

## RAPPORT DE STAGE

# MODÉLISATION NUMÉRIQUE D'OSCILLATEURS NON-LINÉAIRES POUR LA RÉCUPÉRATION D'ÉNERGIE VIBRATOIRE

---

LÉGLISE Cloé  
Polytech Annecy-Chambéry - SNI4  
Année universitaire 2022/2023

Tutrice de stage  
SAINT-MARTIN Camille

Tuteur école  
ATTO Abdourrahmane

Je tiens à remercier Camille SAINT-MARTIN, ma tutrice de stage, pour m'avoir offert cette opportunité et m'avoir épaulée durant la totalité de ce stage.

Je souhaite également remercier chaleureusement Ludovic CHARLEUX pour son aide, et particulièrement pour la relecture du présent rapport.

Je suis aussi reconnaissante envers Abdourrahmane ATTO d'avoir pris le temps, en tant que tuteur école, de rechercher et m'envoyer des ressources, alors que ce n'était pas dans ses obligations.

## Résumé

Ce stage a été réalisé dans le cadre d'un projet de recherche sur la récupération d'énergie vibratoire. Les récupérateurs d'énergie pourraient être une alternative plus propre aux batteries au lithium, à durée de vie limitée et très polluantes. Le but de ce stage est de réaliser une application avec interface graphique permettant de modéliser le comportement d'oscillateurs mécaniques et de pouvoir observer en temps réel l'impact de certains paramètres sur ceux-ci. Cette application a été réalisée en Julia, un langage informatique particulièrement adapté puisqu'il permet de créer des animations de façon facile et intuitive. Au moment de la rédaction de ce rapport, l'application n'est pas encore terminée. Le plus gros de ce stage a consisté en la recherche d'outils les plus adaptés possibles et la maîtrise du langage Julia. Les deux semaines de stage restantes seront donc consacrées à l'assemblage des différents éléments afin de créer une application cohérente, qui consistera en une fenêtre principale permettant de lancer différentes animations.

## Mots-clefs

Récupération d'énergie vibratoire, modélisation, analyse numérique, dynamique non-linéaire, développement d'application.

# Table des matières

<b>1</b>	<b>Introduction</b>	<b>4</b>
<b>2</b>	<b>Contexte du stage</b>	<b>5</b>
2.1	Présentation de l'organisme d'accueil . . . . .	5
2.2	Sujet de stage . . . . .	6
<b>3</b>	<b>Contexte scientifique</b>	<b>8</b>
3.1	Les récupérateurs d'énergie . . . . .	8
3.2	Les sauts d'orbite . . . . .	10
3.3	La place de ce stage . . . . .	11
<b>4</b>	<b>Contenu du stage</b>	<b>12</b>
4.1	Travail effectué . . . . .	12
4.1.1	Interface graphique . . . . .	13
4.1.2	Systèmes dynamiques . . . . .	17
4.2	Perspectives . . . . .	22
<b>5</b>	<b>Conclusion</b>	<b>24</b>
<b>6</b>	<b>Bibliographie</b>	<b>25</b>
<b>7</b>	<b>Annexes</b>	<b>28</b>

# 1 Introduction

Alors que les enjeux climatiques deviennent de plus en plus importants, il est urgent de chercher des alternatives moins polluantes aux sources d'énergie utilisées aujourd'hui. Les récupérateurs d'énergie vibratoire pourraient être une de ces alternatives. Les récupérateurs d'énergie fonctionnent sur certaines bandes de fréquence sur lesquelles différentes sortes d'orbites peuvent cohabiter. Ces orbites sont des cycles limites qui peuvent être chaotiques, ou stables, et hautes, ou basses. Les orbites hautes présentent un plus grand intérêt pour la récupération d'énergie parce qu'elles représentent une plus grande puissance [6]. Afin de récupérer un maximum d'énergie, il est intéressant de mettre en place une stratégie de saut d'orbites : en donnant un peu d'énergie potentielle au système, celui-ci peut passer d'une orbite basse à une orbite haute [3], ce qui permet d'utiliser des bandes de fréquences présentant une cohabitation des deux types d'orbites sans perdre d'énergie.

