmatrix} Y(3) \\ Y(4) \\ Y(5) \\ -\frac{1}{2m} sgn\left(\frac{dY(1)}{dt}\right) \rho C \pi R^2 \left(\frac{dY(1)}{dt} - v_{x,v}\right)^2 - \frac{k}{m}\left(\Omega_{0y}\frac{dY(3)}{dt} - \Omega_{0z}\frac{dY(2)}{dt}\right) \\ -\frac{1}{2m} sgn\left(\frac{dY(2)}{dt}\right) \rho C \pi R^2 \left(\frac{dY(2)}{dt} - v_{y,v}\right)^2 - \frac{k}{m}\left(\Omega_{0z}\frac{dY(1)}{dt} - \Omega_{0x}\frac{dY(3)}{dt}\right) \\ -\frac{1}{2m} sgn\left(\frac{dY(3)}{dt}\right) \rho C \pi R^2 \left(\frac{dY(3)}{dt} - v_{z,v}\right)^2 - \frac{k}{m}\left(\Omega_{0x}\frac{dY(2)}{dt} - \Omega_{0y}\frac{dY(1)}{dt}\right) - g \end{pmatrix}$$

forme dans laquelle nous pouvons utiliser un `solver` pour la résolution. Un `solver` permet d'intégrer un système d'équations différentielles d'ordres un. Il retourne un vecteur colonne t constitué des dates des instants d'intégrations et une matrice Y dont la première colonne représente les solutions $Y(1)$ calculées à ces instants, la deuxième, les $Y(2)$, …

On utilise le `solver ode45` qui peut être vu comme un Runge-Kutta d'ordre 4. Il suffit alors de résoudre les équations du mouvement sur la fenêtre temporelle $[t_0, t_f]$:

```
CI = [P.Xo;Yo;Zo;Vox;Voy;Voz];
F = @(t,Y) [Y(4);Y(5);Y(6);-sign(Y(4))*(D/m)*(Y(4)-
Vwx)^2+(Cm/m)*(Woy*Y(6)-Woz*Y(5));-sign(Y(5))*(D/m)*(Y(5)
Vwy)^2+(Cm/m)*(Woz*Y(4)-Wox*Y(6));-sign(Y(6))*(D/m)*(Y(6)-Vwz)^2-
g+(Cm/m)*(Wox*Y(5)-Woy*Y(4))];

[t,Y] = ode45(F,[to,tf],CI);
```

2.2. Résultats numériques

2.2.1. `Plot` trajectoire

On adapte l'affichage du terrain à l'étendue des valeur prise par les *Y(1)* et *Y(2) i.e.* les *(x,y)* de la balle de golf : Pour ce faire, nous construisons trois vecteurs : deux vecteurs `terrain_x` et `terrain_y` délimitant l'intervalle que l'on souhaite afficher et un vecteur terrain_z associant à ces deux vecteur la hauteur de la surface définie par la fonction `terrain`. On génère donc le vecteur `terrain_x` entre la valeur minimale et maximale du vecteur `Y( :,1)` (la position de la balle en x) ; pour plus de confort, on pourra élargir cet intervalle. On fait de même pour `terrain_y`. Il ne reste plus qu'à définir `terrain_z` en calculant la fonction `terrain` pour les deux vecteurs précédemment calculé puis d'utiliser la fonction `mesh` dans notre boucle pour afficher la surface.

```
% Terrain
terrain_x = min(Y(:,1))-1:0.2:max(Y(:,1))+1 ;
terrain_y = (min(Y(:,2))-1:0.2:max(Y(:,2))+1)';
terrain_z =  terrain(terrain_x,terrain_y);
C = terrain_z;
mesh(terrain_x,terrain_y,terrain_z)

% Trajectoire
hold on
plot3(Y(:,1),Y(:,2),Y(:,3),'b')
title('trajectoire')
xlabel('x (m)')
ylabel('y (m)')
zlabel('z (m)')
grid ON
```

2.2.2. `Plot` énergétique

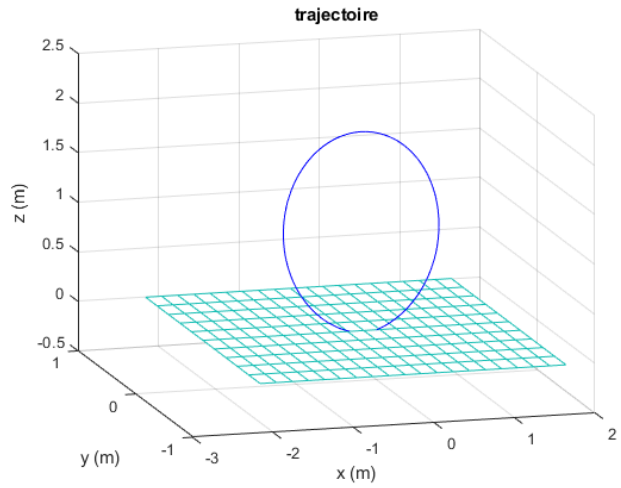
Ce plot servira à vérifier la validité de notre démarche et présente donc un grand interet pour le débogage.

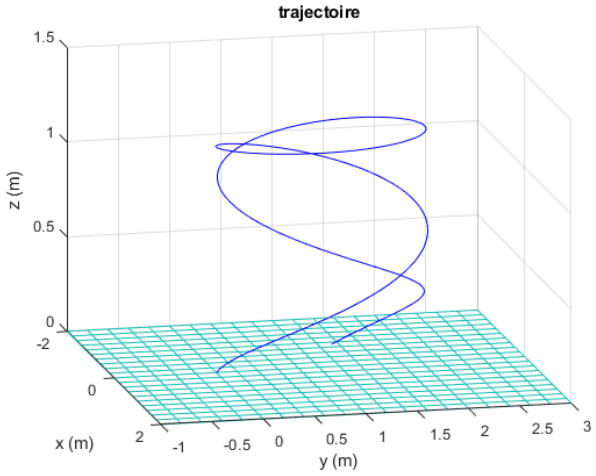
```
% Energétique
Ec = 0.5*m*(Y(:,4).^2+Y(:,5).^2+Y(:,6).^2);
Ep = m*g*Y(:,3);
E = Ec+Ep;

figure(2)
plot(t,Ec,'r',t,Ep,'b',t,E,'k')
```

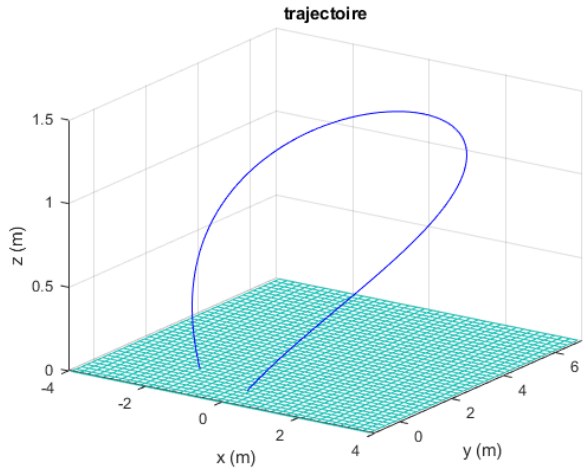
```
title('Energie cinétique-rouge | Energie potentielle-bleue |
Energie mécanique-noire)')
xlabel('Temps (s)')
ylabel('Energie (j)')
```

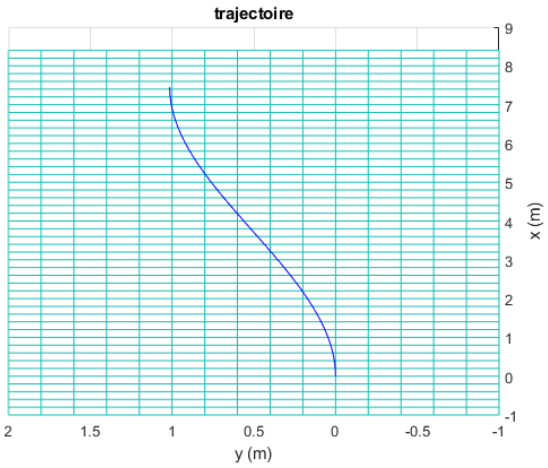
2.2.3. Hook, slice, backspin et compagnie

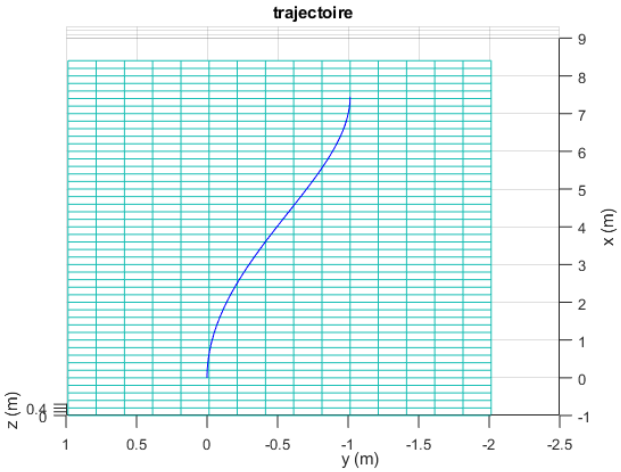
Les effets discutés dans cette section dérivent tous de la valeur et de l'orientation du vecteur rotation de la balle de golf. Les trajectoires présentées ci-dessous sont toute issues de même condition initiales à l'exception de la valeur des composantes du vecteur rotation. Seule une composante de celui-ci est non nulle pour chacune de ces trajectoires. La trajectoire du coin haut-gauche illustre parfaitement le backspin. La balle possède une rotation telle, qu'elle revient sur ses pas pour $\Omega_y = -150\, rad.s^{-1}$, celle-ci revient exactement à son point de lancer pour de bonnes conditions initiales. Les deux dernières trajectoires illustrent quant à elles les effets de Hook et slice. Slice lorsque la trajectoire de la balle va exagérément de la gauche vers la droite et hook pour le contraire.








trajectoire
x (m)
y (m)
9
8
7
6
5
4
3
2
1
0
-1
2
1.5
1
0.5
0
-0.5
-1


trajectoire
x (m)
z (m)
y (m)
9
8
7
6
5
4
3
2
1
0
-1
0.4
1
0.5
0
-0.5
-1
-1.5
-2
-2.5

3. Deuxième simulation : Vol et rebonds sur un plan horizontal

On ajoute ici à notre description du mouvement, le phénomène de rebonds. Pour ce faire et pour, dans un premier temps, ne pas interférer avec les problématiques liées aux impacts sur un terrain courbe, on propose un terrain plan, horizontal, d'équation cartésienne :

$$z = 0$$

3.1. Description théorique du rebondissement

La balle rebondit des que sa trajectoire intersecte le plan horizontale *i.e.* dès que sa composante z égale zéro. Alors, à la position de l'impact, sa vitesse suivant l'axe des z subit une réflexion. Cette réflexion peut être affectée d'une perte *via* un coefficient de restitution[2] e caractérisant l'absorption d'énergie cinétique du au terrain ou encore à la déformation inélastique de la balle de golf.

$$e = \frac{vitesse\ avant\ collision}{vitesse\ après\ collision}$$

Un moyen de déterminer ce coefficient aurait été de mesurer la hauteur après rebond et la flèche de vol avant rebond. On prend la valeur du coefficient de restitution associé à un terrain de type pelouse : $e \approx 0.5$. L'impact se traduit donc par le changement :

$$\dot{z} \leftarrow -\frac{1}{2}\dot{z}$$

3.2. Algorithme

3.2.1. Evènement

Il s'agit, dans un premier temps, de caractériser algorithmiquement l'évènement (physique) du rebond. Aussi doit-on introduire un évènement dans le processus d'intégration, une condition sur l'arrêt de l'intégration. Quand doit-on arrêter l'intégration de l'équation différentielle ? Comment `ode45` peut-il arrêter la résolution si l'évènement apparait alors même que la borne supérieure de la fenêtre temporelle n'est pas atteinte ?

La condition d'arrêt d'intégration *i.e.* d'arrêt de résolution du vol de la balle de golf est la condition détaillée dans la section précédente.

L'arrêt de l'intégration alors même que la borne supérieure de la fenêtre temporelle n'est pas atteinte nécessite la création d'un évènement (au sens de `Matlab`). `ode45` permet l'utilisation de telles `options` nécessitant la simple création d'une fonction :

```
function [value,isterminal,direction] = Evenement_rebond(t,Y)

value = Y(3) > feval('terrain',Y(1),Y(2));
```

[2] Parfois appelé friction par différentiation au coefficient de restitution de rebond traduisant une simple situation de chute libre.

```
isterminal = 1;
direction = -1;
```

`value` indique la condition qui ne doit pas être satisfaite pour que l'intégration s'arrête. `isterminal = 1` indique le fait qu'il s'agit d'arrêter l'intégration et `direction = -1` précise que la condition est valable dans le sens des z(t) décroissants *i.e.* en tombant. La fonction `terrain` représente la surface

On définit enfin l'option via la fonction `odeset` :

```
options = odeset('Events', @Evenement_rebond);
```

L'évènement ainsi définit restera celui-ci dans la suite du compte-rendu et pour les programmes à venir.

