

Rapport du groupe i1312
Simulation d'une cascade automobile

Moez Ben Mbarka
Guillaume Benoit
Thomas Detoux
Xavier Dorel

Professeurs:
Bernard Perrot
Afif Sellami

22 novembre 2005

Table des matières

1	Introduction	3
1.1	Énoncé et but du sujet	3
2	L'organisation du travail	4
2.1	Outils utilisés	4
2.1.1	CVS	4
2.1.2	Script de compilation	4
3	Conception	5
3.1	Structures de base	5
3.1.1	Structure inconnue :	5
3.2	Différents modules	5
3.2.1	Module : inconnue	5
3.2.2	Module : Cauchy	6
3.2.3	Module : rKutta4	7
3.2.4	Module : calculs	7
3.2.5	Modules de test :	8
4	Réalisation	9
4.1	Module : inconnue	9
4.1.1	Problème lié à l'inconnue	9
4.1.2	Les accesseurs	9
4.1.3	Les autres fonctions du module	9
4.2	Module : Cauchy	10
4.2.1	Problème et principe de résolution	10
4.2.2	La fonction calculeTrajectoire	10
4.2.3	Exemples de visualisation de la trajectoire	10
4.3	Module : Runge-Kutta 4	11
4.3.1	Problème et principe de résolution	11
4.3.2	Fonctions	11
4.3.3	Calcul de V_0 pour un impact à une distance d	11
4.3.4	Calcul de l'angle θ pour une vitesse V_0 minimale	13
4.4	Le module menu :	13
4.5	Les variables globales	14
4.6	Exemples de trajectoires	15
4.6.1	Cas limites	15
5	Manuel d'utilisation	16
6	Conclusion	17

Table des figures

4.1	Trajectoire sans vent et sans frottement	10
4.2	Trajectoire avec vent et frottements	11
4.3	Trajectoire pour $\theta = \frac{\pi}{2}$	15
4.4	Trajectoire pour $\theta = 0$	15

Chapitre 1

Introduction

1.1 Énoncé et but du sujet

Ce sujet présente une cascade où une voiture s'élance à une vitesse V_0 sur un tremplin. Le but est d'effectuer divers calculs sur la voiture et sa trajectoire sachant que cette automobile est soumise à un vent. Nous allons donc expliquer la programmation de ces divers algorithmes nécessaires pour obtenir la trajectoire, la vitesse V_0 pour atteindre une distance D ou encore l'angle θ qui minimise la vitesse.

Chapitre 2

L'organisation du travail

Nous avons travaillé la plupart du temps en binômes. Nous avons d'abord déterminé l'équation différentielle nécessaire pour les algorithmes que nous souhaitions écrire. Une fois cette étape réalisée, l'écriture des algorithmes sur feuille a commencé. Moez s'est ensuite surtout chargé de la partie implémentation avec Thomas ou Xavier. Guillaume s'est chargé de l'écriture des deux rapports avec l'aide de l'ensemble de l'équipe.

2.1 Outils utilisés

Différents outils ont été mis en place pour faciliter le travail sur ce projet. Nous tentons ici d'en faire une liste exhaustive de façon à vous permettre de juger de l'effort que nous avons mis à travailler tous ensemble.

2.1.1 CVS

Nous avons mis en place un référentiel CVS dans le but de faciliter le travail de groupe. Cela nous a permis de ne pas nous limiter sur la modification des fichiers : chacun pouvait à sa guise et quand il le voulait travailler sur n'importe quel fichier. Nous avons aussi pu bénéficier des améliorations de chacun au fur et à mesure qu'elles étaient mises en place. Comme un seul membre du groupe savait réellement se servir de CVS, il a créé une documentation au format pdf qui devrait être disponible avec les sources du projet. Chacun a donc pu facilement travailler avec CVS et tout le monde en a vu l'intérêt.

2.1.2 Script de compilation

Le script "compilation" effectue le même travail qu'un Makefile. Il permet de compiler tous les fichiers courants du projet, de créer les liens entre eux et de générer un exécutable. Pour se faire, le script effectue la compilation en vérifiant si le fichier source *.c et les *.h sont plus récents que son fichier objet *.o, si ce n'est pas le cas, la compilation ne s'effectue pas. Le même procédé est utilisé pour l'édition des liens.