Le projet de recherche dans lequel s'inscrit ce stage cherche à mieux comprendre et prédir le comportement des oscillateurs mécaniques utilisés pour la récupération d'énergie, afin de pouvoir mieux les exploiter. Dans ce but, la création d'une application permettant de naviguer entre différentes animations modélisant des comportements d'un oscillateur mécanique permettrait d'avoir un accès facile et rapide à ces informations, en plus de son intérêt pédagogique lors d'une éventuelle utilisation dans un contexte de vulgarisation. Ce stage a été réalisé dans le but de créer une telle interface. Il a pris place au laboratoire SYMME, sous la tutelle d'une doctorante appartenant à l'équipe de recherche sur la récupération d'énergie. L'application a été réalisée en langage informatique Julia pour répondre aux exigences de l'équipe de recherche.

La première partie de ce rapport présentera le contexte et le laboratoire dans lequel ce stage a pris place. Ensuite, la deuxième partie exposera le contexte scientifique autour de la récupération d'énergie vibratoire. Enfin, une troisième partie explicitera le travail réalisé durant le stage en lui-même.

## 2 Contexte du stage

### 2.1 Présentation de l'organisme d'accueil

L'Université Savoie Mont Blanc (USMB) est un établissement de 15 000 étudiants ouvert sur l'Europe et le monde. Les recherches sont menées par des laboratoires labellisés et reconnus, en partenariats étroits avec de grands organismes (CNRS, CEA, INRA), des organisations internationales (CERN) ou d'autres structures (INES, « Institut de la Montagne ») à la pointe de l'innovation. Le SYMME (« Systèmes et Matériaux pour la Mécatronique ») est l'un de ces laboratoires. Il a été créé en 2006 pour renforcer la position stratégique de l'université dans le domaine de la mécatronique. Il emploie environ 80 chercheurs, personnel administratif et étudiants. Ses recherches dans le domaine des microsources d'énergie concernent notamment le développement de structures électromécaniques innovantes aux échelles centimétrique et millimétrique, capables de convertir des énergies mécaniques en énergie électrique. Ces travaux recouvrent les aspects mécaniques et électroniques dans une approche multiphysique globale et cohérente. Plus particulièrement, ses activités de recherche portent sur les transducteurs piézoélectriques, électromagnétiques et électrostatiques pour la conversion d'énergie, sur des oscillateurs mécaniques linéaires et non-linéaires pour élargir la bande de fréquence des générateurs, et sur des interfaces électriques pour le conditionnement de l'énergie et le suivi de fréquences. Ses travaux ont fait l'objet de nombreuses communications dans des journaux et des conférences de référence dans le domaine et sont reconnus par la communauté scientifique.

Les travaux au laboratoire SYMME suivent deux axes principaux : « Matériaux, Systèmes et Instrumentation Intelligents », et « Qualité Industrielle ». Ces deux thèmes sont eux-mêmes divisés en trois thèmes chacuns, respectivement : « Matériaux et nanomatériaux fonctionnels », « Instrumentation pour le médical », et « Valorisation de micro-sources d'énergies ambiantes »; et « Caractérisation et modélisation thermomécanique des matériaux », « Optimisation produit-procédé » et « Optimisation des processus ». La figure 1 montre l'organisation du laboratoire SYMME.