3.2.2. Structure générale

Il s'agit alors de penser le processus précédant (un rebond précédé d'un vol) dans une boucle. Il doit idéalement être réitéré ***tant que*** la condition de « non-rebond » - qui fera l'objet de la section ***5.1.2.*** - n'est pas vérifiée. On l'insère cependant - pour des raisons de commodité de construction et de débogage - dans une boucle « ***pour*** ». L'entrée d'un paramètre sur le nombre maximum de rebonds à effectuer est pour l'heure le seul moyen d'arrêter la boucle. Cette solution étant *ad hoc* puisque nous savons qu'une condition d'arrêt physique, dépendant notamment de la vitesse de la balle lors de l'impact au sol, doit exister. Nous construisons donc pour l'instant, un programme dont le corps de la structure générale est le suivant:

Pour le nombre d'occurrence de l'évènement variant de 1 au nombre maximum de rebonds **faire** :

(i) Intégration du système :

$$\begin{cases} \frac{dY(t)}{dt} = F(Y,t) \\ conditions\ initiales\ Y(0) \\ fenêtre\ temporelle\ [t0\ ;\ tf] \\ évènement\_rebond \end{cases}$$

(ii) Changement des conditions initiales :

- les composantes de Y(0) **prennent les valeurs** des composantes de Y à l'instant de l'arrêt de l'intégration en conservant celles du vol.
- t_0 **prend la valeur** de l'instant de l'arrêt de l'intégration en conservant ceux du vol.
- z **prend la valeur** –e*z.

Si $t_f = t_0$ **alors** :
casser
Finsi

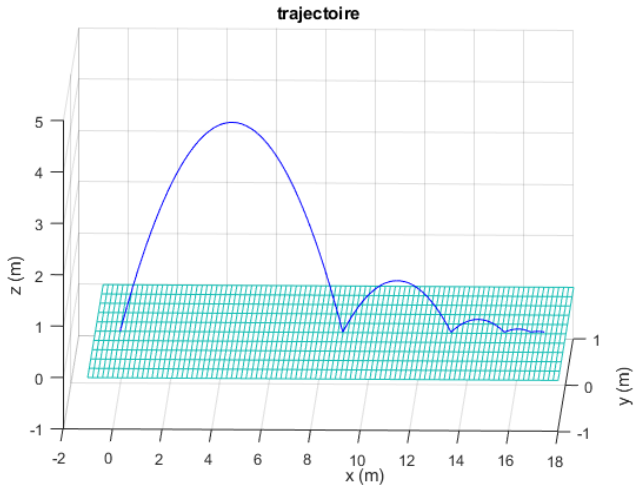
Finpour

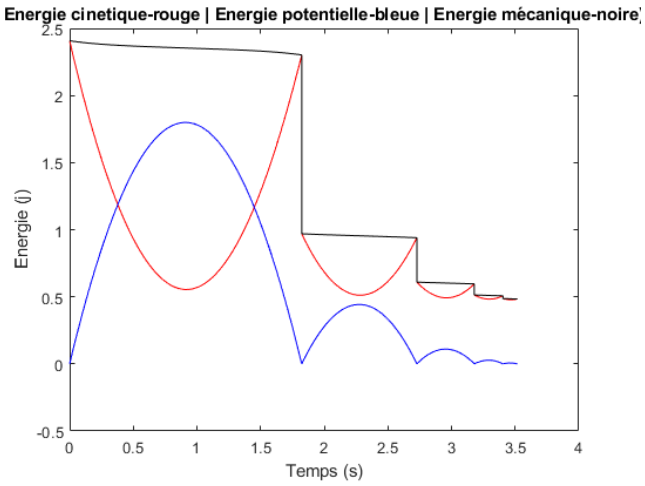
On peut également y ajouter une condition supplémentaire pour être sur que la balle ne crève pas la surface lors de l'impact, puisque l'évènement sur `ode45` ne vaut que lors du vol de celle-ci.

Si z ≤ terrain(x,y) **alors** :
z = terrain(x,y)
Finsi

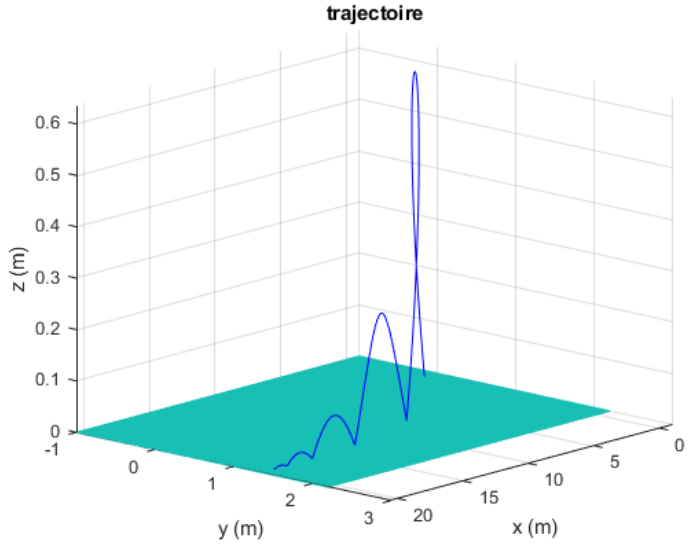
3.3. Résultats numériques

Le graphique énergétique présente une énergie mécanique constante par morceau et semble ainsi corroborer la validité de notre modèle de ce point de vue.





Les effets dus à la rotation sont également observables sur la trajectoire :



4. Troisième simulation : Vol et rebonds sur un terrain quelconque

La structure algorithmique du rebondissement ayant été présentée, nous décrivons désormais le phénomène physique sur une surface quelconque - plus proche de la réalité -. L'idée reste évidemment identique mais les éléments de topographie obligent une formulation théorique de l'impact plus générale que le changement $z \leftarrow -e * z$.

4.1. Description théorique du rebondissement

4.1.1. Normale à un surface courbe – base locale

Considérant la surface constituant le graphe d'une fonction de deux variables : $z = F(x, y)$, un paramétrage de celle-ci devient :

$$\vec{r}(x, y) = \begin{pmatrix} x \\ y \\ F(x, y) \end{pmatrix}$$

Le vecteur tangent à la courbe $\vec{r}(x, y_0)$ est alors

$$\left.\frac{\partial \vec{r}(x, y)}{\partial x}\right|_{(x,y_0)} = \begin{pmatrix} 1 \\ 0 \\ \left.\frac{\partial F}{\partial x}\right|_{(x,y_0)} \end{pmatrix}$$

tandis que celui tangent à la courbe $\vec{r}(x_0, y)$ est

$$\left.\frac{\partial \vec{r}(x, y)}{\partial y}\right|_{(x_0,y)} = \begin{pmatrix} 0 \\ 1 \\ \left.\frac{\partial F}{\partial y}\right|_{(x_0,y)} \end{pmatrix}$$

On construit donc le vecteur normal unitaire à la surface au point (x_0, y_0) comme suit :

$$\vec{n} = \frac{1}{\|\vec{n}\|} \left.\frac{\partial \vec{r}(x, y)}{\partial x}\right|_{(x,y_0)} \wedge \left.\frac{\partial \vec{r}(x, y)}{\partial y}\right|_{(x_0,y)} = \frac{1}{\|\vec{n}\|} \begin{pmatrix} -\left.\frac{\partial F}{\partial x}\right|_{(x,y_0)} \\ -\left.\frac{\partial F}{\partial y}\right|_{(x_0,y)} \\ 1 \end{pmatrix}$$

4.1.2. Impact

La philosophie reste la même : l'impact induit une réflexion de la composante de la vitesse selon la normale au plan tangent associé au point de collision. Il s'agit donc de construire le plan tangent et la normale (calcul détaillé en ***4.2.2.***), d'exprimer le vecteur vitesse dans cette base et d'obtenir la réflexion de la composante normale. Si l'on considère la base normale $\{n\} = \{nx, ny, nz\}$, le passage de la vitesse dans cette nouvelle base devient :

$$V\_n = \begin{pmatrix} n_{x,1} & n_{y,1} & n_{z,1} \\ n_{x,2} & n_{y,2} & n_{z,2} \\ n_{x,3} & n_{y,3} & n_{z,3} \end{pmatrix}^{-1} V$$

4.2. Algorithme

4.2.1. Structure générale

On conserve la boucle « pour » mais on ajoute ici un paramètre booléen (`rebond`) permettant de rendre compte d'un éventuel arrêt du processus de rebondissement. Il renseigne sur l'état de la balle : en train de rebondir ou non. La structure générale devient simplement :

- ***Initialisation***

rebond = 1

Pour le nombre d'occurrence de l'évènement variant de 1 au nombre maximum de rebonds **faire** :

(i) Intégration du système :

$$\begin{cases} \frac{d\text{Y\_vol(t)}}{dt\_vol} = \text{F(Y\_vol, t\_vol)} \\ conditions\ initiales\ \text{Y\_vol(0)} \\ fenêtre\ temporelle\ [\text{t0 ; tf}] \\ évènement\_rebond \end{cases}$$

(ii) Changement des conditions initiales :

- les composantes de Y_vol(0) **prennent les valeurs** des composantes de Y_vol à l'instant de l'arrêt de l'intégration et conserve celles du vol.
- Les composantes de Y_vol(0) **prennent les valeurs** des six premières composantes de **impacte**(Y_vol(0), terrain).
- Rebond **prend la valeur** de la septième composante de **impacte**(Y_vol(0), terrain).
- t_0 **prend la valeur** de l'instant de l'arrêt de l'intégration et conserve ceux du vol.

Si $t_f = t_0$ **alors** :

casser

Finsi

Si rebond = 0 **alors** :

casser

Finsi

Finpour

4.2.2. Impact

On introduit la fonction `impact` qui servira à éclaircir la structure générale du script principal dans les sections suivantes. Cette fonction prend en entrées la position et la vitesse de la balle dans la base cartésienne et renvoie sa position, sa vitesse et la valeur du booléen `rebond`. La structure de la fonction impact est, comme le suggère la section ***4.1.2.,***

```
function I = impacte(CI,F)

X = CI ;
Vlimite = 0.2;
```

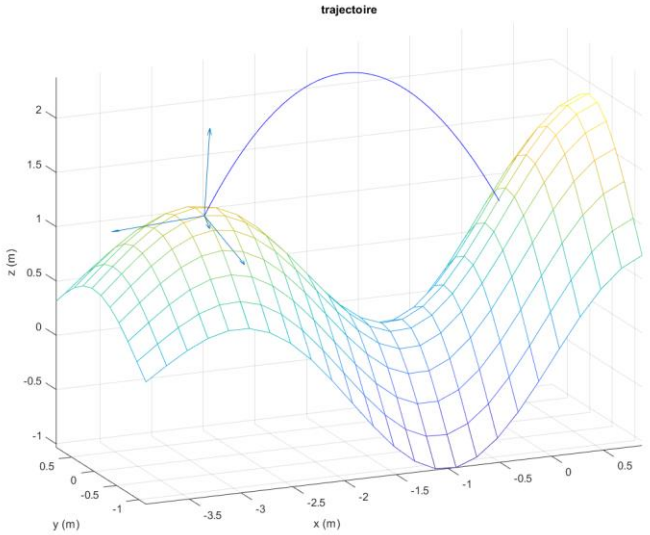
```
% création d'une base orthonormale locale en un point d'impact de
la surface et matrice de passage :

nz= normale(X(1),X(2),F) ;
nx = [-nz(3),0,nz(1)];
ny = cross(nz,nx);
nx = nx./norm(nx) ;
ny = ny./norm(ny);
Mat_n_e = [nx',ny',nz'];

% réflexion dans dans la base orthonormale :

V = [X(4),X(5),X(6)]' ;
V_n = Mat_n_e\V   ;
V_n(3) = -e*V_n(3) ;
V_e = Mat_n_e*V_n ;

% arret des rebonds :

if V_n(3) < Vlimite
      rebond = 0 ;
      V_n(3) = 0 ;
else
      rebond = 1 ;
end

X(4) = V_e(1) ;
X(5) = V_e(2) ;
X(6) = V_e(3) ;
I = [X,rebond];
```

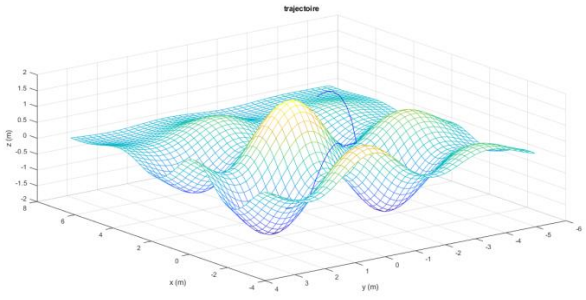
où le paramètre `rebond` permet l'arrêt es rebonds si la vitesse en fin de vol n'est pas assez suffisante et où l'on définit la fonction normale qui, conformément aux calculs de la section ***4.1.1.,*** renvoie les composante de la normale à la surface en un point d'entrée :

