PHS4700

Physique pour les applications multimédia

Automne 2015

Numéro de devoir : 2

Numéro de l’équipe : 18

|  |
| --- |
| Nom: Gagné Prénom : Alex matricule: 1689761  Signature : |
| Nom: La Rocque Carrier Prénom : Félix matricule:1621348  Signature : |
| Nom: Gamache Prénom : Mathieu matricule: 1626377  Signature : |
| Nom: Fedorov Prénom : Konstantin matricule: 1679095  Signature : |

# Description du problème à résoudre (1 pt)

Le but de ce devoir est d’effectuer une simulation qui est d’apparence simple mais qui nécessite l’utilisation de plusieurs concepts. Un de ses concepts est l’utilisation de méthode numérique pour résoudre des problèmes de simulations. Pour cet aspect, il va falloir déterminer quel méthode qu’on veut ainsi que ses paramètres, ce choix va se faire en fonction de nos besoins de précisions qu’on détermine nécessaire. La compréhension seulement théorique ne sera pas suffisante car il va falloir l’appliquer dans un programme matlab. Un autre but de ce devoir est de voir comment simuler un objet en mouvement. Puis finalement, il va falloir quel sont les moyens pour déterminer si la simulation est terminé ou non.

Plus précisément, le contexte de ce devoir est de simuler un lancer de base-ball. On doit considérer la trajectoire de la balle mais aussi les conditions pour déterminer s’il s’agit d’une prise ou d’une balle (selon si elle atteint une cible imaginaire). Trois types de simulations sont demandés, chacune d’elle intégrant une difficulté supplémentaire. Le premier lancer va seulement considérer la force gravitationnelle, une force qui va être constante tout le long de son vol. Le second lancer intègre une force de frottement, une force qui dépend de la vitesse de la balle. Puis le dernier lancer, intègre la force Magnus, qui dépend de la vitesse angulaire de la balle.

Les équations importantes pour les simulations (3 pts)

2.1 Équations du mouvement à résoudre (1 pt)

Notre fonction reçoit comme paramètre la vitesse initiale de la balle. On connaît les possibles forces qui s’applique sur la balle le long de son parcourt et on cherche à obtenir la positon de la balle à plusieurs temps t. On connait aussi la position initiale et la masse de la balle. On cherche donc des équations qui mettent en relation x0, m et F et qui permettent de déterminer x.

### 2.1.1 Équation pour un mouvement rectiligne uniformément accéléré

Dans le cas d’une accélération constante, ce qui est le cas avec l’option où seulement le gravité est à considéré, il n’est pas nécessaire d’utiliser des méthodes numériques, puisque les intégrales sont facile à résoudre analytiquement. Il existe 3 équations pour les MRUA

|  |  |
| --- | --- |
| **Variable** | **Signifiation** |
| x(t) | Position de l’objet au temps t |
| x0 | Position initiale de l’objet (à t = 0) |
| v0 | Vitesse initiale de l’objet (à t = 0) |
| t | Temps écoulé depuis le temps initial (t=0) |
| a | Accélération de l’objet (soit être constante) |

|  |  |
| --- | --- |
| **Ce qu’on cherche** | **Équations** |
| La position de l’objet au temps t |  |

### 2.1.1 Équations pour les accélérations non constante.

Pour les options de lancers 2 et 3, l’accélération n’est pas constante puisque la force appliqué sur la balle dépend de la vitesse de celle ci.

|  |  |
| --- | --- |
| **Variable** | **Signifiation** |
| v(t) | Vitesse au temps t |
| v(t0) | Vitesse initiale |
| a(t) | Accélération au temps t |
| r(t) | Position au temps t |

|  |  |
| --- | --- |
| **Ce qu’on cherche** | **Équations** |
| La vitesse aux temps t |  |
| La position aux temps t |  |

Hors ici l’accélération ne dépend pas du temps, il va donc falloir chercher d’autre équation.

### 2.1.3 Équations pour trouver la vitesse en fonction de la force

En premier lieu, il va falloir être en mesure de transformer la force, chose qu’on connaît, en une accélération, quelque chose qu’on a besoins.

|  |  |
| --- | --- |
| **Variable** | **Signifiation** |
| a(t) | Accélération au temps t |
| Fnet(t) | Force net au temps t |
| m | Masse (kg) |

|  |  |
| --- | --- |
| **Ce qu’on cherche** | **Équations** |
| L’accélération au temps t |  |

On est donc en mesure de calculer l’accélération au temps t connaissant la force appliqué au temps t.

### 2.1.4 Équations de la force dans notre devoirForc

Dans notre devoir 3 types de forces vont s’appliquer à notre balle.