Chapitre 3

Conception

Dans cette partie, nous allons décrire un à un les différents modules que nous avons développés pour découper notre travail. Nous expliquerons aussi quels sont les liens qui les unissent. Les détails concernant les champs des structures utilisées, la gestion de la mémoire, la gestion des exceptions ainsi que les prototypes des différentes fonctions seront abordés dans la partie réalisation. Nous avons essayé d'uniformiser au maximum notre travail tout au long de la réalisation du projet.

3.1 Structures de base

3.1.1 Structure inconnue :

Une inconnue peut être représentée sous la forme d'une matrice à 4 lignes et une seule colonne contenant des valeurs réelles. Pour stocker ces valeurs, nous avons créé une structure à quatre champs, chacun stockant une valeur. Toutes ces inconnues sont ensuite stockées dans un tableau.

3.2 Différents modules

3.2.1 Module : inconnue

Description

Ce module sert à stocker les quatre valeurs qui définissent une inconnue et les diverses actions associées à l'inconnue. Ces valeurs sont contenues dans une structure qui permet une manipulation aisée des différentes valeurs.

Routines publiques

- **Les accesseurs**

Rôle : Accéder à un champ ou remplir un champ de la structure inconnue.

Entrée : Une structure inconnue (respectivement une structure inconnue et une valeur)

Sortie : Une valeur (respectivement une structure avec une nouvelle valeur)

- **Créer inconnue**

Rôle : Créer une structure inconnue

Entrée : Quatre valeurs correspondant aux futures valeurs de la structure inconnue.

Sortie : Une structure inconnue ayant ses quatre champs de rempli.

– **Produit scalaire inconnue**

Rôle : Effectuer le produit scalaire d'un réel par une inconnue.

Entrée : Une structure inconnue X et un réel k.

Sortie : Le produit scalaire de X par k et donc un réel.

– **Somme inconnue**

Rôle : Sommer un nombre non connu d'inconnues.

Entrée : Plusieurs inconnues dont tous les champs sont initialisés.

Sortie : Une inconnue dont chaque champ correspond à la somme de tous les mêmes champs des inconnues placées en paramètres.

3.2.2 Module : Cauchy

Description

Le module Cauchy a pour but de calculer la trajectoire de la voiture à partir de différentes données. Cette trajectoire s'obtient à partir de diverses opérations sur les inconnues qui retournent aussi des inconnues. Ces inconnues sont ensuite stockées dans un tableau qui permet ensuite de définir la trajectoire.

Routine publiques

– **cauchy**

Rôle : La fonction cauchy correspond à une fonction F où l'image d'une inconnue est sa dérivée temporelle "membre à membre". Cette fonction sera souvent ensuite appelé par un pointeur de fonction.

Entrée : Une inconnue dont les champs sont remplies.

Sortie : Une inconnue dont les champs ont été substitués par les valeurs correspondant à une dérivation temporelle (pour se faire, nous nous sommes servis de la relation fondamentale de la dynamique).

– **Test arrêt**

Rôle : Cette petite fonction sert à déterminer l'arrêt de la fonction "calcul trajectoire" en vérifiant si la valeur selon l'axe des Y est inférieur ou égal à zéro ou non.

Entrée : Une inconnue dont les champs sont remplies.

Sortie : Un booléen valant Vrai si Y est négatif ou nul et Faux sinon.

– **Calcule Trajectoire**

Rôle : Cette fonction a pour but de calculer la trajectoire qui sera stockée dans un tableau sous la forme de structure d'inconnues. Ce tableau pourra ensuite permettre de tracer la trajectoire avec le logiciel Gnuplot.

Entrée : Un tableau d'inconnues, le pas qui servira pour la détermination des valeurs des inconnues et enfin l'erreur qui permet d'arrêter la boucle "while" lorsque le taux d'erreur est atteint.

Sortie : Un entier indiquant le nombre d'éléments dans le tableau d'inconnues.

3.2.3 Module : rKutta4

Description

Le module concernant la méthode de Runge Kutta d'ordre quatre permet de calculer, point par point, les valeurs prises par x , y , la vitesse selon l'axe des x et la vitesse selon l'axe des y . Ce module est nécessaire car les fonctions présentes sont appelées par la fonction "Calcule Trajectoire" du module précédent.