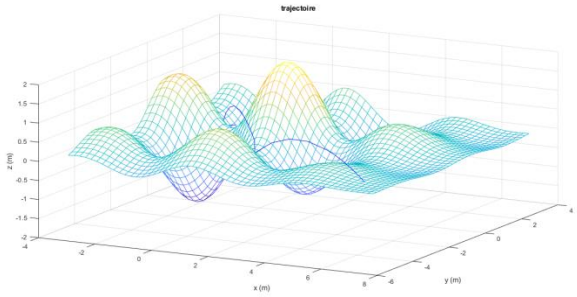
```
function n = normale(x0,y0,F)
h = 0.0001 ;
fgradx = (feval(F,x0+h,y0) - feval(F,x0,y0))/h;
fgrady = (feval(F,x0,y0+h) - feval(F,x0,y0))/h;
n = [-fgradx,-fgrady,1] ;
n = n./norm(n);
end
```

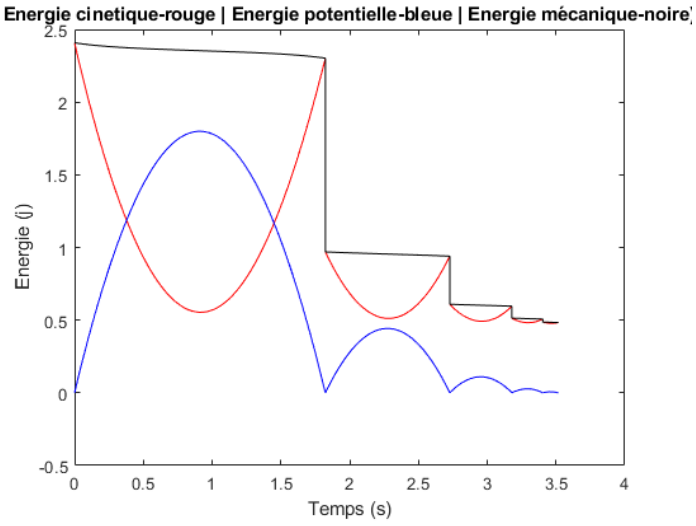
4.3. Résultats numérique

La base normale orthonormée est correcte et les rebonds qui en découlent ne semblent pas aberrants. L'énergétique est, de plus, conforme aux attentes. On reporte ci-dessous quelques graphiques illustrant le modèle :









4.4. Animation des résultats numérique

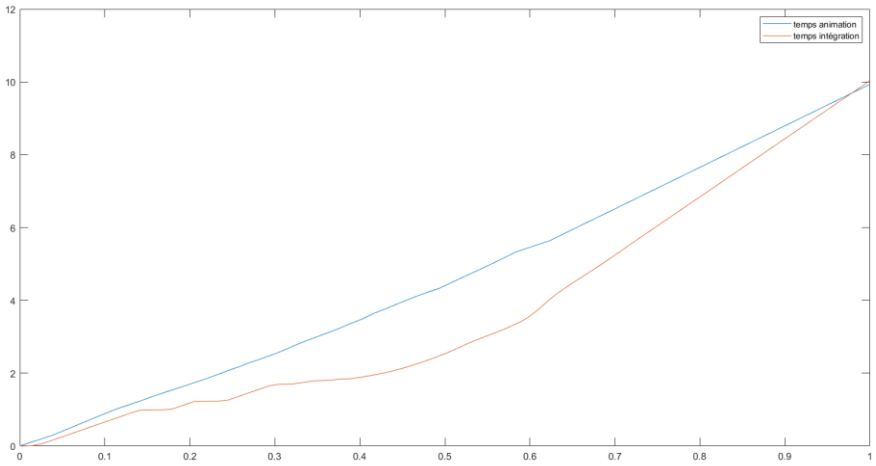
Nous disposons donc d'un tableau `Y` contenant toutes les positions et toutes les vitesses de la balle à chaque instant. Le temps associé à chacune de ces valeurs est contenu dans le vecteur `t` les deux tableaux ont donc la même longueur. Pour construire une animation il nous suffit d'utiliser une boucle *for* :

pour *i* allant de *1* à taille de *t* faire :

tracer les position de *1* à *i*
afficher la surface
finpour

Perciste toutefois un problème : notre simulation utilise un pas de temps adaptatif pour plus de précision (typiquement celui de `ode45`). Aussi, certaines sections de la trajectoire comportent plus de points que d'autres. Nous accordons cependant autant de temps à chaque point dans notre boucle précédente et biaisons ainsi la notion de vitesse réelle de la balle lors de la reconstitution que constitue notre animation. Pour pallier ce problème, nous n'affichons pas tous les points : On mesure l'intervalle de temps *dt* entre le point associé à l'itération courante et le dernier point affiché. Si cet intervalle est plus petit qu'une certaine valeur, on n'affiche pas le point.

Bien qu'imparfaite, cette méthode permet tout de même de s'approcher d'une perception linéaire du temps sans perte de précision dans la méthode d'intégration.




```
to =t(1);
dto = 0.01 ;
figure('Position',[0 0 1550 800])

for i = 1:length(Y(:,1))
    dt = t(i)-to;
    if dt >= dto
        mesh(terrain_x,terrain_y,terrain_z)
        axis equal
        hold on
```

```
plot3(Y(1:i,1),Y(1:i,2),Y(1:i,3),'b',Y(i,1),Y(i,2),Y(i,3),'-ob')
        view(-30,52)
        title('trajectoire')
        title(sprintf('t=%f',t(i)))
        xlabel('x (m)')
        ylabel('y (m)')
        zlabel('z (m)')
        grid ON
        hold off

        to = t(i);
    end
    pause(0.00001)
end
```

`view` permet de contrôler la position de la caméra qui ne peut être modifiée pendant l'animation. En coordonnée sphérique, la première valeur correspond à l'angle phi, et la deuxième à l'angle thêta .

N.B : La technique restera la même pour pallier les effets du pas adaptatif introduit dans la section suivante sur le glissement.

5. Quatrième simulation : Glissement sur une surface

Nous avions précédemment usé d'un artefact pour palier la structure en boucle « *pour* » du programme et arrêter le rebondissement. Nous développons ici, dans les approximations dont il sera question, les véritables conditions d'arrêt de rebonds et nous décrivons la phase qui s'ensuit : le glissement. L'articulation des deux se faisant dans la section ***6.*** Avec la structure finale du programme relatif à la simulation.

5.1. Description théorique

5.1.1. Glissement simple

L'étude du glissement sur une surface courbe peut se ramener, pour des déplacements $\delta\vec{r} = \vec{r}(t+dt) - \vec{r}(t)$ en $dt \ll 1$, à l'étude du glissement sur un plan incliné par rapport à l'horizontale *i.e.* par rapport au plan (xOy) de la base cartésienne, d'un angle α. La surface courbe est alors localement plane et il suffit de décrire le mouvement dans la base locale normale constituée du plan local tangent et de la normale locale. On établit les équations du mouvement dans la base cartésienne et on pose le bilan des forces suivant :

$$\vec{P} = mg\begin{pmatrix} 0 \\ 0 \\ -1 \end{pmatrix}$$
$$\vec{R} = mg\cos(\alpha)\vec{n}$$
$$\vec{f} = -\mu\vec{v}$$

où $\alpha = \cos^{-1}(-\vec{P}.\vec{n})$
Il vient simplement :

$$\vec{v} = \int \frac{1}{m}\left(\vec{P} + \vec{R} + \vec{f}\right) dt$$
$$\vec{r} = \int \vec{v}\, dt$$

5.1.2. Transition rebonds-glissement, approximations relatives

Lors d'un « vrai » contact, soit la balle de golf glisse sur la surface *i.e.* elle frotte, soit elle roule *i.e.* elle accroche la surface soit elle glisse puis roule. On approxime l'étude au cas d'une balle qui ne peut que glisser c'est-à-dire que l'on suppose qu'au moment du contact, la balle tourne trop vite pour accrocher le support. Cette condition se traduit par une vitesse à l'équateur plus importante que la vitesse du centre de masse de la balle : $R\left\|\vec{\Omega}\right\| \gg \|\vec{v}\|$. On note qu'une balle pourrait très bien redécoller et rebondir à nouveau après une phase de glissement puis de roulement dans le cas où la barrière de potentiel (petit vallon sur le terrain) serait aisément franchissable. On se restreint au seul cas {glissement systématique + arrêt} précédé de rebonds et on développe dans cette partie la description du glissement.

5.2. Algorithme

5.2.1. Structure générale

(i) *Calcul des éléments de la base normale orthonormée au point d'impact.*

(ii) *Ecriture de la vitesse au moment de l'arrêt de l'intégration du dernier rebond (dernier vol) dans la base normale : V_n pour supprimer sa composante V_n(3).*

(iii) *Ecriture de cette dernière dans la base cartésienne fixe :V_e.*

(iv) $V\_e \leftarrow V\_e + a.dt$

$x \leftarrow x + V\_e(1).dt$

$y \leftarrow y + V\_e(2).dt$

$z \leftarrow z + V\_e(3).dt$

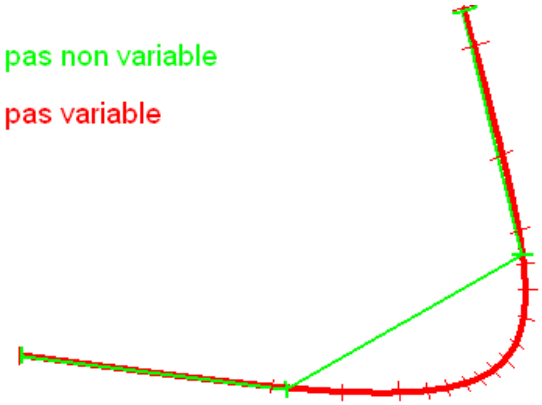
On construit donc la fonction glissement prenant en entrée les coordonnées cartésienne de la balle de golf, les composantes de sa vitesse dans la base cartésienne et le pas temporel d'intégration pour retourner les coordonnées de la balle dans la base cartésienne et les composantes de la vitesse dans la base cartésienne :

```
function Y = glissement(r,V_e,h,P)

x = r(1);
y = r(2);
z = r(3);
g=P.gravite ;
m=P.masse ;
mu = P.frottement;
dt = h ;

% Construction de la nase locale normale orthonormée {n}
gradx = (terrain(x+h,y) - terrain(x,y))/h;
grady = (terrain(x,y+h) - terrain(x,y))/h;
t1 = [1,0,gradx];
t2 = [0,1,grady];
n = cross(t1,t2);
nz = n./norm(n);
nx = [-nz(3),0,nz(1)];
ny = cross(nz,nx);
nx = nx./norm(nx);
ny = ny./norm(ny);

% Matrice de la base {n} dans {e}
Mat_n_e = [nx',ny',nz'] ;

% Vitesse dans la base {n} (on colle la surface donc la composante
suivant la normale est nulle)
V_n = Mat_n_e\V_e ;
V_n(3)=0;

% Vitesse dans la base {e}
V_e = Mat_n_e*V_n;

% Intégration
Poid = m*g*[0;0;-1];
alpha = acos(dot([0;0;1],nz'));
Resistance = m*g*cos(alpha)*nz';
frottement = -mu*V ;
```

```
a = (Resistance+Poid+frottement)/m;

V_e = V_e + a*dt;
x = x + V_e(1)*dt ;
y = y + V_e(2)*dt ;
z = z + V_e(3)*dt ;

% Condition pour ne pas crever la surface
if z< terrain(x,y)
    z=terrain(x,y);
end

Y = [x,y,z,V_e(1),V_e(2),V_e(3)];
```

5.2.2. Adaptation du pas

Pour plus de précision nous avons décidé d'adapter le pas d'intégration au fur et à mesure. Cela permet d'être plus précis quand cela est nécessaire et plus rapide lorsqu'on peut se permettre de faire de plus grands pas. L'algorithme compare son erreur d'intégration à une valeur de tolérance définie par l'utilisateur et gagne ainsi en précision.



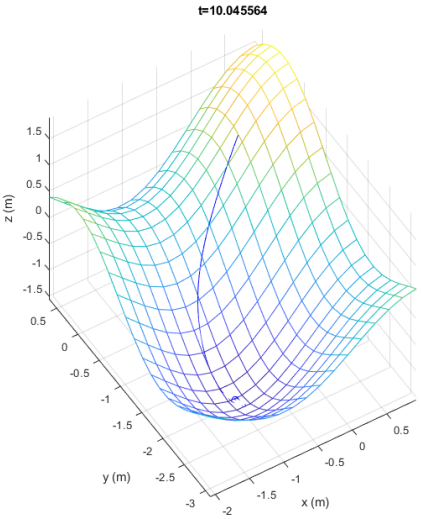
La fonction `glissement` détermine le point suivant ainsi que l'évolution de la vitesse à l'aide d'une méthode d'intégration d'ordre 1. On lui fournit la position et la vitesse à *t* ainsi qu'un pas *h* et la fonction nous renvoie la position et la vitesse en *t+h*.