**Force gravitationnelle**

|  |  |
| --- | --- |
| **Variable** | **Signifiation** |
| Fg | Force gravitationnelle |
| mb | Masse de la balle |

|  |  |
| --- | --- |
| **Ce qu’on cherche** | **Équations** |
| Force gravitationnelle |  |

**Force de frottement**

|  |  |
| --- | --- |
| **Variable** | **Signifiation** |
| Fv | Force de frottement |
| db | Diamètre sphère |
| pair | Densité de l’air |
| Cv | Coefficient de frottement visqueux de l’air |
| v | Vitesse de la balle |

|  |  |
| --- | --- |
| **Ce qu’on cherche** | **Équations** |
| Force de frottement |  |

**Force de Magnus**

|  |  |
| --- | --- |
| **Variable** | **Signifiation** |
| Fm | Force de magnus |
| db | Diamètre de la balle |
| pair | Densité de l’air |
| Cm | Coefficient de Magnus |
| w | Vitesse angulaire |

|  |  |
| --- | --- |
| **Ce qu’on cherche** | **Équations** |
| Force de Magnus |  |

2.2 Équations qui contrôlent l’arrêt de la simulation(2 pts)

Il est important d’avoir des conditions d’arrêts pour être en mesure d’arrêter les calculs de la simulation. Sans ces conditions on pourrait gaspiller du temps de calcul et l’affichage graphique de la solution serait confondante. Dans ce devoir il existe 2 conditions d’arrêts, une pour déterminer si le lancer est une prise, c’est à dire que la balle la balle est entièrement à l’intérieur de la zone de prise, une carré imaginaire au dessus du marbre d’une taille prédéfinis, ou d’une balle, la balle à toucher le sol avant d’atteindre la zone de prise.

Pour les formules suivantes, on assume que notre méthode va avoir un mécanisme de contrôle pour s’assurer que la balle ne passe pas tout droit.

|  |  |
| --- | --- |
| **Variable** | **Signifiation** |
| zi | Position de la balle au temps i, par rapport à l’axe des z |
| r | Rayon de la balle |
| zs | Position du sol, par rapport à l’axe des z |
| ycmin | Position en y du côté gauche de la cible |
| yi | Position de la balle au temps i, par rapport à l’axe des y |
| ycmax | Position en ydu côté droit de la cible |
| zcmin | Position en z du côté en bas de la cible |
| zcmax | Position en z du côté en haut de la cible |
| xi | Position de la balle au temps i, par rapport à l’axe des x |
| xc | Position de la cible par rapport à l’axe des x |

|  |  |
| --- | --- |
| **Ce qu’on cherche** | **Équations** |
| La balle traverse le sol | zi-1 + r >= zs > zi + r |
| La balle traverse la position de la cible | ycmin <= yi + r <= ycmax  zcmin <= zi + r <= zcmax  xi-1 - r <= xc <= xi + r |

Méthodes de résolution des équations

## 3.1 Équations numérique à résoudre

Puisque ça nous tente pas de résoudre analytiquement ces équations, on va employer des méthodes numériques. Qui consiste à trouver les valeurs de q(tn) basé sur la valeur de q(tn-1).

|  |  |
| --- | --- |
| **Variable** | **Signifiation** |
|  | Condition initiale (ici la position initiale de la balle) |
|  | Différentielle de q au temps t. (ici la vitesse de la balle au temps t) |
|  | Valeur de q au temps t. |
| ki | Approximation de pour la méthode Runge-Kutta |
| O(delta) | Erreur |

|  |  |
| --- | --- |
| **Ce qu’on cherche** | **Équations** |
| Méthode de Euler |  |
| Méthode de Runge-Kutta |  |

Dans notre cas les vecteurs g et q ont les significations suivantes:

|  |  |
| --- | --- |
| **Variable** | **Formule** |
| Vecteur ayant comme composant la vitesse et la position |  |
| Vecteur ayant comme composant l’accélération et la vitesse. |  |

Donc on part du principe que si on connaît la vitesse et la position au temps t, on est capable de trouver l’accélération à ce temps (dépendant de l’option choisi). Ensuite les méthodes numériques permettent d’estimer la dérivée de q, ce qui permet d’estimer la vitesse et la position au temps t+1.

Lors des simulations pour ce devoir il est requis que pour chaque temps t fournis, il doit y avoir une précision de ±1mm pour la position en x, y ,z.

Question d’avoir une meilleur précision nous allons utiliser la méthode de Runge-Kutta au lieu de la méthode d’Euler. Cette dernière à une erreur globale proportionnelle à delta-t et est donc moins précise du (delta-t)^4 de Runge-Kutta.

3.2 Description des mesures de vérifications de la précision(2 pts)

Nous avons 2 contraintes de précisions, la première étant que la réponse finale doit avoir une précision de ±1mm et la deuxième il faut que pour toute simulation soit capable de détecter les conditions d’arrêts: il ne faut pas que entre 2 temps consécutif on manque la cible ou le sol. Nous utilisons donc 2 mécanismes pour respecter ces contraintes.

## 3.2.1 Contrainte sur la réponse finale de ±1mm

|  |  |
| --- | --- |
| **Variable** | **Signifiation** |
|  | Réponse exacte de q(t) |
|  | Estimation de q(t), avec un pas de temps spécifié |
|  | Erreur de l’estimation de q(t) utilisant le pas de temps spécifié |

|  |  |
| --- | --- |
| **Ce qu’on cherche** | **Équations** |
| Première estimation |  |
| Deuxième estimation avec un pas de temps plus petit |  |
| Estimation de la constante c |  |
| Vecteur de la solution moyenne | avgSol = (qb(t, tm+1) + qa(t, tm))/2 |
| Vecteur d’erreur de la solution | errSol = (qb(t, tm+1) - qa(t, tm)) |
| Évaluation de l’erreur relative | MaxErr = max(abs(errSol/avgSol)) |

Donc en effectuant 2 simulations avec des pas de temps différents on est capable d’estimer l’erreur et donc la précision de notre simulation. On calcule d’abord la solution moyenne en se basant sur les 2 premières simulations, puis l’erreur à chaque point. Ensuite on estime que notre précision est égale à la plus grande erreur relative. Le principe va donc de rendre le pas de temps de plus en plus petit jusqu’à ce qu’on obtienne une erreur de moins de 1mm. L’avantage avec cette méthode est qu’on a un pas de temps constant et donc évite de calcul à chaque pas, ce qui va simplifier l’écriture du code. Le désavantage est qu’on doit souvent effectuer plusieurs fois la simulation, ce qui est coûteux.