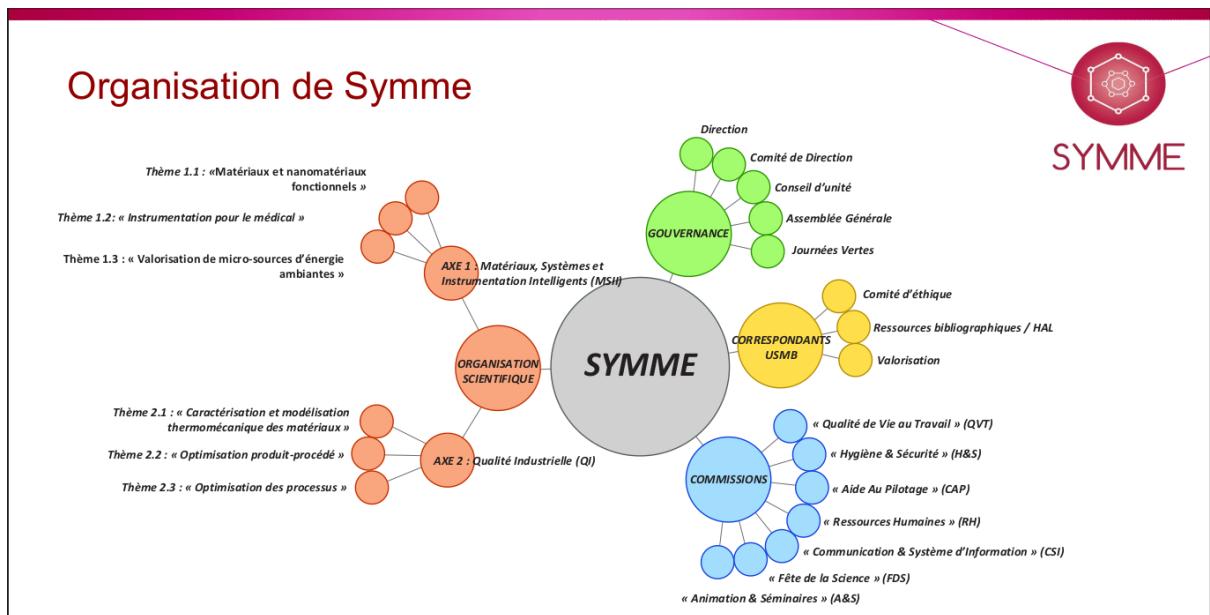


FIGURE 1 – Organisation du laboratoire SYMME.

C'est dans le thème « Matériaux, systèmes et instrumentation intelligents - valorisation de micro-sources d'énergies ambiantes » que s'inscrit ce stage.

## 2.2 Sujet de stage

Le déploiement de réseaux de capteurs sans fil communicants nécessite l'utilisation de batteries qui ont une durée de vie limitée. Le domaine de la récupération d'énergie vibratoire vise à remplacer, ou du moins compléter, l'usage de ces batteries au moyen de systèmes innovants de récupération et de stockage d'énergie robustes et fiables. Un récupérateur d'énergie vibratoire est généralement composé d'un oscillateur mécanique, de transducteurs électromécaniques (par exemple des transducteurs piézoélectriques) pour la conversion de l'énergie mécanique des vibrations en énergie électrique, et d'un circuit d'extraction électrique. Les premiers travaux dans le domaine utilisaient des récupérateurs linéaires, c'est-à-dire ceux composés d'un oscillateur linéaire. Cependant, de tels récupérateurs présentent une bande de fréquences étroite et ne sont donc pas adaptés à un environnement présentant un spectre de vibration riche et varié. Une solution prometteuse pour élargir la bande de fréquences est de considérer un oscillateur non-linéaire. Cependant, les comportements des récupérateurs non-linéaires sont plus difficiles à prédire de par l'existence de plusieurs solutions pour une fréquence de vibration donnée. Une analyse approfondie est nécessaire pour une compréhension complète de la dynamique. Dans le cadre du stage, le candidat sera en charge de modéliser un récupérateur non-linéaire développé au sein du laboratoire SYMME et se familiarisera avec les outils numériques déjà mis en place.