Routines publiques

– **Calcule suivant**

Rôle : Calculer l'élément suivant d'un élément donné grâce à la méthode Runge Kutta

Entrée : Une inconnue (dont on cherche l'élément suivant), le pas et un pointeur de fonction (qui pointera vers la fonction "cauchy").

Sortie : Une inconnue représentant l'élément suivant calculé par la méthode de Runge Kutta.

– **rKutta4**

Rôle : Stocker les différentes inconnues ayant été calculées dans un tableau.

Entrée : Un tableau d'inconnues, le pas de discrétisation, la première inconnue (nécessaire pour calculer la suite des autres inconnues), un pointeur de fonction (vers la fonction "cauchy" quand cette fonction sera appelé et un pointeur vers la fonction testArret.

Sortie : Après avoir rempli, on renvoie un entier correspondant au nombre de cases occupées dans le tableau.

3.2.4 Module : calcules

Description

Ce module contient les fonctions qui répondent aux deux dernières questions du sujet du projet. A savoir : le calcul de la vitesse initiale nécessaire pour que l'impact de la voiture avec le sol soit à une distance d , et l'angle θ pour que cette vitesse soit minimal. Pour plus de détails sur les idées algorithmiques pour effectuer les calculs, se référer à la partie réalisation(4.3.2).

Routine publiques

– **determineV0 :**

Rôle : Calculer la vitesse V_0 pour un impact à une distance d donnée.

Entrée : la distance d , le pas de discrétisation, l'erreur tolérée.

Sortie : la vitesse trouvée.

– **determineTéta :**

Rôle : Calculer l'angle du tremplin pour une vitesse minimale

Entrée : la distance d , le pas de discrétisation, l'erreur tolérée.

Sortie : l'angle trouvée en radian.

3.2.5 Modules de test :

Un module de test unitaire a été réalisé pour le module *inconnue*. Pour les autres modules de calcul de la trajectoire, le test a été effectué en comparant les résultats obtenus avec des résultats théoriques, ou encore en vérifiant ces résultats pour des cas limites (exemples : $\theta = 0$, $\theta = \Pi$, vitesse de vent nulle...). Pour certains de ces cas la trajectoire est donnée dans la partie exemples de trajectoire(4.6) .

Fonction de test pour le module inconnue

test_Lecturecoordonnees : Elle teste la lecture des coordonnées d'une inconnue(*lire_u,...*).

test_modif_coordonnees : Elle teste la modification des valeurs des coordonnées(*ecrire_u...*).

test_somme2inconnues : Elle teste le résultat de la somme de deux inconnues.

test_sommePlusInconnues : Elle teste la somme de plusieurs inconnues.

test_produit : Elle teste le résultat du produit d'une inconnue par un scalaire.

Chapitre 4

Réalisation

Dans cette partie nous décrivons comment sont implémentées les fonctions et les structures.

4.1 Module : inconnue

4.1.1 Problème lié à l'inconnue

Une inconnue est composé de quatre valeurs. Le meilleur moyen de stocker ces valeurs reste une structure ayant quatre champs, chaque champ correspondant à une valeur réelle.

4.1.2 Les accesseurs

Prototypage de la fonction pour obtenir une valeur :

```
float lire_x (inconnue *X)
```

où x représente un des champs de la structure inconnue (donc les différentes valeurs de x sont u, v, w ou z).

Prototypage de fonction pour écrire une valeur :

```
bool ecrire_x(inconnue *X, float x)
```

x peut prendre les mêmes valeurs que précédemment et la fonction présente un effet de bord en indiquant si l'écriture s'est bien passé en renvoyant un booléen.

4.1.3 Les autres fonctions du module

- **La fonction creer_inconnue** Cette fonction utilise les accesseurs créés pour fournir une structure dont les champ seront remplis selon les valeurs données en paramètres.

Prototypage de la fonction :

```
inconnue creer_inconnue(float u, float v, float w,  
float z)
```

- **La fonction produit scal_inconnue.** Cette fonction se sert de la fonction "creer inconnue" pour faire le produit scalaire d'un "vecteur" par un réel.