## 3.2.2 Contrainte pour ne pas manquer condition de terminaison

Comme il est possible qu’on commence avec des pas de temps initialement grand, on doit avoir un mécanisme pour s’assurer qu’on ne passe pas tout droit autour des zones de terminaison, c’est à dire la cible ou le sol. Le principe de notre solution est que lorsque qu’on approche d’une de ses zones on diminue le pas de temps pour faire sorte qu’on ne manque pas la zone. Il y a 2 problèmes à résoudre pour notre solution: il faut trouver ce qu’on veut dire par “approche de la zone” et de combien on diminue le pas de temps. On commence à diminuer le pas de temps, si avec sa position, sa vitesse et le pas de temps courante la balle se retrouverait de l’autre côté de la cible. On va diminuer le pas de temps de façon a ce qu’avec sa vitesse courante, elle ne se déplace pas plus de son diamètre, de cette façon, la balle ne va jamais passer tout droit.

3.3 Justification du pas de temps (1 pt)

On va donc avoir un pas de temps initial, qui va être raffinés au fur des itérations de simulations jusqu’à obtenir la précision désiré et un pas de temps d’approche, qui va être utilisé lorsque la balle va s’approcher des zones critique (nous avons déjà abordé ce pas de temps dans la section 3.2.2). Pour ce qui est du pas de temps initial il est important qu’il ne soit ni trop grand, car il va demander beaucoup de raffination et donc être coûteux, ni trop petit car il exécuterait plus d’itérations et donc de calcul que nécessaire. Après plusieurs test on a déterminé que le meilleur pas de temps initial pour notre contexte est de 0,02s. Toutefois, même si nous utilisions un pas de temps plus élevé, notre programme se chargerait de le diminuer jusqu’au point où nous respectons la contrainte de précision (pour les options 2 et 3 qui sont résolues numériquement).

Description du logiciel matlab (4 pts)

Dans cette section, nous décrivons le fonctionnement de notre programme. Pour le consulter

directement, voir l’Annexe A qui contient le code Matlab.

Contrairement au premier devoir, nous avons décidé de ne pas utiliser la programmation orienté-objet puisque nous ne croyons pas que cela est pertinent à la résolution du problème. Nous avons donc décider de diviser le programme en plusieurs fonctions, chacunes contenant une partie de la fonctionnalité du programme.

test.m est la partie principale du programme. Dans cette partie, nous appelons la fonction Devoir2 nous permettant de calculer la position de la balle selon les trois tirs et selon les différents modèles utilisés (lignes 1 à 6, 53 à 56 et 153 à 155). Les lignes 8 à 48, 108 à 149 ainsi que 157 à 198 permette l’affiche des résultats calculés par la fonction Devoir2.

La fonction Devoir2 calcule les résultats d’un tir selon l’option utilisé. Les variables initiales du problème sont définies aux lignes 2 à 11. Si l’on choisit de calculer la position de la balle en prenant seulement en compte l’accélération gravitationelle, les calculs sont faits directement dans la fonction.. Lors de ces calculs, nous utilisons FixDTZoneCrit ainsi que checkEnd. FixDTZoneCrit nous permet de modifier le pas utilisé lors du calcul de la position et checkEnd vérifie si la position de la balle vérifie une des conditions signifiant l’arrêt de la simulation. Pour les autres options, on fait appel à la fonction SOLRK4C pour les calculs en utilisant les autres forces s'appliquant sur la balle. Dans l’appel de fonction, on spécifie la fonction g à utiliser lors de la méthode de Runge-Kutta.

La fonction FixDTZoneCrit permet de modifier le pas de temps lorsque la balle se rapproche du sol ou bien de la cible. On vérifie d’abord si la balle est entre la position de la cible ou du sol et deux fois plus loin que la distance qu’elle peut traverser en deux fois son pas de temps de base. Si oui, nous modifions le pas de temps courant afin que la distance parcourue avant la prochaine vérification soit plus petite que le diamètre de la balle, nous assurant que nous n’allons pas dépasser la cible.

La fonction checkEnd permet de détecter la condition d’arrêt de la simulation. Les lignes 2 à 5 sont une définition des différentes conditions de base du problème. Nous vérifions ensuite les conditions d’arrêt en implémentant les équations définies à la section 2.2.