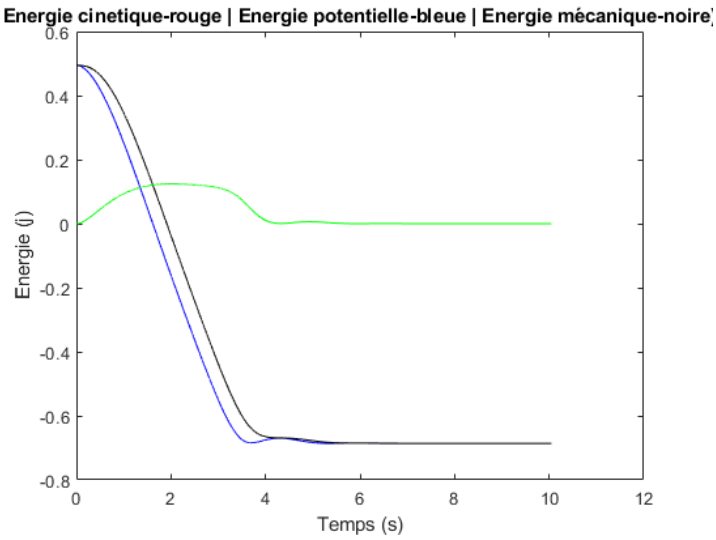
Pour mettre en place l'adaptation du pas *h* on commence par initialiser la valeur de *h* et déterminer la valeur de tolérance. On demande ensuite à notre fonction `glissement` de déterminer, d'une part la position et la vitesse au temps *t+h* et, indépendamment, la position et la vitesse au temps *t+h/2* puis à partir de ce résultat on détermine la position et la vitesse au temps *t+h*. Nous sommes donc en possession de deux jeux de coordonnées l'un calculé avec un pas *h*, l'autre avec un pas h/2 qui sera donc forcément plus précis. On mesure l'erreur d'intégration sur la positon et on la compare avec la tolérance précédemment définie. *Tant que* l'erreur est plus grande que la tolérance on répète ces étapes à l'aide d'une boucle « *while* ». Cela fonctionne car avant de revenir au début de la boucle nous modifions le pas avec la formule *h = 0.9*h*sqrt(tolérance/erreur).* Cela a pour conséquence de réduire le pas tant que l'erreur est supérieure à la tolérance. A l'inverse si l'erreur est plus petite que la tolérance on accepte les coordonnées calculées, on augmente le pas pour gagner en vitesse de calcul et on sort de la boucle « *while* ». Bien-entendu, cette boucle est elle-même comprise

dans une boucle « *while* » qui permet de continuer l'intégration tant que $t < tf$ où tf est la borne supérieur de la fenêtre temporelle rentrée par l'utilisateur.

5.3. Résultats numérique

On présente une trajectoire de glissement simple suivie de son `plot` énergétique :





6. Simulation finale : Vol, rebonds et glissement sur un terrain quelconque – Conclusion partielle

6.1. Vol, rebonds et glissement sur un terrain quelconque

6.1.1. Structure générale

On remplace la boucle « *pour* » des rebonds par une véritable boucle « *tant que* », on insère une méthode d'adaptation du pas dans l'intégration du glissement et l'on anime le tracé de la trajectoire en conservant l'échelle de temps réelle de l'évènement physique (chronophotographie). Le changement « *pour* » ← « *tant que* » permet de ne pas casser la boucle comme nous le faisions dans la boucle « *pour* » si rebond valait zéro. Voici le corps de l'algorithme :

- ***Initialisation rebonds***
  rebond = 1 + Conditions initiales

  Tant que rebond = 1 **faire:**

  (i) Intégration du système :

$$\begin{cases} \frac{d\text{Y\_vol(t)}}{dt\_vol} = \text{F(Y\_vol, t\_vol)} \\ conditions\ initiales\ \text{Y\_vol(0)} \\ fenêtre\ temporelle\ [\text{t0 ; tf}] \\ évènement\_rebond \end{cases}$$

  (ii) Changement des conditions initiales :
  - les composantes de Y_vol(0) **prennent les valeurs** des composantes de Y_vol à l'instant de l'arrêt de l'intégration et conserve celles du vol.
  - Les composantes de Y_vol(0) **prennent les valeurs** des six premières composantes de **impacte**(Y_vol(0), terrain).
  - rebond **prend la valeur** de la septième composante de **impacte**(Y_vol(0), terrain).
  - t_0 **prend la valeur** de l'instant de l'arrêt de l'intégration et conserve ceux du vol.

  Si $t_f = t_0$ **alors** :

  casser

  Finsi

  Fintantque

- ***Initialisation rebonds***
  Pas d'intégration + le temps de glissement prend la valeur de la dernière composante du vecteur temps *t* + Conditions initiales

  Tant que le temps de glissement est inférieur à la borne supérieure *tf* de la fenêtre globale **faire :**

  (iii) Adaptation du pas sur l'intégration du système :

Integration = 1 + tolérance

Tant que integration = 1 **faire** :

- r1 **prend la valeur** de glissement pour les valeurs des conditions initiales et un pas h comme entrées.
- r2 **prend la valeur** de glissement pour les valeurs des conditions initiales et un pas h/2 comme entrées.
- r3 **prend la valeur** de glissement pour les valeurs de r2 et un pas h/2 comme entrées.
- erreur **prend la valeur** de la distance entre r1 et r2.
- h **prend la valeur** 0.9*h*sqrt(tolérance/erreur).

Si l'erreur est plus petite que la tolérance **alors** :

- les conditions initiales **prennent les valeurs** des composantes de r2.
- On casse.

Finsi

Fintantque

- On ajoute h au temps de glissement.
- On concatène la valeur du temps de glissement aux valeurs de t.
- On concatène les valeurs des conditions initiales aux valeurs des positions et trajectoires stockées depuis le lancé de la balle.

Si la valeur de la vitesse est plus petite que la tolérance **faire** :

Casser

Finsi

Fintantque

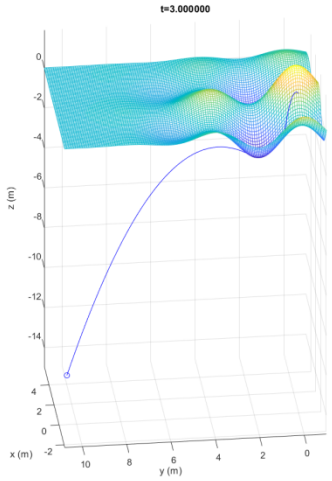
(iv) Animation de la trajectoire

(v) Garph énergétique

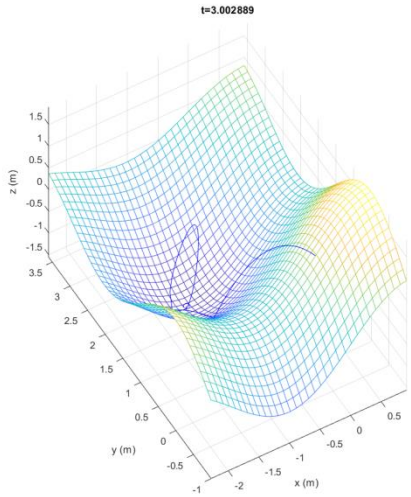
Afin de facilité la lecture et l'utilisation du programme, notamment pour la partie **II.**, nous faisons appel à des fonctions externes à la structure générale détaillée ci-dessus. Les paramètres sont, par exemple, à instruire par l'utilisateur en dehors du programme principal.

6.1.2. Importance de la tolérance

De même qu'il nous a été possible d'introduire une option évènement dans `ode45`, nous ajoutons une option de tolérance fixant un pas maximum entre deux points d'intégration. Sans option de tolérance :



tandis qu'en ajoutant une option de tolérance dans `ode45`, ces phénomènes n'apparaissent plus :

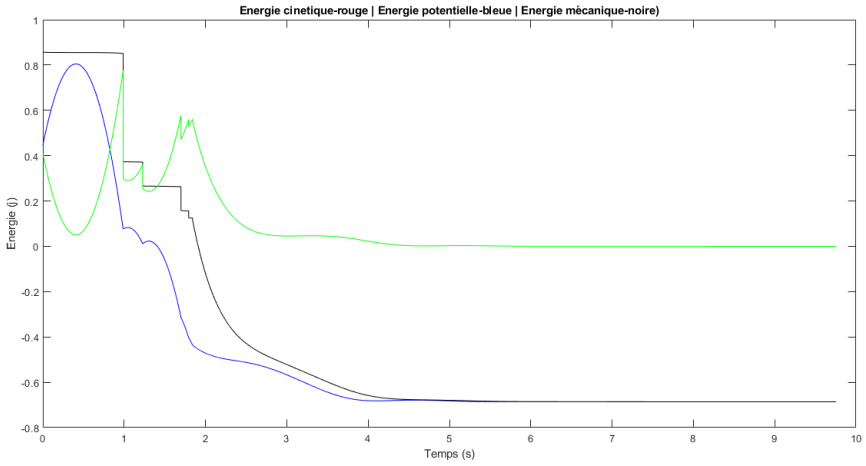


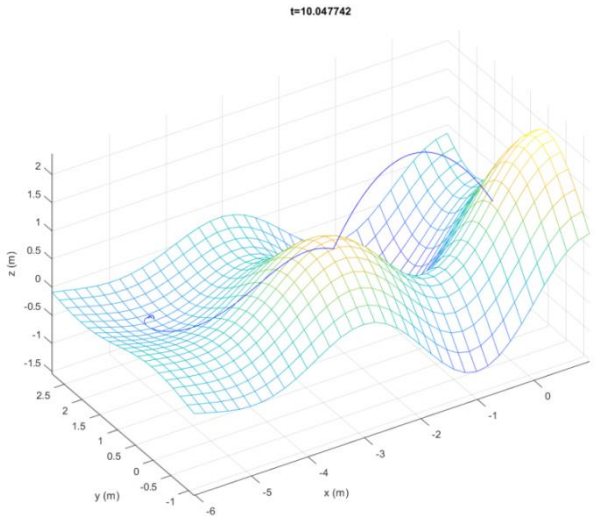
In fine,

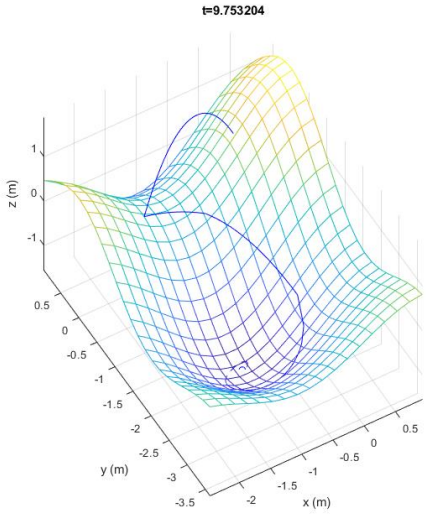
```
options = odeset(  'RelTol',1e-6,...
                   'AbsTol', 1e-6,...,
                   'Events', @bounceEvents);
```

6.1.3. Résultats numériques

Voici les résultats numériques du dernier degré de simulation (quelques animations figurent également dans le mail) :







6.2. Conclusion partielle

Le programme construit semble proche des comportements observables mais n'est cependant pas physiquement proximal. Nous rappelons donc - sans ordonnancement particulier - les hypothèses majeures faites consciemment et les approximations jusqu'ici égrenées dans le compte-rendu :

- Le coefficient de restitution est constant *i.e.* on postule un terrain homogène fait d'un même matériau. Ce coefficient n'est, de surcroit, pas réellement explicité dans la littérature tandis que celui du club contre la balle l'est plus. Enfin, nous n'avons pas pris en compte le coefficient de restitution du club contre la balle.

- La vitesse du vent est constante durant le tir et sur tout le terrain. On évince la possibilité de toute rafale ou évènement météorologique inopiné.

- La rotation de la balle est postulée constante durant tout le processus alors qu'elle doit bien-entendu être modifiée lors des impacts et devrait affecter les phases de roulement (et donc de glissement).

- Une phase de rebonds serait tout à fait envisageable après une phase de glissement. En effet, la balle pourrait redécoller. Nous ne l'avons pas considérer, de même que la distinction roulement-glissement qui s'imposerait.