Prototypage de la fonction :

```
inconnue prod_scal_inconnue (float a, inconnue X)
```

- **La fonction somme_inconnue** Cette fonction effectuer la somme d'un nombre variable d'inconnues. On doit donc se servir des "va-start", "va-list", "va-arg" et "va-end" pour effectuer l'implémentation de cette fonction.

```
inconnue somme_inconnue(int nb, ...)
```

4.2 Module : Cauchy

4.2.1 Problème et principe de résolution

La but de la première question du sujet est de déterminer la trajectoire d'une voiture à partir des données environnementales et initiales. La recherche de la trajectoire s'effectue point par point grâce à la méthode de calcul de Runge Kutta d'ordre 4. Chaque point calculé est ainsi stocké dans une structure inconnue qui sera elle-même contenue dans un tableau qui contiendra donc toutes les valeurs calculés. On calcule les points de la trajectoire tant que la condition d'arrêt est vérifiée (en l'occurrence, lorsque Y devient négatif). Ainsi on obtient tous les points de la trajectoire car celle-ci s'arrête lorsque la voiture touche le sol.

4.2.2 La fonction calculeTrajectoire

Prototypage de la fonction :

```
int calculeTrajectoire (inconnue echantillonage[], float pas, float
erreur)
```

Cette fonction ne renvoie pas le tableau d'inconnues mais un entier servant à déterminer le nombre de cases occupées.

4.2.3 Exemples de visualisation de la trajectoire

Sous Gnuplot, on peut visualiser la trajectoire. Voici deux exemples : Sans vent ni frottement, la trajectoire présente sur la première figure la forme d'une parabole.

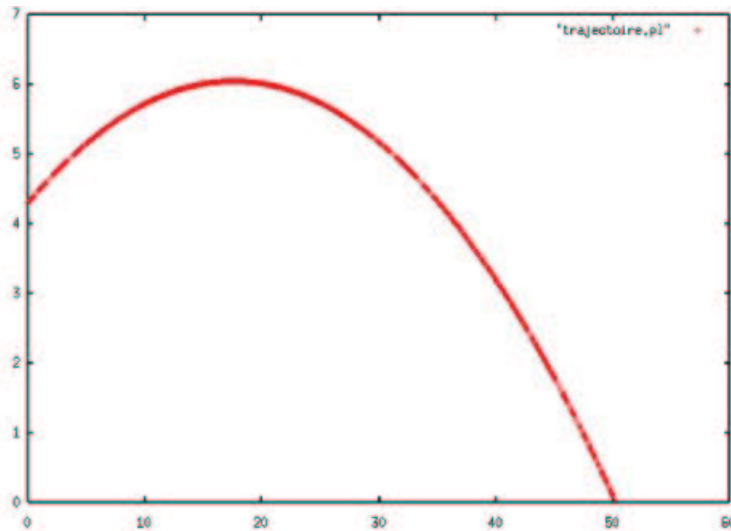


FIG. 4.1 – Trajectoire sans vent et sans frottement

A contrario, lorsque le vent et les frottements sont forts, la voiture ralentit rapidement comme l'illustre la seconde figure.

Pour plus d'exemples de trajectoire voir la partie ??.