La fonction SOLRK4C contient la méthode de résolution utilisée pour les options 2 et 3. Elle reçoit en paramètre la fonction utilisée pour calculer l’accélération et la vitesse de la balle en fonction de sa vitesse et position précédente (soit les fonctions Option1 et Option2). SOLRK4C appelle tout d’abord SOLRK4 pour avoir une première simulation de la trajectoire de la balle avec le pas de temps par défaut. Ensuite, SOLRK4 est rappelée avec un pas deux fois plus petit jusqu’à ce que deux simulation consécutives aient une erreur relative respectant nos contraintes. Par la suite, étant donné qu’il est possible qu’une collision avec la zone des prise n’ait pas été détectée si le pas de temps était trop grand, une seconde simulation est effectuée à partir du point juste avant de dépasse le plan x = 0. Cette seconde simulation utilise un pas de temps bien plus petit pour faire en sorte qu’entre chaque point la balle se déplace d’au plus la moitié de son rayon. Si cette nouvelle simulation détecte une prise, ses points sont ajoutés à la liste de la première simulation et on retourne le résultat. Sinon, on effectue une troisième simulation similaire à la deuxième à partir du dernier point avant que la balle touche le sol, pour avoir une bonne précision sur la position et la vitesse de la bille à la collision.

La fonction SOLRK4 est essentiellement une boucle appelant la fonction SEDRK4 avec la fonctiong, permettant le calcul de la prochaine position de la balle et vérifiant si la balle atteint une condition d’arrêt avec la fonction checkEnd. Lorsqu’une condition d’arrêt est atteinte, la boucle s’arrête et on obtient les positions de la balle dans le temps.

La fonction SEDRK4 est l’implémentation de la méthode de Runge-Kutta d’ordre 4. En utilisant cette méthode, nous pouvons obtenir la position de la balle selon la fonction g et nous pouvons nous s’assurer d’obtenir la précision nécessaire au problème. La fonction appelle feval, permettant d’évaluer la fonction g.

Les deux différentes fonctions g possibles sont Option2 et Option3. La première fonction g calcule l’accélération du à la force visqueuse puis y rajoute l’accélération gravitationelle de la balle. La deuxième fonction g calcul l’accélération du force de magnus ainsi que la force visqueuse et ajoute l’accélération de ces deux forces à l’accélération gravitationelle.

Présentation et analyse des résultats (3 pts)

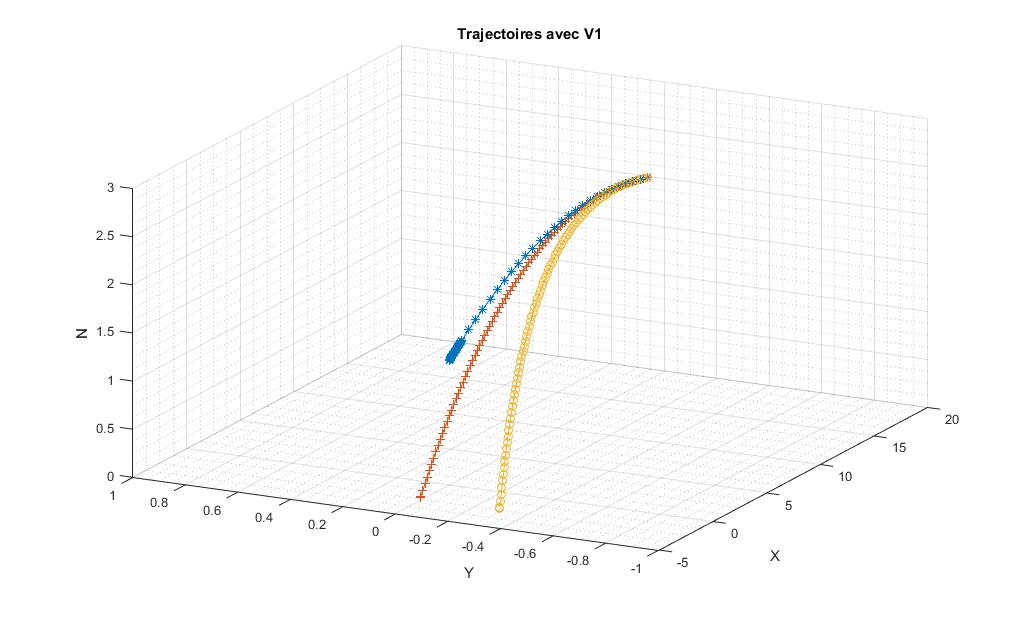
## Tir #1

vi = (-120, 0, 4.55)km/h

### Tableau des résultats:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Résultat | Temps d’arrêt (sec) | Vitesse  (m/s) | Position  (m) |
| Option #1 | Prise | 0.55 | (-33.33, 0, -4.15) | (-0.017, 0, 1.29) |
| Option #2 | Balle | 0.83 | (-19.81, 0, -5.73) | (-2.71, 0, 0.032) |
| Option #3 | Balle | 0.83 | (-19.81, -0.57, -5.73) | (-2.71, -0.30, 0.032) |

### Graphique



### Analyse

En ne considérant que la force gravitationnelle, nous pouvons voir que la balle garde la vitesse initiale en X (33.3 m/s) et gagne une vitesse en Y due a la gravité. La balle ne prend aucune vitesse en Z.

Pour ce qui est de l’ajout de la force de frottement visqueuse avec l’air, nous pouvons observer que la balle ralenti beaucoup (du à cet force) selon l’axe des X. Avec ce ralentissement, la balle subit plus longtemps la force gravitationnelle qu’avec l’option 1. Cela explique la plus grande vitesse selon l’axe des Y. Encore une fois, aucune vitesse n’est en Z.