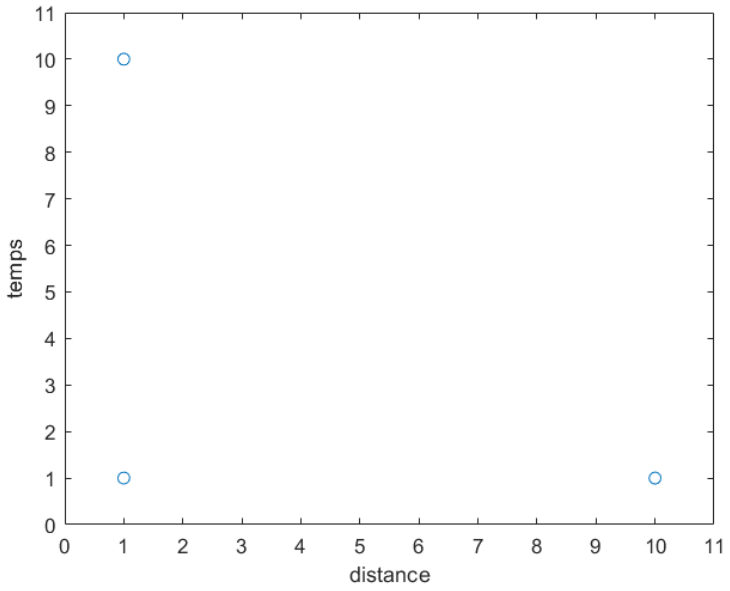
II

Optimisation

7. Positionnement général

Se pose, sans cette partie, le problème suivant : Est-il possible de construire un algorithme capable de trouver, pour une position de départ et d'arrivée données, les conditions initiales optimales ? Qu'optimiser ? La première étape consiste donc à déterminer quel sera la fonction à optimiser (minimiser). Nous transformons, pour cela, notre modélisation en une fonction `simulation` prenant en entrées les trois vitesses initiales Vx, Vy et Vz sous contrainte de la position initiale de la balle de golf et renvoyant la trajectoire, la vitesse et le temps (On rajoute le vecteur colonne temps en première position).

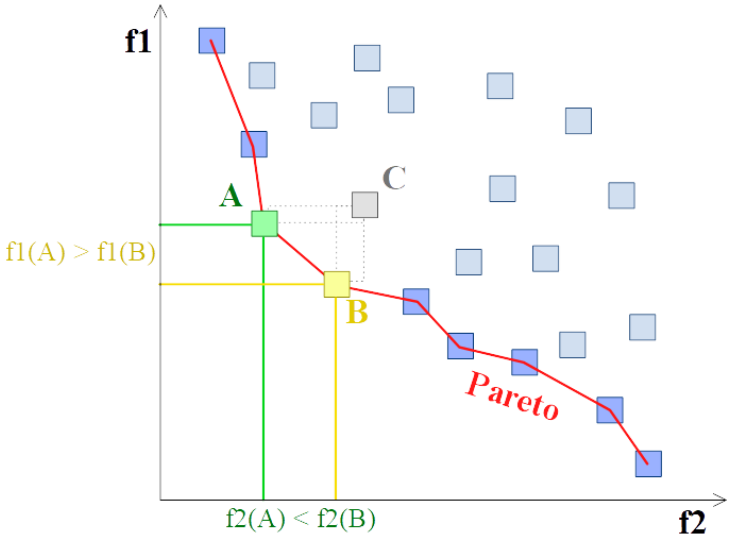
Nous évaluerons chaque lancé à l'aide d'une fonction `score` (`fitness` dans la littérature) qui devra décrire proximalement notre problème et que l'on cherchera à minimiser. Se présente donc une boite noire prenant trois entrées (les conditions initiales sur la vitesse) et renvoyant une valeur. Cette fonction `score` peut enfin être vue comme une surface dans un espace à quatre dimensions.



Une première approche serait d'évaluer la simulation sur la distance entre la balle et le trou une fois le glissement terminé. Cette fonction `score` possède cependant une infinité de minimum globaux potentiels. On pourrait aussi exiger de nos conditions initiales qu'elles minimisent le temps de trajet de la balle. Comment comparer une distance et un temps ?

On se convainc que le point *(1,1)* est objectivement meilleur que les deux autres. En revanche il est impossible de déterminer objectivement lequel des deux autres points est meilleur que l'autre. Raisonnant de même, il est possible de déduire une frontière d'efficacité nommée « front de Pareto ». Dans la figure ci-dessous, le point C est objectivement moins bon que les points A et B. On ne peut conclure quant à la qualité de A par rapport à B mais il nous est possible de rejeter C.

N.B : Un oint est bon s'il minimise l'espace et le temps.

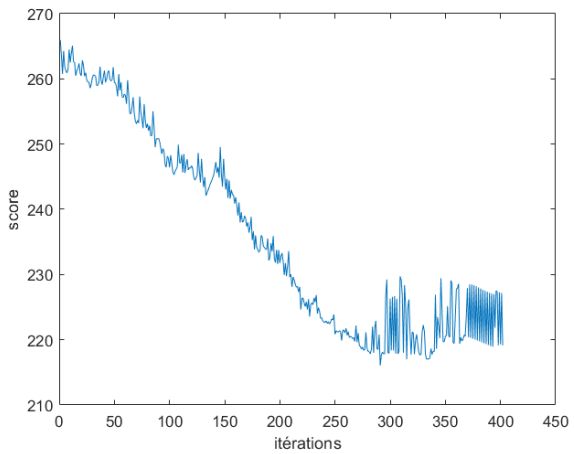


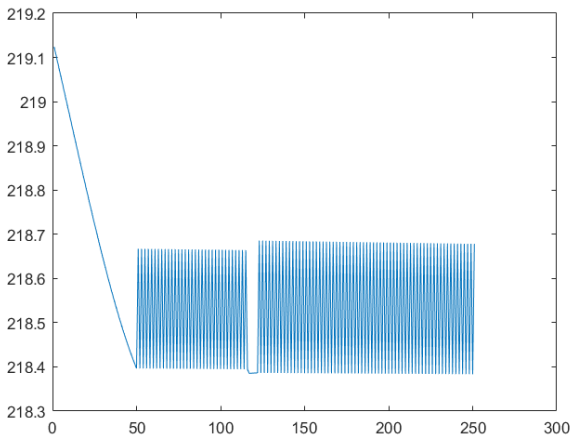
Il est aussi nécessaire d'adapter l'*échelle* entre la distance et le temps. En effet nous accorderons plus d'importance à minimiser la distance plus tôt que le temps afin d'éviter de rester *coincé* dans un minima local où la balle ne bouge quasiment pas. Nous évaluerons donc chaque lancé par une fonction du type score = a*distance²+b*temps² ; le but étant de minimiser le score.

8. Méthodologie – résolution du problème

8.1. Justification de l'optimisation par algorithme génétique

La fonction à optimiser n'est pas analytique, on la qualifie de boite noire. Un premier algorithme à tester est la descente du gradient. L'analogie naïve avec une balle glissant sur une surface peut être faite. Un minimum local peut ainsi être atteint *via* cette méthode. Cet algorithme nécessite tout de même que la fonction soit dérivable sur 'intervalle d'optimisation. Un rapide essai de cette méthode nous laisse penser que ce n'est pas le cas (sûrement à cause des rebonds) :





Une méthode stochastique retient donc plus notre attention. Ces méthodes ne sont pas rares en physique ; La méthode de Métropolis qui est un algorithme de Monte-Carlo à, par exemple, su montrer son efficacité. Nous présentons tout de même une méthode originale puisqu'il s'agit d'un algorithme génétique.

8.2. Présentation de l'algorithmique génétique

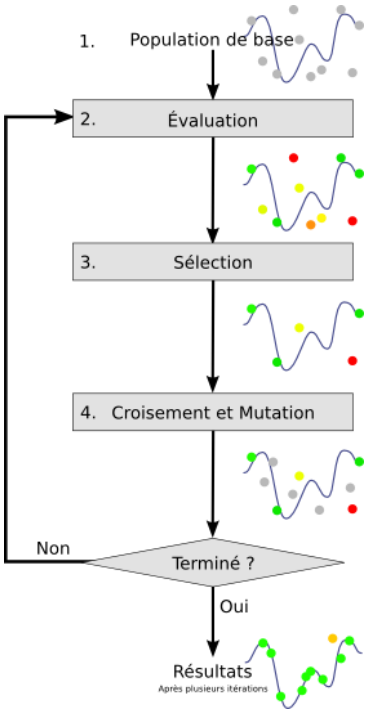
Dans cette partie nous utiliserons abondamment un langage emprunté de la biologie. Le vocabulaire a été détourné de sa réelle signification afin d'illustrer un concept mathématique abstrait. Un algorithme génétique se décompose de cette façon :

On manipule une population contenant un certain nombre d'individus chaque individus est en réalité un vecteur de trois coordonnée (Vox,Voy,Voz), un individu représente donc une coordonnée de notre fonction score. L'objectif de notre algorithme va donc être de trouver les individus associé aux valeurs les plus basse de la fonction score.

Pour ce faire la première étape est appelé sélection. Elle consiste à sélectionner un groupe d'individus qui transmettront leurs caractéristiques à la génération suivante. On peut pour ce faire sélectionner par exemples les x meilleurs individus du groupes (ceux qui sont associé aux valeurs de score les plus faibles). Ce processus est évidemment inspiré du processus de sélection naturel à l'origine de la théorie de l'évolution.

La deuxième étape consiste à reproduire les individus précédemment sélectionné. Et ainsi construire une nouvelle population qui constituera la seconde génération. Comme toujours il existe plusieurs méthodes et seul l'imagination du programmeur limite la créativité.

Il existe une troisième étape qui est la mutation. Cette étapes consiste à ajouter du *bruit* sur les valeurs déterminées par l'étape précédente. Ce processus permet de maintenir un certain niveau de *diversité* dans notre population et nous le verrons de dépasser des minimum locaux.



9. Structure détaillée de l'algorithme – application au golf

9.1. Structure

Nous vous avons exposé plus haut comment un algorithme génétique fonctionne en détaillant les étapes qui le compose. Cependant l'on doit faire preuve de créativité pour chaque étape. Nous vous exposeront ici l'algorithme que nous avons créé.

- ***Création de la population initiale***

Pour ce faire, nous générons un tableau de valeurs aléatoires. Les individus sont représentés par des vecteurs lignes comportant trois valeurs (génome) (Vox,Voy,Voz) (**nbp :** *la grande majorité du code s'adapte au formats de l'individu, libre a l'utilisateur d'augmenter ou de réduire la dimension du problème*), une population est donc une matrice taille **nombres_d'individus x nombres_de_CI**. Les valeurs de cette matrice sont générées aléatoirement libre à l'utilisateur de définir comment. La syntaxe matlab est la suivante :

```
population = 30*(ones(nbr_individus,3)-2*rand(nbr_individus,3));
```

On soustrait à une matrice unitaire une matrice de même dimension contenant des nombres aléatoires variant de -2 à 2. il en résulte une matrice de variables aléatoires comprises entre -1 et 1. Il ne reste plus qu'à augmenter l'amplitude des valeurs selon nos besoins.

- ***Processus de sélection***

A chaque individu est attribué une note calculée à partir du résultat de la simulation lancée avec les *coordonnée* de l'individu. La notes de chaque individus est intégrée dans un vecteur colonne appelé score. Pour sélectionner nos individus nous avons combiné plusieurs techniques.

La première sélection est une sélection *élitiste*. Elle consiste à *mettre de côté* les x meilleurs individus. Pour ce faire on classe le vecteur score du plus petit (le meilleur) au plus grand (le moins bon) et l'on récupère les indices associées à ce classement. On peut maintenant trier notre population selon le même ordre (les classer du meilleur au moins bon). Il ne reste plus qu'a renvoyer les *x* premier individus qui seront stocké dans le tableau « elite ». La syntaxe matlab est la suivante :

```
[~,idx] = sort(score);
    population_classe = population(idx,:);

    elite = population_classe(1:taux_elitisme,:);
```

le nombres d'individus composant l'élite est définis en haut du script principal par la variable taux_elitisme NB : Ces individus ne sont pas destiné à la reproduction

Puis nous appliquons un deuxième processus de sélection appelé le tournois (*tournament*) . Ici nous sélectionnerons les deux meilleurs individus dans un groupe de x individus tiré au sort dans la population. Cette façon de procéder permet de maintenir de la diversité dans notre population. Ces deux individus forment un couple de parent à partir desquels on générera un enfant. Pour ce faire on fonctionne de la même façon que

précédemment.