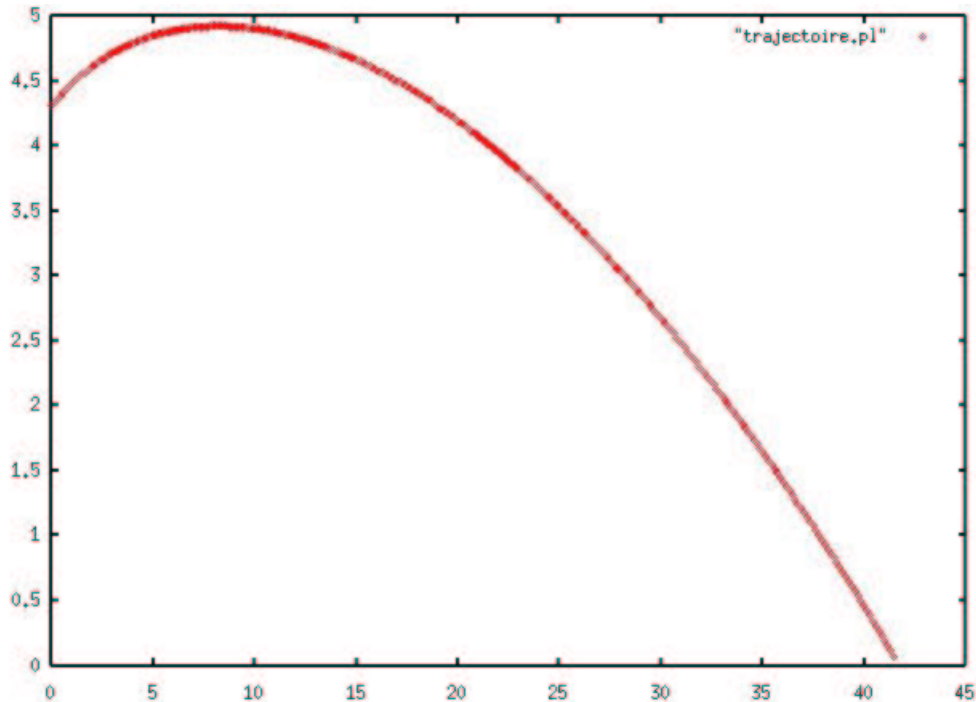


FIG. 4.2 – Trajectoire avec vent et frottements

4.3 Module : Runge-Kutta 4

4.3.1 Problème et principe de résolution

Ce module illustre la méthode de Runge-Kutta d'ordre 4 afin de réaliser le calcul de la trajectoire. L'algorithme de résolution a donc été implémenté dans ce module.

4.3.2 Fonctions

- La fonction `calculerSuivant` Prototypage de la fonction :
`inconnue calculerSuivant (inconnue precedent, float pas, inconnue (*f) (inconnue))`

Cette fonction calcule le terme suivant par la méthode de Runge-Kutta en utilisant un pointeur de fonction vers la fonction Cauchy.

- La fonction `rKutta4` Prototypage de la fonction :
`int rKutta4 (inconnue echantillonage[], float pas, inconnue initial, inconnue (*f)(inconnue), int (*testArret)(inconnue))`

Cette fonction calcule l'ensemble des termes grâce à la méthode de Runge Kutta et stocke ces valeurs dans un tableau d'inconnues. Le programme ne s'arrête que lorsque le tableau est plein ou que le test d'arrêt est vérifié.

4.3.3 Calcul de V_0 pour un impact à une distance d

Problème et principe de la résolution

Le but de la deuxième question du sujet est de calculer la vitesse initiale V_0 nécessaire pour que la voiture atteigne la distance d avant de toucher le sol. Pour résoudre le problème il faut remarquer que la distance d est proportionnelle à

la vitesse recherchée. Une méthode algorithmique naturelle c'est la *dichotomie*. En effet, on commence la recherche dans un intervalle limite (On fait une l'hypothèse que la vitesse rechercher sera dedans). On compare la distance de l'impact Y correspondant au milieu de cet intervalle à d , et selon qu'elle est plus grand ou plus petite on recommence avec la partie supérieure ou inférieure de l'intervalle initial. Ce procédé est réitéré tant que la condition suivante est vérifiée :

$|Y - d| > \epsilon$ et $|V1 - V2| > \epsilon$, où *epsilon* est l'erreur, $V1$ et $V2$ les bornes de l'intervalle de la dichotomie.

Clairement, la premier partie de la condition est naturelle : on teste le rapprochement de la valeur calculée à la valeur désirée. Quand à la deuxième, elle est là pour éliminer toute possibilité de divergence numérique de l'algorithme. En effet, bien que la relation de proportionnalité entre d et $V0$ parait évidente, son degré n'est pas aussi explicite. Il dépend des autres paramètres du programme. Par exemple : dans le cas d'un vent dans le sens de la cascade la distance d est très sensible à $V0$, tandis que ce n'est pas le cas pour un vent dans le sens contraire.

Bien entendu, le faite que l'algorithme peut terminer à cause de la deuxième condition, ne prouve pas sa correction, le seul inrérêt est de gérer le cas la largeur de l'intervalle de la dichotomie diminue beaucoup plus rapidement que la distance d .

Prototype de la fonction :

```
float determineV0(float d, float pas, float epsilon)
```

Effet de bord :

Le seul effet de bord causé par cette fonction est quelle change la valeur de la variable globale $V0$. En effet, celle-ci prend la valeur trouvée par la fonction. Ce choix est dû qu'en phase de test ceci facilite de vérifier immédiatement la correction de la valeur trouvée en lançant le calcul de la trajectoire.