Pour l’option 3, l’ajout de la force de Magnus influence encore une fois le comportement de la balle. Cette force s’ajoute aux deux autres déjà appliqués sur la balle. La balle tournant autour de l’axe des X, n’influence pas les vitesses dans les autres directions que les Z. C’est ce que nous pouvons observer dans les résultats plus haut. La balle ne prend pas plus de temps à parcourir que l’option 2 (pas d’influence de la vitesse en X) et elle atteint le sol au même moment (pas de changement de vitesse en Y).

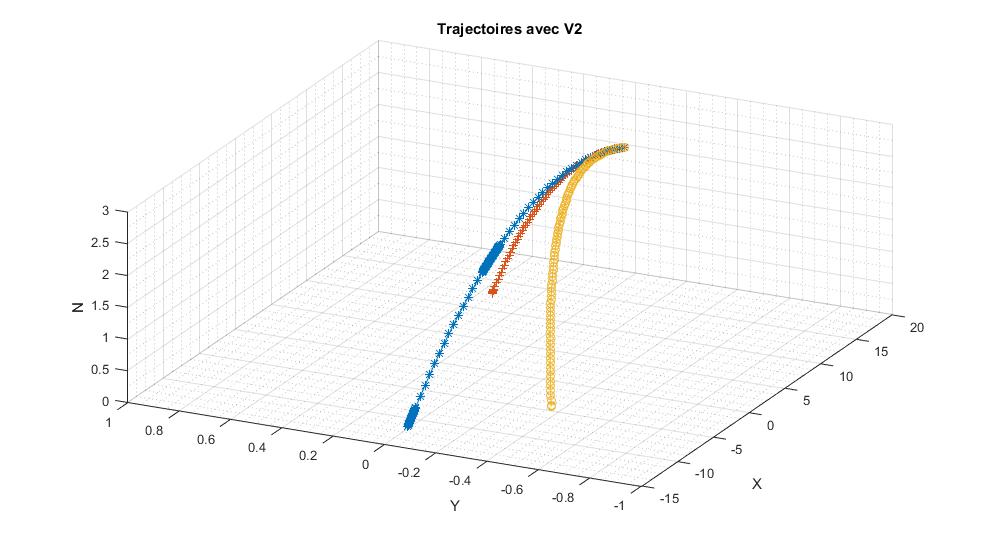
## Tir #2

vi = (-120, 0, 7.79)km/h

### Tableau des résultats:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Résultat | Temps d’arrêt  (sec) | Vitesse  (m/s) | Position  (m) |
| Option #1 | Balle | 0.91 | (-33.33, 0, -6.71) | (-11.77, 0, 0.036) |
| Option #2 | Prise | 0.698 | (-21.23, 0, -4.22) | (0.006, 0, 1.23) |
| Option #3 | Balle | 0.933 | (-18.86, -0.59, -5.91) | (-4.68, -0.36, 0.036) |

### Graphique



### Analyse

Dans ce tir, nous ajoutons une plus grande vitesse de tir de balle dans l’axe des Y (le lanceur lance plus fort vers le haut). Les résultats devraient beaucoup ressembler le premier tir au niveau de l’influence des différentes forces mais peut largement varier dans les résultats

Avec l’option 1, nous pouvons voir que l’augmentation de la force initiale en Y influence beaucoup le temps d’arrêt de la balle (celle-ci est lancé plus fort vers le haut), même si la vitesse en x ne change pas entre les deux tirs. Au niveau du résultat, l’augmentation de la force initial en Y fait passer la balle au dessus de la zone de frappe, causant une balle.

Avec l’ajout de la force de frottement, nous pouvons encore voir que la garde sa vitesse en X. Par contre, ici la balle est ralenti assez par la force de frottement pour atteindre la zone de tir et ainsi causer une Prise.

Avec l’ajout de la force de Magnus, nous pouvons encore observer que la vitesse de la balle augmente selon l’axe des Z. Avec l’ajout de la force en Y, la balle peut atteindre la zone de frappe, comparer au tir 1.