La syntaxe matlab est la suivante :

```
num_participant=ceil(length(population)*rand(k,1));
score_participant=zeros(k,1);

for n=1:k
score_participant(n)=score(num_participant(n)) ;
end

[~,idx] = sort(score_participant);
population_classe = population(idx,:);

parents = population_classe(1:nbr_parent,:)
```

Pour la troisième étape nous avons utilisé la méthode du *crossover* aussi appelé *recombination*. Nous allons mélanger les caractéristiques des deux parents pour générer un nouvel individu que l'on appellera l'enfant. Pour chacune de ces caractéristiques l'enfant à une chance sur deux de choisir celle de son père ou celle de sa mère

```
enfant = zeros(1,length(population(1,:)));
pere = parent(1,:);
mere = parent(2,:);
for i =1:length(population(1,:))
      if rand <=0.5
            enfant(1,i) = pere(1,i);
      else
            enfant(1,i) = mere(1,i);
      end
      end

end
```

Il ne nous reste plus à présent qu'à appliquer une mutation sur le nouvel individus. Cela se traduit par ajouter du bruit sur chacune de ces caractéristiques. Comme certaines caractéristiques peuvent être plus sensible que d'autre le bruit doit être adapté. L'utilisateur devra donc fournir un tableau de valeur de même dimension qu'un individu définissant l'amplitude de la mutation sur chacune des caractéristiques de l'enfant.

```
enfant = zeros(1,length(population(1,:)));
pere = parent(1,:);
mere = parent(2,:);
for i =1:length(population(1,:))
      if rand <=0.5
            enfant(1,i) = pere(1,i);
      else
            enfant(1,i) = mere(1,i);
      end
end
```

Ces trois étapes sont répétées en boucles jusqu'à ce qu'une nouvelle génération est été créée.

9.2. Stratégies : diversité - élitisme

- ***L'élitisme***

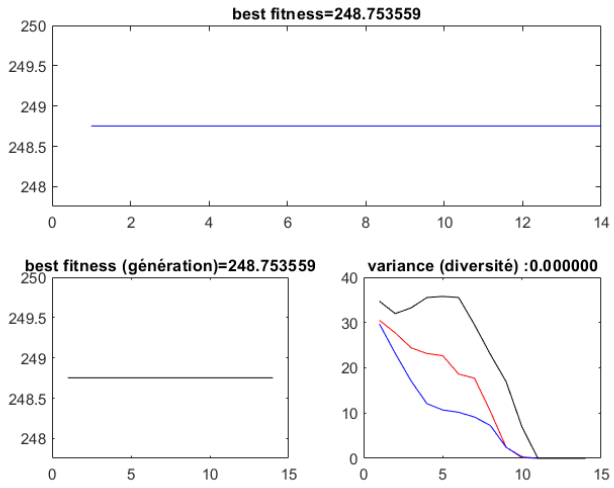
Nous avions mis de côté un groupe d'individus présélectionné appelé l'élite. Nous allons

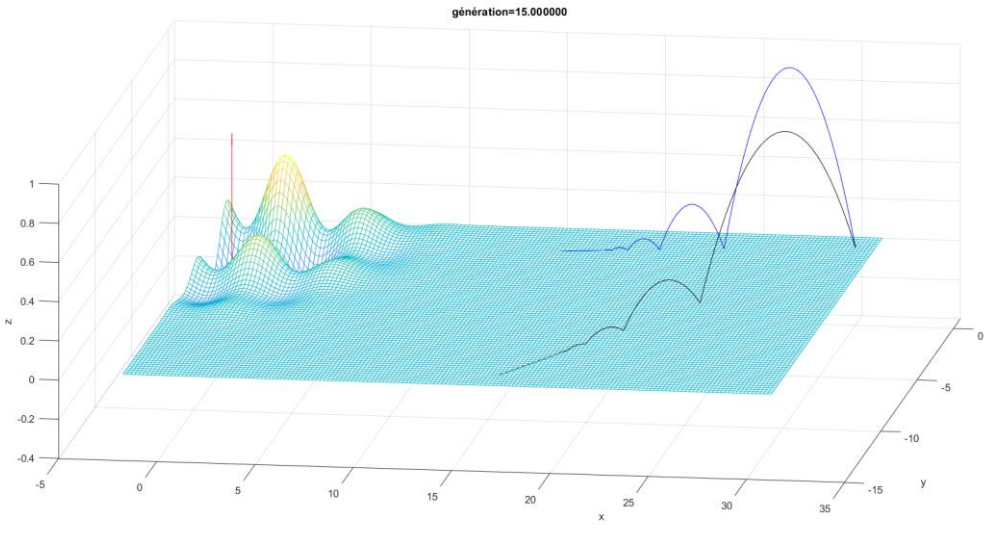
rajouter une étape avant de passer à la nouvelle génération. Nous réévaluons tous nos nouveaux individus et les classons du meilleur au moins bon. On remplace à présent les x moins bon individus par nos x individus composant l'élite. Cette étapes garanti de ne pas trop nous écarter d'une solution correcte au fil des générations.

- ***La diversité***

La diversité est un élément important que l'on tentera de maintenir. Un bon indicateur de la diversité de notre population peut être la variance de chaque génome. Pour ne gérer qu'une valeur nous ferons une moyenne de ces trois valeurs.

Voyons à présent en quoi la diversité est un paramètre essentiel. Nous avons lancé l'algorithme sans mutation et avec un fort taux d'élitisme (10 individus pour 60). NB l'aléatoire à été contrôlé afin de pouvoir comparer les changements de paramètres.

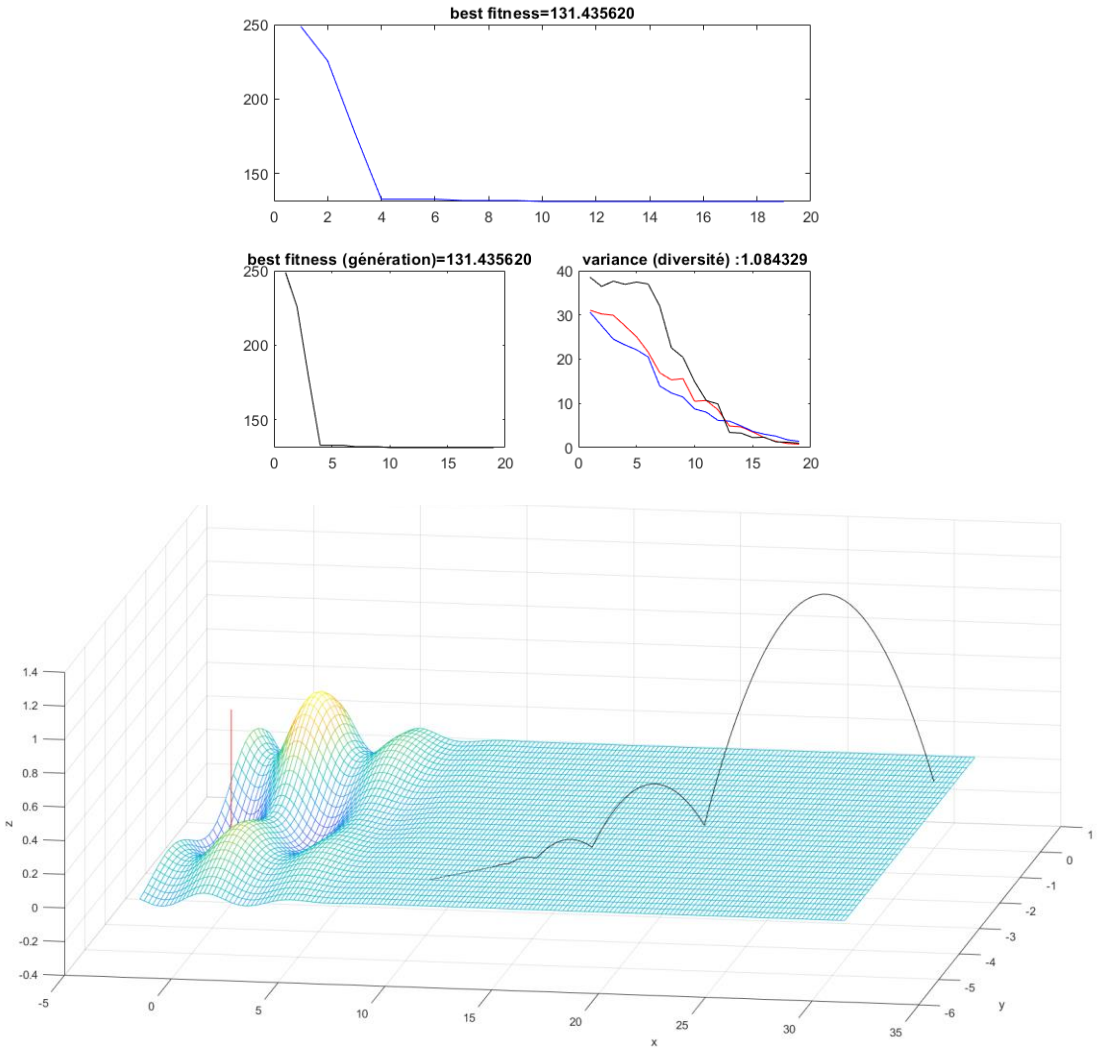




On observe que l'algorithme converge rapidement vers un minimum local et qu'après une dizaine de génération seulement la diversité chute à zéro. À ce moment tous nos individus sont identiques nous ne trouverons pas de meilleurs solution que celle proposé à droite qui est loin d'être satisfaisante. NB : la solution en noir est la meilleur de la génération et celle en

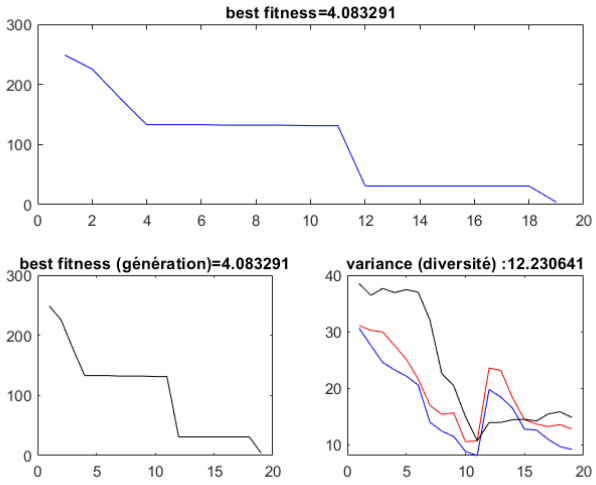
bleu est la meilleurs toutes générations confondu .

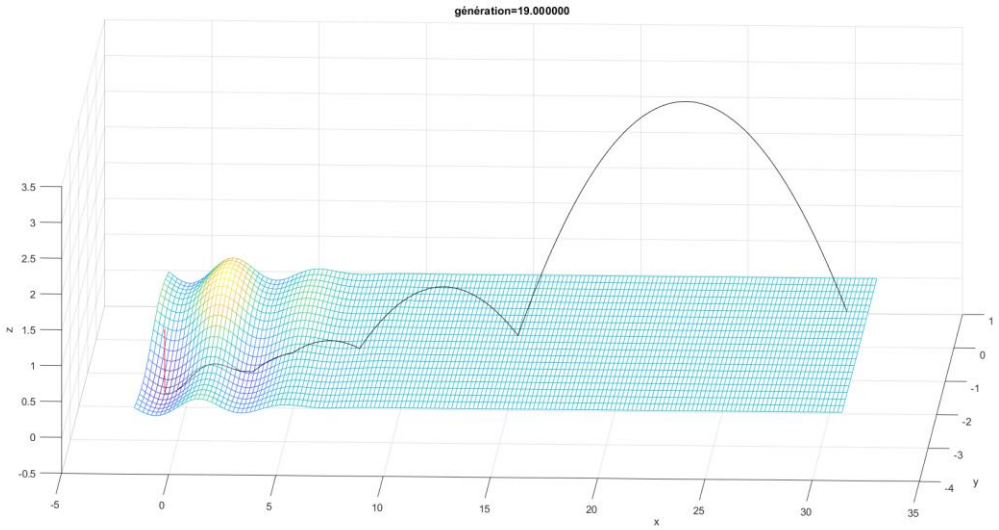
Corrigeons cela en ajoutant de la mutation (taux de mutation : [5,5,5] sur les enfants ; [2,2,2] sur les élites).



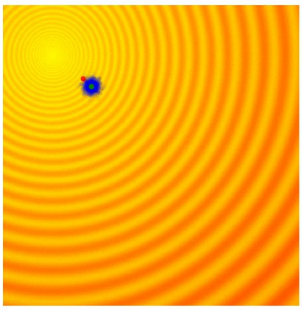
On remarque que la diversité finis toujours par chuter mais bien plus tard ce qui nous permet de trouver une solution bien plus satisfaisante. L'algorithme finis toujours par converger vers un minimum local.