L'objectif de ce stage est de développer une interface graphique pour l'analyse d'influence des paramètres du modèle. La dynamique pourra être visualisée sur l'interface. Le projet open-source sera développé en Python ou en Julia. Le stagiaire définira les fonctionnalités de l'application et passera à la phase de développement de l'application. Par

exemple, il faudra faire en sorte que l'utilisateur puisse choisir un modèle de récupérateur d'énergie vibratoire, définir le type d'excitation, changer les valeurs numériques des paramètres et choisir les variables qu'il souhaite visualiser. Le stagiaire utilisera un outil de développement logiciel open-source de son choix (Gitlab, GitHub) afin d'assurer une intégration continue et associer une documentation (manuel d'utilisation) à l'application. Durant le stage, le candidat développera ses compétences en développement logiciel et modélisation numérique. Il sera intégré dans une équipe ayant une expertise dans la modélisation, conception et développement de solutions pour la récupération d'énergie, au laboratoire SYMME (Annecy). Selon l'avancement du stage, l'application développée par le stagiaire pourra faire l'objet d'une publication d'un article scientifique.

### 3 Contexte scientifique

#### 3.1 Les récupérateurs d'énergie

L'utilisation de récupérateurs d'énergie permet de récolter de l'énergie d'un gisement vibratoire pour la convertir en électricité. On les décompose en plusieurs sous systèmes comme le montre la figure 2 :

1. Un oscillateur mécanique dont la masse sismique est mise en mouvement dans le référentiel de l'oscillateur par l'accélération ambiante.
2. Un transducteur électromécanique qui prélève une partie de l'énergie mécanique disponible dans l'oscillateur et l'injecte dans le circuit électrique.
3. Un système de stockage qui assure les tâches de conditionnement et de stockage de l'énergie.

Certains systèmes reposent sur des oscillateurs linéaires d'un point de vue mécanique. Ces derniers ne peuvent récupérer une puissance importante qu'au sein d'une bande passante très étroite. Les récupérateurs d'énergie vibratoire non linéaires ont une bande passante bien plus larges comme le montre la figure 3, bien que la complexité de leurs comportements puisse les rendre difficiles à analyser. Dans le contexte de ce stage, nous nous intéressons à une technologie particulière de récupérateurs d'énergie vibratoire. Celle-ci est notamment développée au laboratoire SYMME au travers de plusieurs projets de recherche. Ils sont dits bistables car leur énergie potentielle mécanique comporte deux puits qui correspondent à deux positions stables comme le montre la figure 4. L'extraction de l'énergie mécanique y est réalisée par des cellules piézoélectriques et alimentent une charge simple qui sera assimilée à une résistance, dénotée  $R$ .

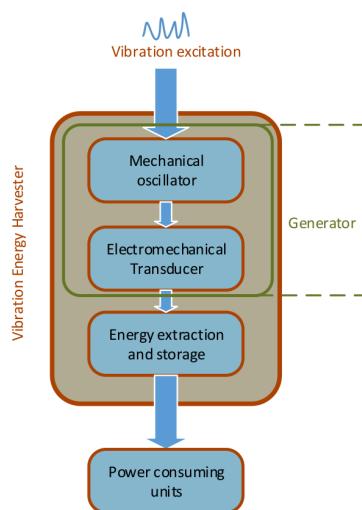


FIGURE 2 – Décomposition d'un système de récupération d'énergie vibratoire en plusieurs sous-systèmes. Ce stage se concentre sur la partie « Generator », c'est-à-dire l'oscillateur mécanique et le transducteur électromécanique. La charge électrique est considérée comme une simple charge résistive. Source : [5].