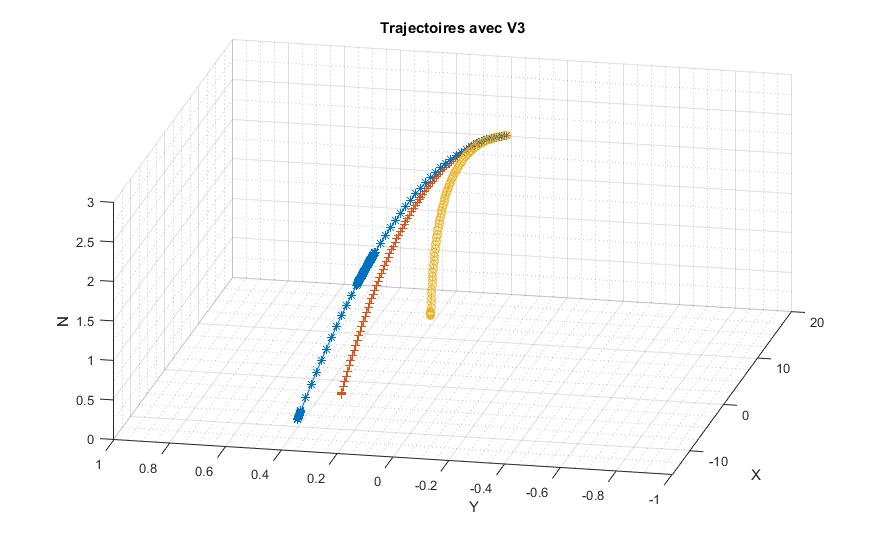
## Tir #3

vi = (-120, 1.8, 5.63)km/h

### Tableau des résultats:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Résultat | Temps d’arrêt (sec) | Vitesse  (m/s) | Position  (m) |
| Option #1 | Balle | 0.828 | (-33.33, 0.5, -6.54) | (-9.16, 0.41, 0.036) |
| Option #2 | Balle | 0.864 | (-19.50, 0.29, -5.78) | (-3.35, 0.32, 0.036) |
| Option #3 | Prise | 0.698 | (-21.23, -0.21, -4.60) | (0.006, 0.048, 0.895) |

### Graphique



### Analyse

Dans ce tir, nous ajoutons une plus petite vitesse de tir de balle dans l’axe des Y, mais une vitesse en Z (le lancer est sur le côté).

Pour l’option 1, nous pouvons voir que l’ajout d’une vitesse initiale en Z influence la position finale et que la vitesse finale en X et Z est identique à celle initiale (pas de modification par la gravité sur ces axes). Le lancer est ici aussi une balle comme il passe par dessus de la zone de frappe.

Pour l’option 2, nous pouvons voir que la force de friction influence la vitesse en X, Y et X car le déplacement de la balle se fait selon ces 3 axes. L’ajout de la vitesse initiale en Z fait passer la balle hors de la zone de frappe.

Finalement, pour l’option 3, en ajoutant la force de Magnus, nous pouvons encore voir l’influence sur l’axe des Z. Avec l’ajout de la force initiale, la balle peut passer dans la zone de frappe.

Discussion des défis rencontrés

Avec le devoir #1, on a acquis une certaine compréhension du langage Matlab, donc son utilisation cette fois ci était plus facile. La difficulté de ce devoir concernait le problème la résolution du problème lui même, notamment avec les méthodes de résolution numérique et les conditions d’arrêts. L’utilisation des méthodes numériques demande une toute autre mentalité, car au lieu d’avoir des réponses exactes on a des estimations ce qui implique qu’on doit considérer l'erreur de l’estimation et donc trouver un mécanisme pour la contrôler. L’autre difficulté provient des considérations d’arrêts avec l’utilisation de méthodes discrètes, on devait penser à un mécanisme qui ferait en sorte qu’on ne manque pas la condition d’arrêt.