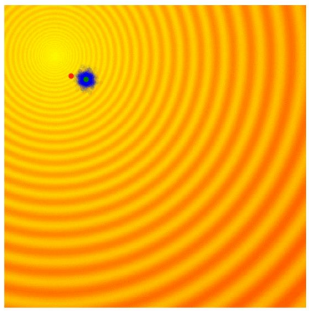
Pour pallier ce problème ajoutons un nouvel élément à notre algorithme. Nous testerons à chaque génération la diversité de notre population. Si elle passe en dessous d'une certaine valeur nous détruirons notre population et en reconstruirons une nouvelle. Pour ne pas que cette manipulation soit équivalente à repartir de zéro nous tirerons au hasard les génomes de chaque individus à l'aide d'une loi normale centrée autour de la valeur moyenne de ce génome mesurée sur toutes la population. La variance de chaque génomes peut être définie indépendamment par l'utilisateur de la même façon que le taux de mutation. C'est la fonction extinction qui gère ce renouvellement de population.

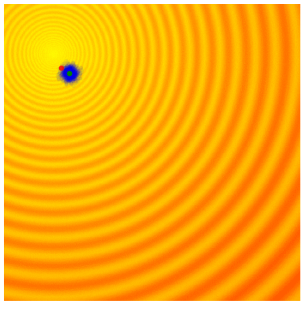


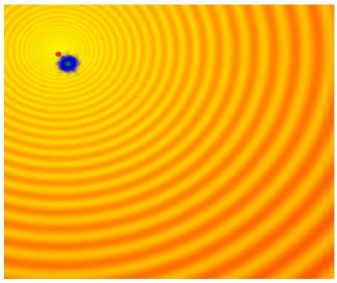
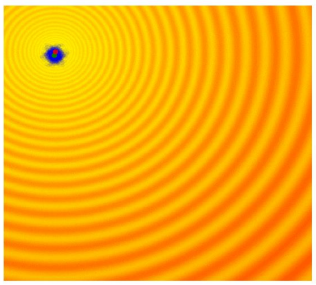
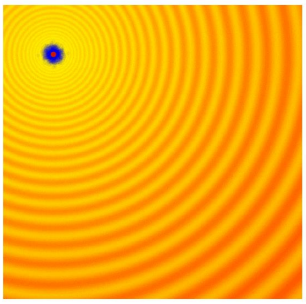


On remarque immédiatement que l'extinction c'est produite peu après la dixième génération et a entraîné une chute drastiques de la fitness permettant de trouver la première solution satisfaisante où la balle entre dans le trous. En un sens générer une nouvelle population autour de la meilleur solution est analogue à la descente de gradient où l'on suit la pente la plus forte.







Cette méthode à été appliqué à l'optimisation d'une surface 3D et on observe bien les similarités avec la descente de gradient.

10. Conclusion partielle

Nous avons donc vu que les méthodes d'optimisation génétiques peuvent être au moins aussi efficaces que les méthodes plus classiques comme la descente de gradient. Mais leurs plus gros avantage c'est qu'elle ne nécessite pas de forte conditions d'application ainsi a l'inverse de la descente du gradient elles peuvent facilement s'appliquer sur des fonctions non dérivables. Elles nécessitent en revanche d'être correctement paramétrée ce qui peut prendre plusieurs tentatives. De plus la réussite d'une tel méthode nécessite de passer un peu de temps à positionner correctement la problème. C'est à nous de déterminer la fonction fitness (score dans notre cas) que l'algorithme chercheras à optimiser. Si cette fonction n'est pas pertinente l'algorithme ne convergera pas vers une solution satisfaisante pour l'utilisateur.

III

Bibliographie

- Techniques Matlab :

  http://nte.mines-albi.fr/MATLAB/co/cnEDO.html

  http://www.thphys.nuim.ie/CompPhysics/matlab/help/techdoc/ref/ode45.html

- Techniques mathématiques :

  http://www.uqac.ca/calcul/chap3/chap39/VectTangent.html?fbclid=IwAR1Xc8qxXbR5QieeqJafYcqeQx9Unp0-aKX_5KEONVFx3oM-TW3n34cqEiE

  https://media4.obspm.fr/public/M2R/cours/chapitre3/souschapitre2/section4/page6/section3_2_4_6.html

- Golf :

  https://practicegolf.wordpress.com/video-ben-hogans-2/

  http://www.lecondegolf.com/glossaire_golf.htm

  https://fr.wikipedia.org/wiki/Balle_de_golf

- Techniques d'optimisation :

  http://blog.otoro.net/2017/10/29/visual-evolution-strategies/

  https://fr.wikipedia.org/wiki/Algorithme_g%C3%A9n%C3%A9tique

IV

Appendices

Appendice I.6.

Vol, rebonds et glissement sur un terrain quelconque

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%                              Projet Numérique                              %%%
%%%                               final_simulation                             %%%
%%%                            CHARRY. A - YAACOUB. D                          %%%
%%%                                                                            %%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
clear all; close all; clc ;


% Paramètres
P = Parametres() ;
m = P.masse ;
D = 0.5*P.traine*P.densite*pi*P.rayon^2 ;
Cm = P.magnus ;
g = P.gravite ;
r = P.rayon ;

options=P.options;
to = P.to ;
tf = P.tf ;

Wox = P.omega0x ;
Woy = P.omega0y ;
Woz = P.omega0z ;
CI = [P.Xo;P.Yo;P.Zo;P.Vox;P.Voy;P.Voz];

% fonction de l'EDO
F = @(t,Y) [Y(4);Y(5);Y(6);-sign(Y(4))*(D/m)*(Y(4)-P.Vwx)^2+(Cm/m)*(Woy*Y(6)-
Woz*Y(5));-sign(Y(5))*(D/m)*(Y(5)-P.Vwy)^2+(Cm/m)*(Woz*Y(4)-Wox*Y(6));-
sign(Y(6))*(D/m)*(Y(6)-P.Vwz)^2-g+(Cm/m)*(Wox*Y(5)-Woy*Y(4))];

% initialisation
t = to;
Y = transpose(CI);
rebond = 1 ;

% intégration

% rebonds
while rebond ;

    [t_vol,Y_vol]=ode45(F,[to,tf],CI,options);

    duree_vol = length(t_vol);
    t= [t;t_vol(1:duree_vol)];
    Y = [Y;Y_vol(1:duree_vol,:)];

    CI = Y_vol(duree_vol,:);
    X1 = impact(CI,'terrain',P);
    CI = X1(1:6);
```

```
    rebond = X1(7);
    to = t_vol(duree_vol);

    if tf == to
        break
    end
end

% glissement

% initialisation
x = CI(1) ;
y = CI(2) ;
z = CI(3) ;
Vx = CI(4) ;
Vy = CI(5) ;
Vz = CI(6) ;
V = [Vx,Vy,Vz]' ;
h = 0.0001 ;
tg = t(end) ;

% intégration
while tg < tf

    tol = 10^(-3) ;
    integration = 1 ;
    r = [x,y,z];

    while integration == 1                      % adaptation du pas

        r1 = glissement(r,V,h,P) ;

        r2_temp = glissement(r,V,h/2,P) ;
        r2 = glissement(r2_temp(1:3),r2_temp(4:6)',h/2,P) ;

        erreur = norm([r1(1);r1(2);r1(3)]-[r2(1);r2(2);r2(3)]);
        h = h*0.9*(tol/erreur)^(1/2);

        if erreur < tol
            x = r2(1) ;
            y = r2(2) ;
            z = r2(3) ;
            V = [r2(4);r2(5);r2(6)] ;
            break
        end
    end

    tg = tg +h ;
    t= [t;tg];

    Y = [Y;[x y z V(1) V(2) V(3)]];

    if norm(V) < tol
        break
    end
end

% Graphique
```

```
% terrain
terrain_x = min(Y(:,1))-1:0.2:max(Y(:,1))+1 ;
terrain_y = (min(Y(:,2))-1:0.2:max(Y(:,2))+1)';
terrain_z =  terrain(terrain_x,terrain_y);
C = terrain_z;

% Animation trajectoire

to =t(1);
dto = 0.01;

figure('Position',[0 0 1550 800])
for i = 1:length(Y(:,1))

    dt = t(i)-to;
    if dt >= dto
        mesh(terrain_x,terrain_y,terrain_z)
        axis equal
        hold on
        plot3(Y(1:i,1),Y(1:i,2),Y(1:i,3),'b',Y(i,1),Y(i,2),Y(i,3),'-ob')
        view(-30,52)
        title('trajectoire')
        title(sprintf('t=%f',t(i)))
        xlabel('x (m)')
        ylabel('y (m)')
        zlabel('z (m)')
        grid ON
        hold off

        to = t(i);
    end
    pause(0.00001)
end

% énergétique
Ec2=0.5*m*(Y(:,4).^2+Y(:,5).^2+Y(:,6).^2);
Ep = m*g*Y(:,3);
E = Ec2+Ep;

figure(2)
plot(t,Ec2,'r',t,Ep,'b',t,E,'k',t,Ec2,'g')
title('Energie cinetique-rouge | Energie potentielle-bleue | Energie
mécanique-noire)')
xlabel('Temps (s)')
ylabel('Energie (j)')
```

fonction impact

```
function I = impact(CI,F,P)

%Paramètres
e = P.restitution ;
r = P.rayon ;
X = CI ;
Vlimite = P.Vlimite;

% création d'une base orthonormée locale en un point d'impact de la surface

nz= normale(X(1),X(2),F) ;                  % axe z de la base locale {n}
nx = [-nz(3),0,nz(1)];                      % axe x de la base locale {n}
orthogonal au précédent
ny = cross(nz,nx);                          % axe y engendré par produit
nx = nx./norm(nx)  ;                        % normalisation de {n}
ny = ny./norm(ny);
Mat_n_e = [nx',ny',nz'];                    % Matrice de la base {n} dans {e}

% vitesse dans {n} - réflexion

V = [X(4),X(5),X(6)]' ;                     % vitesse
Vn = Mat_n_e\V   ;                          % Ovitesse dans la base du plan
tangeant
Vn(3) = -e*Vn(3) ;

% condition pour l'arret ou non des rebonds

if Vn(3) < Vlimite
    rebond = 0 ;
    Vn(3) = 0 ;
else
    rebond = 1 ;
end

% passage des résultats dans la base cartésienne fixe de travail

Ve = Mat_n_e*Vn ;
X(4) = Ve(1) ;
X(5) = Ve(2) ;
X(6) = Ve(3) ;

I = [X,rebond];
```

fonction normale

```
function n = normale(x0,y0,F)
h = 0.0001 ;
fgradx = (feval(F,x0+h,y0) - feval(F,x0,y0))/h;
fgrady = (feval(F,x0,y0+h) - feval(F,x0,y0))/h;
t1 = [1,0,fgradx];
t2 = [0,1,fgrady];
n = cross(t1,t2) ;
n = n./norm(n);
end
```

fonction glissement

```
function Y = glissement(r,V,h,P)
```

```
x = r(1);
y = r(2);
z = r(3);
g=P.gravite ;
m=P.masse ;
mu = P.frottement;
dt = h ;

% base locale normale orthonormée {n}

gradx = (terrain(x+h,y) - terrain(x,y))/h;
grady = (terrain(x,y+h) - terrain(x,y))/h;
t1 = [1,0,gradx];
t2 = [0,1,grady];
n = cross(t1,t2);
nz = n./norm(n);
nx = [-nz(3),0,nz(1)];                          % axe x de la base locale {n}
ny = cross(nz,nx);                              % axe y engendré par produit
nx = nx./norm(nx);                              % normalisation de {n}
ny = ny./norm(ny);
Mat_n_e = [nx',ny',nz'] ;                       % matrice de passe de {n} vers
{e}

% vitesse dans la base {n}

Vn = Mat_n_e\V ;
Vn(3)=0;
V = Mat_n_e*Vn;

% glissement

Poid = m*g*[0;0;-1];
alpha = acos(dot([0;0;1],nz'));
Resistance = m*g*cos(alpha)*nz';
frottement = -mu*V ;
a = (Resistance+Poid+frottement)/m;

% intégration

V = V + a*dt;
x = x + V(1)*dt ;
y = y + V(2)*dt ;
z = z + V(3)*dt ;

if z< terrain(x,y)
    z=terrain(x,y);
end

Y = [x,y,z,V(1),V(2),V(3)];
```

fonction évènement

```
function [value,isterminal,direction] = Evenement_rebond(t,Y)
value = Y(3) > feval('terrain',Y(1),Y(2));                    % arrêt de
l'itération quand cette condition n'est plus vérifiée
isterminal = 1;                                               % arrêt de
l'intégration
direction = -1;                                               % pour les
directions négatives seulement
```

fonction paramètres

```
function P = Parametres()           % fonction des paramètre pour un usage
clair et efficace

P.Xo = 0 ;                          % positions initiales
P.Yo = 0 ;
P.Zo = 0;

if P.Zo < terrain(P.Xo,P.Yo)        % on ne peut crever la surface dans les
paramètres instruits par l'utilisateur
    P.Zo = terrain(P.Xo,P.Yo)+0.01 ;
end

P.Vox = -1.5 ;                      % vitesses intiales de la balle de golf
P.Voy = 0 ;
P.Voz = 4 ;