Exemples

Pour les paramètres suivants :

```
m=500.000000 g=9.810000 Cf=0.600000
Cv=5.000000 V=10.000000 V0=50
L=20.000000 teta=0.098125 h=0.400000
Pas=0.010000 epsilon=0.050000
```

On obtient :

d(m)	V_0 (km/h)
45	87.48
60	188.32
100	269.89

Si on inverse le sens du vent : $V = -10m/s$ on trouve des vitesses initiales plus importantes :

d(m)	V_0 (km/h)
45	167.76
100	300.92

4.3.4 Calcul de l'angle téta pour une vitesse V_0 minimale

Problème et principe de la résolution

Le but ici est de calculer l'angle θ qui minimise la vitesse V_0 nécessaire pour atteindre une distance d . Cette fois, la relation entre les deux grandeurs θ et v_0 n'est pas une relation de proportionnalité simple. Une étude plus approfondie peut montrer que la courbe $V_0 = f(\theta)$ a la forme d'une parabole, la valeur recherchée étant le minimum de cette parabole. Les grands lignes de l'algorithme sont les suivantes :

1. Comme pour le problème précédent, on commence par un intervalle $[\theta_0, \theta_n]$ en supposant que la valeur recherchée est dedans.
2. On divise l'intervalle en n sous intervalles, en calculant pour chacun la différence entre les deux valeurs V_0 trouvées (en lançant la fonction `determineV0`), pour θ égale aux extrémités de l'intervalle.
3. θ_0 prend comme valeur celle de la plus grande extrémité pour laquelle la différence est négative (Cette extrémité est dans la partie décroissante de la courbe). θ_n prend la valeur de la plus petite extrémité pour laquelle la différence est positive (la partie croissante de la courbe).
4. On recommence à l'étape 1 tant que la largeur de l'intervalle est supérieure à ϵ .

Il est à remarquer que cette fonction peut prendre comme paramètre n : le nombre de partitions sur l'intervalle initiale. En effet, le choix initial était 3 (c'est le nombre minimal de partitions nécessaires), mais on s'est rendu compte en faisant les tests que le fait d'augmenter le nombre de partitions accélère considérablement le fonctionnement de la fonction. Surtout, qu'il est à noter que cette fonction est la plus gourmande en terme de temps d'exécution dans le projet, conséquence du fait qu'elle lance tant d'appels pour la fonction `determineV0` qu'elle génère d'extrémités d'intervalles.

fonction auxiliaire :

Pour gérer la partition de l'intervalle en n sous intervalle, on utilise un tableau de dimension n . Une fonction auxiliaire `remplir_tab_teta` est utilisée. son rôle est de partitionner un intervalle en n parties égales. Son prototype est :

```
void remplir_tab_teta(float tab_teta[], float t0, float tn, int n)
```

prototype de la fonction `determineTeta`

```
float determineTeta(float d, float pas, float epsilon)
```

Effet de bord

Pour les mêmes raisons que pour la fonction `determineV0`, La valeur globale téta garde la valeur trouvée par la fonction.

4.4 Le module menu :

Ce module a été réalisé pour faciliter l'utilisation des différents modules de calculs.

Des détails sur les options fournies par le menu sont données dans la partie manuel d'utilisation(5). Il est implémenté à l'aide d'une structure menu :

```
typedef struct Menu_Elem_  
{  
    char *description;    /**< Le nom de l'élément. */  
    void (*fonc) (void); /**< Un pointeur vers la fonction qui va  
                           exécuter l'opération demandée. */  
  
    unsigned int niveau; /**< Le niveau auquel appartient l'élément. */  
}  
*Menu_Elem;
```

Chaque élément de la structure est défini par son description (le titre qui s'affiche à l'écran), un pointeur vers la fonction à exécuter si l'option est sélectionnée et le niveau pour lequel cette option est disponible. une variable globale Niveau détermine à chaque instant le niveau du menu.

4.5 Les variables globales

Tout les modules du projets (sauf le module inconnues) utilisent en commun les paramètres de la cascade :

m, g, Cf, Cv, V, V0, L, teta, h, Pas et epsilon.

D'où le choix que ces paramètres soient des variables globales accessibles par tous les modules.