Annexe A - Code de l’application Matlab

### test.m

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116  117  118  119  120  121  122  123  124  125  126  127  128  129  130  131  132  133  134  135  136  137  138  139  140  141  142  143  144  145  146  147  148  149  150  151  152  153  154  155  156  157  158  159  160  161  162  163  164  165  166  167  168  169  170  171  172  173174175176177178179180181182183184185186187188189190191192193194195196197198 | %% Premier  clear;  clc;  [ prise11 vf11 x11 y11 z11 t11 ] = Devoir2(1,[-120;0;4.55] / 3.6);  [ prise12 vf12 x12 y12 z12 t12 ] = Devoir2(2,[-120;0;4.55] / 3.6);  [ prise13 vf13 x13 y13 z13 t13 ] = Devoir2(3,[-120;0;4.55] / 3.6);  figure(01)  plot3(x11,y11,z11,'Marker','\*');  hold on  plot3(x12,y12,z12,'Marker','+');  plot3(x13,y13,z13,'Marker','o');  axis([-5 20 -1 1 0 3]);  xlabel('X');  ylabel('Y');  zlabel('N');  title('Trajectoires avec V1');  view([-63 29]);  grid on  grid minor  hold off  %Affichage 11  disp('Tir 1');  disp('Option 1');  fprintf('Prise: %d, Temps d''arret: %d \n',prise11,t11(end));  disp('Vitesse');  disp(vf11.');  disp('Position');  disp([x11(end) y11(end) z11(end)]);  disp('');  %Affichage 12  disp('Option 2');  fprintf('Prise: %d, Temps d''arret: %d \n',prise12,t12(end));  disp('Vitesse');  disp(vf12);  disp('Position');  disp([x12(end) y12(end) z12(end)]);  disp('');  %Affichage 13  disp('Option 3');  fprintf('Prise: %d, Temps d''arret: %d \n',prise13,t13(end));  disp('Vitesse');  disp(vf13);  disp('Position');  disp([x13(end) y13(end) z13(end)]);  %-------------------------------------------%  %% Tir 2%  [ prise21 vf21 x21 y21 z21 t21 ] = Devoir2(1,[-120;0;7.79] / 3.6);  [ prise22 vf22 x22 y22 z22 t22 ] = Devoir2(2,[-120;0;7.79] / 3.6);  [ prise23 vf23 x23 y23 z23 t23 ] = Devoir2(3,[-120;0;7.79] / 3.6);  figure(02)  plot3(x21,y21,z21,'Marker','\*');  hold on  plot3(x22,y22,z22,'Marker','+');  plot3(x23,y23,z23,'Marker','o');  axis([-15 20 -1 1 0 3]);  xlabel('X');  ylabel('Y');  zlabel('N');  title('Trajectoires avec V2');  view([-63 29]);  grid on  grid minor  hold off  %Affichage 21  disp('-----------------------\n-----------------------');  disp('Tir 2');  disp('Option 1');  fprintf('Prise: %d, Temps d''arret: %d \n',prise21,t21(end));  disp('Vitesse');  disp(vf21.');  disp('Position');  disp([x21(end) y21(end) z21(end)]);  disp('');  %Affichage 22  disp('Option 2');  fprintf('Prise: %d, Temps d''arret: %d \n',prise22,t22(end));  disp('Vitesse');  disp(vf22);  disp('Position');  disp([x22(end) y22(end) z22(end)]);  disp('');  %Affichage 23  disp('Option 3');  fprintf('Prise: %d, Temps d''arret: %d \n',prise23,t23(end));  disp('Vitesse');  disp(vf13);  disp('Position');  disp([x23(end) y23(end) z23(end)]);  %-------------------------------------------%  %% Tir 2%  [ prise21 vf21 x21 y21 z21 t21 ] = Devoir2(1,[-120;0;7.79] / 3.6);  [ prise22 vf22 x22 y22 z22 t22 ] = Devoir2(2,[-120;0;7.79] / 3.6);  [ prise23 vf23 x23 y23 z23 t23 ] = Devoir2(3,[-120;0;7.79] / 3.6);  figure(02)  plot3(x21,y21,z21,'Marker','\*');  hold on  plot3(x22,y22,z22,'Marker','+');  plot3(x23,y23,z23,'Marker','o');  axis([-15 20 -1 1 0 3]);  xlabel('X');  ylabel('Y');  zlabel('N');  title('Trajectoires avec V2');  view([-63 29]);  grid on  grid minor  hold off  %Affichage 21  disp('----------------------------------------------');  disp('Tir 2');  disp('Option 1');  fprintf('Prise: %d, Temps d''arret: %d \n',prise21,t21(end));  disp('Vitesse');  disp(vf21.');  disp('Position');  disp([x21(end) y21(end) z21(end)]);  disp('');  %Affichage 22  disp('Option 2');  fprintf('Prise: %d, Temps d''arret: %d \n',prise22,t22(end));  disp('Vitesse');  disp(vf22);  disp('Position');  disp([x22(end) y22(end) z22(end)]);  disp('');  %Affichage 23  disp('Option 3');  fprintf('Prise: %d, Temps d''arret: %d \n',prise23,t23(end));  disp('Vitesse');  disp(vf23);  disp('Position');  disp([x23(end) y23(end) z23(end)]);  %% Tir 3  [ prise31 vf31 x31 y31 z31 t31 ] = Devoir2(1,[-120;1.8;5.63] / 3.6);  [ prise32 vf32 x32 y32 z32 t32 ] = Devoir2(2,[-120;1.8;5.63] / 3.6);  [ prise33 vf33 x33 y33 z33 t33 ] = Devoir2(3,[-120;1.8;5.63] / 3.6);  figure(03)  plot3(x31,y31,z31,'Marker','\*');  hold on  plot3(x32,y32,z32,'Marker','+');  plot3(x33,y33,z33,'Marker','o');  axis([-15 20 -1 1 0 3]);  xlabel('X');  ylabel('Y');  zlabel('N');  title('Trajectoires avec V3');  view([-63 29]);  grid on  grid minor  hold off  %Affichage 31  disp('---------------------------------------------');  disp('Tir 3');  disp('Option 1');  fprintf('Prise: %d, Temps d''arret: %d \n',prise31,t31(end));  disp('Vitesse');  disp(vf31.');  disp('Position');  disp([x31(end) y31(end) z31(end)]);  disp('');  %Affichage 32  disp('Option 2');  fprintf('Prise: %d, Temps d''arret: %d \n',prise32,t32(end));  disp('Vitesse');  disp(vf32);  disp('Position');  disp([x32(end) y32(end) z32(end)]);  disp('');  %Affichage 33  disp('Option 3');  fprintf('Prise: %d, Temps d''arret: %d \n',prise33,t33(end));  disp('Vitesse');  disp(vf33);  disp('Position');  disp([x33(end) y33(end) z33(end)]); |

### Devoir2.m

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43 | function [ prise vf x y z t ] = Devoir2( option,vi )  dt0 = 0.02; %Delta t par defaut  x = 18.44;  y = 0;  z = 2.1;  t = 0;  ri = [x;y;z];  dt = dt0;  prise = 0;  auSol = 0;  ray = 7.3/100/2;  if option == 1  a = [0;0;-9.8];  i = 1;  while prise == 0 && auSol == 0  vn = vi + t(end)\*a;  dt = FixDTZoneCrit(dt0,vn,x,z);  tn = t(end) + dt;  r = ri + vi \* tn + 0.5 \* a \* tn^2;  x = [x r(1)];  y = [y r(2)];  z = [z r(3)];  t = [t tn];  i = i+1;  [prise, auSol] = checkEnd(r);  end  vf = vn;  else  q0 = [0 vi.' ri.'];  if(option == 2)  [qSol, prise ] = SOLRK4C(q0,dt0,'Option2');  elseif(option == 3)  [qSol, prise ] = SOLRK4C(q0,dt0,'Option3');  else  warning('Option invalide ');  end  vf = qSol(end, 2:4);  x = qSol(:,5);  y = qSol(:,6);  z = qSol(:,7);  t = qSol(:,1);  end  end |