P.omega0x = 0 ;                     % rotations initiales de la balle de golf
P.omega0y = 0 ;
P.omega0z = 10 ;

P.gravite = 9.81 ;                  % acélération de la pesanteur
P.densite = 1 ;                     % denstité volumique de l'air
P.traine =  0.2 ;                   % coefficient de trainé
P.magnus =  0.0005 ;                % coefficient de Magnus 5e-6 si non
dumpled sinon 5e-4 au moins
P.restitution = 0.5 ;               % coefficient de restitution (homogène
dans l'espace)
P.Vlimite = 0.5 ;                   % condition de roulement
P.frottement = 0.15 ;                % coefficient de frottement

P.rayon = 0.021135 ;                % rayon de la balle de golf
P.masse = 0.045 ;                   % masse de la balle de golf

P.Vwx = 0 ;                         % vitesses du vent
P.Vwy = 0 ;
P.Vwz = 0 ;

% Parametres d'integration
P.to = 0 ;                          % fenetre temporelle pour l'intégration
P.tf = 10 ;
P.options = odeset(  'RelTol',1e-6,...
                     'AbsTol', 1e-6,...
                     'Events', @bounceEvents,'MaxStep',0.1);
% option pour l'intégration
```

fonction terrain

```
function z = terrain(x,y)
%z = -1*x +0*y ;
%z = exp(-3*(x-1).^2 -3*y.^2)+exp(-3*(x+1).^2 -3*(y+1).^2);
z = exp(-0.05*x.^2 -0.05*y.^2).*(sin(1.5*x)+cos(1.5*y));
%z = 0.1+ 0*x+0*y ;
%z=sin(1.5*x)+cos(1.5*y);
%z=cos(1.5*x).*exp(-0.07*x.^2)+0*y;
```

Appendice II.9.

Optimisation par algorithme génétique

```
close all
clc
P = Parametres() ;


nbr_individus = 200;
taux_elitisme = 25;
population = 10*(ones(nbr_individus,3)-2*rand(nbr_individus,3));
lambda =  [5,5,5];
minimal_diversity = 10;

record = 9999999999999999;
record_generation = 99999999999999999;
generation = 1 ;
rec_plot=0;
recgen_plot=0;
gen_plot = 0 ;
var1 =0;
var2 =0;
var3 =0;

score = zeros(nbr_individus,1);
Y_record_generation=zeros(4,4);
Y_record=zeros(4,4);


while record > 0.1
    tic

    parfor n=1:nbr_individus

        Ytemp=simulation(population(n,:),P);
        score(n) = 0.8*norm(Ytemp(end,2:4)'-
[P.tXo;P.tyo;terrain(P.tXo,P.tyo)])^2+0.2*Ytemp(end,1)^2;

    end

    Y_record_generation=simulation(population(find(score==min(score)),:),P);
    record_generation = min(score);

    toc
    if min(score)<record
        Y_record=Y_record_generation;
        record = min(score);

    end

    % Graphique

    % terrain
```

```
    terrain_x = min([Y_record_generation(:,2);Y_record(:,2);P.tXo])-
1:0.2:max([Y_record_generation(:,2);Y_record(:,2);P.tXo])+1 ;
    terrain_y = (min([Y_record_generation(:,3);Y_record(:,3);P.tyo])-
1:0.2:max([Y_record_generation(:,3);Y_record(:,3);P.tyo])+1)';
    terrain_z =  terrain(terrain_x,terrain_y);
    C = terrain_z;

    figure(1)
    mesh(terrain_x,terrain_y,terrain_z)
    %axis equal
    hold on

plot3(Y_record(:,2),Y_record(:,3),Y_record(:,4),'b',Y_record_generation(:,2),
Y_record_generation(:,3),Y_record_generation(:,4),'k')
    view(5,40)
    title(sprintf('génération=%f',generation))
    xlabel('x')
    ylabel('y')
    zlabel('z')
    quiver3([P.tXo],[P.tyo],[terrain(P.tXo,P.tyo)],[0],[0],[1],'r')
    grid ON
    hold off

   elite = elitisme(population,score,taux_elitisme);

   for i=1:taux_elitisme
        elite(i,:)=mutation(elite(i,:),[0,0,0]);
   end

   new_population = zeros(nbr_individus,length(population(1,:)));

    for i=1:nbr_individus

        parent=selection(population,score,20,2);
        enfant=crossover(parent,population);
        enfant=mutation(enfant,lambda);
        new_population(i,:)=enfant;
    end

    newscore = zeros(nbr_individus,1);
    parfor i=1:nbr_individus

        Ytemp=simulation(new_population(i,:),P);
        newscore(i) = 2*norm(Ytemp(end,2:4)'-
[P.tXo;P.tyo;terrain(P.tXo,P.tyo)])^2+0.2*Ytemp(end,1)^2;

    end

    population = tri(new_population,population,newscore,elite);

    rec_plot=[rec_plot;record];
    recgen_plot=[recgen_plot;record_generation];
    var1=[var1;var(population(:,1))];
    var2=[var2;var(population(:,2))];
    var3=[var3;var(population(:,3))];
    gen_plot = [gen_plot;generation];
```

```
    figure(2)
    subplot(2,2,[1,2]);

plot(gen_plot(2:end),recgen_plot(2:end),'k',gen_plot(2:end),rec_plot(2:end),'
b');
    title(sprintf('best fitness=%f',record))
    subplot(2,2,3);
    plot(gen_plot(2:end),recgen_plot(2:end),'k');
    title(sprintf('best fitness (génération)=%f',record_generation))
    subplot(2,2,4);

plot(gen_plot(2:end),var1(2:end),'r',gen_plot(2:end),var2(2:end),'b',gen_plot
(2:end),var3(2:end),'k');
    title(sprintf('variance (diversité)
:%f',mean([var1(end),var2(end),var3(end)])))

    if mean([var1(end),var2(end),var3(end)])< minimal_diversity
        population = extinction(population,lambda);
    end

    generation=generation+1;
    record_generation =9999999999999;

end
```

fonction élitisme

```
function elite=elitisme(population,score,taux_elitisme)

    [~,idx] = sort(score);
    population_classe = population(idx,:);
    elite = population_classe(1:taux_elitisme,:);

end
```

fonction tri

```
function new_population = tri(new_population,population,score,elite)

    taux_elitisme=length(elite(:,1));
    nbr_individus=length(new_population(:,1));

    [~,idx] = sort(score);
    population_classe = population(idx,:);

    new_population = population_classe(1:nbr_individus-taux_elitisme,:);
    new_population = [elite;new_population];

end
```

fonction extinction

```
function new_population = extinction(population,sigma)

    nbr_individus=length(population(:,1));

    for i=1:length(population(1,:))
        new_population(:,i) =
normrnd(mean(population(:,i)),sigma(i),[nbr_individus,1]);
    end
end
```

fonction crossover

```
function enfant=crossover(parent,population)

enfant = zeros(1,length(population(1,:)));
pere = parent(1,:);
mere = parent(2,:);
    for i =1:length(population(1,:))
        if rand <=0.5
            enfant(1,i) = pere(1,i);
        else
            enfant(1,i) = mere(1,i);
        end
    end

end
```

fonction sélection

```
function parents=selection(population,score,k,nbr_parent)

num_participant=ceil(length(population)*rand(k,1));

score_participant=zeros(k,1);
for n=1:k
  score_participant(n)=score(num_participant(n)) ;
end

[~,idx] = sort(score_participant);
population_classe = population(idx,:);

parents = population_classe(1:nbr_parent,:);
```

fonction mutation

```
function individu = mutation(individu,lambda)

individu = individu - lambda*(0.5-rand);
end
```

fonction simulation

```
function solution = simulation(ci,P)

    m = P.masse ;
    D = 0.5*P.traine*P.densite*pi*P.rayon^2 ;
    Cm = P.magnus ;
    g = P.gravite ;
    options=P.options;
    to = P.to ;
    tf = P.tf ;
    Wox = P.omega0x ;
    Woy = P.omega0y ;
    Woz = P.omega0z ;
    CI = [P.Xo;P.Yo;P.Zo;ci(1);ci(2);ci(3)];

    % fonction de l'EDO

    F = @(t,Y) [Y(4);Y(5);Y(6);-sign(Y(4))*(D/m)*Y(4)^2+(Cm/m)*(Woy*Y(6)-
Woz*Y(5));-sign(Y(5))*(D/m)*Y(5)^2+(Cm/m)*(Woz*Y(4)-Wox*Y(6));-
sign(Y(6))*(D/m)*Y(6)^2-g+(Cm/m)*(Wox*Y(5)-Woy*Y(4))];

    t = to;
    Y = transpose(CI);
    P = Parametres() ;
    F = @(t,Y) [Y(4);Y(5);Y(6);-sign(Y(4))*(D/m)*Y(4)^2+(Cm/m)*(Woy*Y(6)-
Woz*Y(5));-sign(Y(5))*(D/m)*Y(5)^2+(Cm/m)*(Woz*Y(4)-Wox*Y(6));-
sign(Y(6))*(D/m)*Y(6)^2-g+(Cm/m)*(Wox*Y(5)-Woy*Y(4))];

    rebond = 1 ;

    while rebond

        [t_rebond,Y_rebond]=ode45(F,[to,tf],CI,options);

        n_steps = length(t_rebond);
        t= [t;t_rebond(1:n_steps)];

        Y = [Y;Y_rebond(1:n_steps,:)];
        CI = Y_rebond(n_steps,:);

        X1 = impacte(CI,Wox,Woy,Woz,'terrain',P);
        CI = X1(1:6);
        rebond = X1(7);
        to = t_rebond(n_steps);

        if tf == to
            break
        end
    end

    x = CI(1) ;
    y = CI(2) ;
    z = CI(3) ;
    Vx = CI(4) ;
    Vy = CI(5) ;
    Vz = CI(6) ;
    V = [Vx,Vy,Vz]' ;
```

```
    h = 0.0001 ;
    tg = t(end) ;

    while tg < tf

        tol = 10^(-3) ;
        integration = 1 ;
        r = [x,y,z];

        while integration == 1

            r1 = glissement(r,V,h,P) ;
            r2_temp = glissement(r,V,h/2,P) ;
            r2 = glissement(r2_temp(1:3),r2_temp(4:6)',h/2,P) ;        %
            erreur = norm([r1(1);r1(2);r1(3)]-[r2(1);r2(2);r2(3)]) ;
            h = h*0.9*(tol/erreur)^(1/2);

            if erreur < tol
                x = r2(1) ;
                y = r2(2) ;
                z = r2(3) ;
                V = [r2(4);r2(5);r2(6)] ;
                break
            end
        end

        if z < terrain(x,y)
            z=terrain(x,y);
        end

        tg = tg +h ;

        t= [t;tg];
        Y = [Y;[x y z V(1) V(2) V(3)]];

        if and(norm([x;y;z]-
[P.tXo;P.tyo;terrain(P.tXo,P.tyo)])<P.rayontrous,norm(V) < P.Vlimite)
            break
        end

        if norm(V) < tol
            break
        end

    end

solution = [t,Y] ;
end
```

fonction impact

idem ***I.6.***

fonction normale

idem ***I.6.***

fonction glissement

idem ***I.6.***

fonction évènement

idem ***I.6.***

fonction paramètres

```
function P = Parametres()           % fonction des paramètre pour un usage
clair et efficace

P.Xo = 30 ;                          % positions initiales
P.Yo = 0 ;
P.Zo = -4;

P.tXo = -1 ;                            % positions du trous
P.tyo = -2 ;
P.rayontrous = 0.04 ;

if P.Zo < terrain(P.Xo,P.Yo)        % on ne peut crever la surface dans les
paramètres instruits par l'utilisateur
    P.Zo = terrain(P.Xo,P.Yo)+0.01 ;
end

P.omega0x = 0 ;                     % rotations initiales de la balle de golf
P.omega0y = 0 ;
P.omega0z = 0 ;

P.gravite = 9.81 ;                  % acélération de la pesanteur
P.densite = 1 ;                     % denstité volumique de l'air
P.traine =  0.2 ;                   % coefficient de trainé
P.magnus =  0.000005 ;              % coefficient de Magnus
P.restitution = 0.5 ;               % coefficient de restitution (homogène
dans l'espace)
P.Vlimite = 0.2 ;                   % condition de roulement
P.frottement = 0.15 ;                % coefficient de frottement

P.rayon = 0.021135 ;                % rayon de la balle de golf
P.masse = 0.045 ;                   % masse de la balle de golf

P.Vwx = 0 ;                         % vitesses du vent
P.Vwy = 0 ;
P.Vwz = 0 ;

% Parametres d'integration

P.to = 0 ;                          % intervalle temporel
P.tf = 100 ;
P.options = odeset(  'RelTol',1e-6,...
                     'AbsTol', 1e-6,...
                     'Events', @Evenement_rebond,'MaxStep',0.1);
```

fonction terrain

idem ***I.6.***