4.6 Exemples de trajectoires

4.6.1 Cas limites

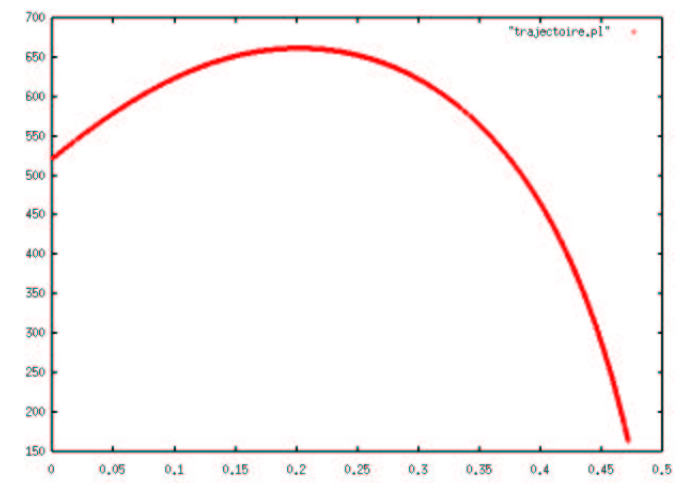


FIG. 4.3 – Trajectoire pour $\theta = \frac{\pi}{2}$

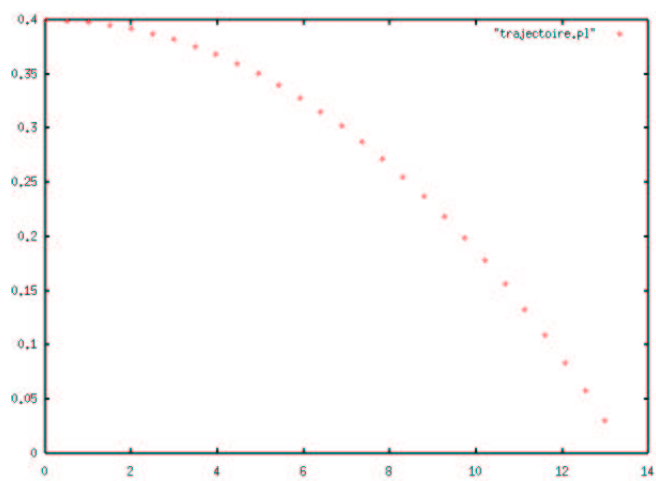


FIG. 4.4 – Trajectoire pour $\theta = 0$

Chapitre 5

Manuel d'utilisation

On va essayer ici de présenter la façon la plus simple pour utiliser le programme. La première étape est évidemment la compilation. Un script de compilation est fourni avec les sources. Il suffit de l'exécuter :

```
.\compilation
```

Un exécutable nommé *cascade* est généré. En l'exécutant le menu principal apparaît. Les options disponibles permettent de calculer la trajectoire (pour les paramètres par défaut affichés au dessus du menu), de lancer le calcul de V_0 ou de θ , de modifier les paramètres, et bien sûr de quitter le programme. Le menu a été créé par niveaux. Par exemple la façon de lancer le calcul de la trajectoire, incrémente le niveau du menu, et des nouvelles options apparaissent : afficher la trajectoire et exporter la trajectoire.

La première permet d'afficher la trajectoire sur l'écran en 4 listes verticales qui représentent successivement en allant de gauche à droite :

L'abscisse X , l'ordonnée Y , \dot{X} , \dot{Y} . la deuxième option offre la possibilité d'enregistrer la trajectoire (avec la même forme précédente) dans un fichier dont le choix du nom est laissé à l'utilisateur. Cette option permet après de visualiser la courbe à l'aide de GNUPLOT .

Une autre option principale c'est : *modifier les paramètres*. Elle déplace le menu à un nouveau niveau. Les options qui y sont disponibles permettent de modifier tous les paramètres ou un certain nombre. A ce niveau les options précédentes disparaissent. cependant, l'option *Retour* permet de retourner au niveau précédent.

Chapitre 6

Conclusion

Dans ce projet, nous avons appris à résoudre un problème qui paraissait difficile qui fut simplifié par la méthode point par point utilisé en algorithmique numérique. Ainsi, cette cascade a pu être simulée en C et l'obtention visuelle de la trajectoire grâce au logiciel Gnuplot a apporté un aspect concret. De plus, le fait d'avoir du rédiger le rapport d'analyse en anglais nous a permis de travailler encore plus en groupe. Enfin la plupart des problèmes rencontrés lors de l'implémentation ont été résolus et tous les algorithmes marchent.