### FixDTZoneCrit.m

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | function [ dt ] = FixDTZoneCrit( dt0,vn,x,z )  ray = 7.3 / 100 /2;  if(x(end) <= -vn(1)\*dt0\*2 && x(end)>= vn(1)\*dt0\*2 || z(end)<vn(3)\*dt0\*-2)  dt = ray / norm(vn);  else  dt = dt0;  end  end |

### checkEnd.m

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | function [ prise auSol ] = checkEnd( r )  ray = 7.3/100/2;  long = 30.48 / 2/100;  prise = 0;  auSol = 0;  if ((r(1) >= -ray/2 && r(1) <= ray/2) && ...  (r(2) + ray <= long && r(2) - ray >= -long) &&...  (r(3) + ray <= 1.8 && r(3) - ray >= 0.8))  prise = 1;  end  if (r(3)-ray < 0)  auSol = 1;  end  end |

### SOLRK4C.m

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50 | function [ qSol, prise ] = SOLRK4C( q0, dt0, fonctiong )  [qSol, prise] = SOLRK4(q0,dt0,fonctiong,0);  dtn = dt0;  n = 1;  e = 1;  while e >= 0.001 && n < 10  dtn = dtn / 2;  [q2, prise] = SOLRK4(q0, dtn, fonctiong,0);  [pos,ignore] = size(q2(:,1));  if(mod(pos,2) == 1)  pos = pos+1;  end    posPrecis = pos - 1;  pos = pos/2;  err = q2(posPrecis,5:7)-qSol(pos,5:7);  avg = q2(posPrecis,5:7)+ qSol(pos,5:7) /2.;  e = max(abs(err./avg));  qSol = q2;  n = n+1;  end      % check for collision with at x = 0    ray = 7.3 /100 /2;  reprise = 0;  lastQ = qSol(end,:);  if(~prise && lastQ(7) < ray && lastQ(5) < 0)  I = find(qSol(:,5)>0);  before0 = I(end);  v = norm(qSol(before0,2:4));  dtp = ray / v / 2; % delta t pour que le deplacement soit environ 1/4 de la balle  [qSol2, prise] = SOLRK4(qSol(before0,:), dtp, fonctiong,1);  if(prise)  reprise = 1;  qSol = [qSol(1:before0-1,:); qSol2];  end  end    % add precision at collision point  if(~reprise)  v = norm(qSol(end-1,2:4));  dtp = ray / v / 2; % delta t pour que le deplacement soit environ 1/4 de la balle  [qSol2, ignore2] = SOLRK4(qSol(end-1,:),dtp,fonctiong,0);  qSol = [qSol(1:end-2,:); qSol2];  end      end |

### SOLRK4.m

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | function [ qSol, prise ] = SOLRK4(q0, dt0, fonctiong, priseOnly )  i = 1;  prise = 0;  auSol = 0;  qSol (1 ,:) = q0;  while prise == 0 && auSol == 0 && (~priseOnly || qSol(end,5)>0)  qSol (i+1 ,:)= SEDRK4 ( qSol (i ,:) , dt0 ,fonctiong);    [prise, auSol] = checkEnd(qSol(end,5:7));  i = i+1;  end  end |

### SEDRK4.m

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | function qs= SEDRK4 (q0 , Deltat , fonctiong )  %  % Solution ED dq/dt= fonctiong (q,t)  % Methode de Runge - Kutta d� ordre 4  % qs : vecteur final [tf q(tf )]  % q0 : vecteur initial [ti q(ti )]  % Deltat : intervalle de temps  % fonctiong : membre de droite de ED.  % Ceci est un m- file de matlab  % qui retourne [1 dq/dt(ti )]  %  k1= feval ( fonctiong ,q0 )\* Deltat ;  k2= feval ( fonctiong ,q0+k1 /2)\* Deltat ;  k3= feval ( fonctiong ,q0+k2 /2)\* Deltat ;  k4= feval ( fonctiong ,q0+k3 )\* Deltat ;  qs=q0 +( k1 +2\* k2 +2\* k3+k4 )/6; |

### Option2.m

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | function g= Option2 (q0)  %  % corps en chute libre et force visqueuse  %  a = - pi \* 0.073^2 / 8 \* 1.1644 \* 1.45 \* norm(q0(2:4))\* q0(2:4) / 0.145;  a(3) = a(3) - 9.8;  g=[1 a q0(2) q0(3) q0(4)] ; |

### Option3.m

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | function g= Option3 (q0)  %  % corps en chute libre et force visqueuse et truc weird  %  vr = [0 ;0 ;50];  v = q0(2:4);  av = - pi \* 0.073^2 / 8 \* 1.1644 \* 1.45 \* norm(v)\* v / 0.145;  ag = [0 0 -9.8];  am = pi \* 0.073^2 / 8 \* 1.1644 \* 0.0016 \* norm(v) \* cross(vr,v) / 0.145 ;  a= av+ ag +am;  g=[1 a q0(2) q0(3) q0(4)] ; |

### 