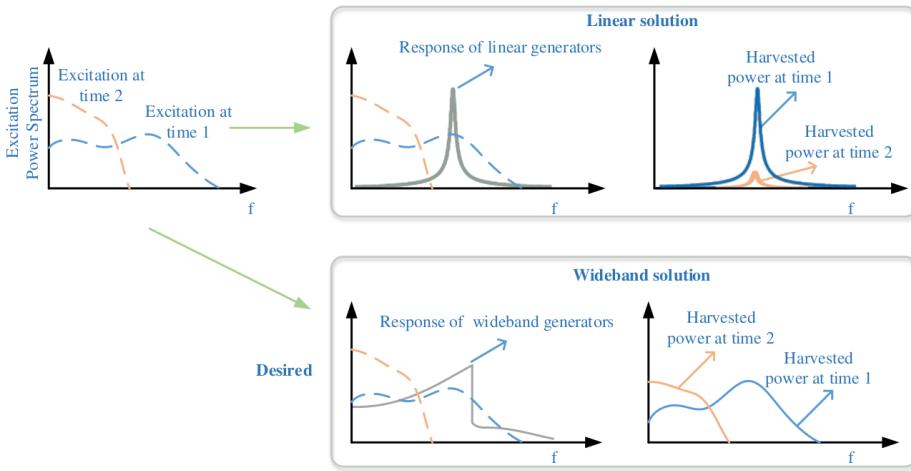


FIGURE 3 – Présentation des avantages des oscillateurs non-linéaires dans le cadre de la récupération d'énergie vibratoire. En haut à gauche, deux paysages vibratoires sont représentés par leur densité spectrale d'énergie. Le signal orange présente une grande accélération à basse fréquence et le bleu une accélération répartie sur une plus grande plage de fréquences avec un pic marqué. Dans le cadre « Linear solution », les performances d'un oscillateur linéaire sont présentées. Celui-ci est optimisé pour le signal bleu et sa fréquence propre est donc ajustée à l'endroit du pic. La réponse en puissance est représentée sur la droite du cadre et la réponse bleue est très bonne. Lorsque ce même oscillateur est soumis au signal orange, la réponse est très mauvaise. En dessous, dans le cadre « Wideband solution », la même démarche est effectuée avec un oscillateur non-linéaire. Il est optimisé pour le signal bleu pour lequel il fournit une bonne puissance. Par contre, il s'adaptera aussi relativement bien au signal orange. Cela illustre le caractère large bande du système non-linéaire. Source : [5].

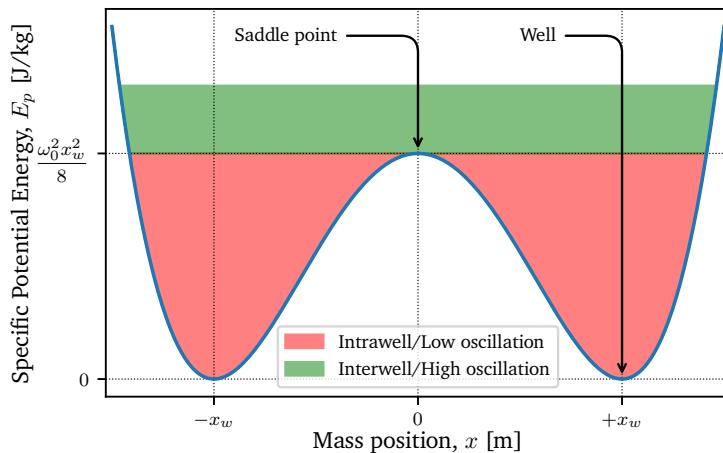


FIGURE 4 – Représentation schématique de l'énergie potentielle des systèmes de récupération d'énergie vibratoire développés au SYMME en fonction du déplacement de la masse  $x$ . Le potentiel présente deux minimums qui sont aussi des positions d'équilibre en  $x = \pm x_w$ . Un point col est présent en  $x = 0$ . Les zones rouges de basse énergie représentent le lieu des orbites dites basses. La zone verte de haute énergie est celui des orbites hautes. Source : [1].

Dans ce qui suit, nous faisons aussi l'hypothèse que les oscillateurs sont soumis à une accélération ambiante  $\ddot{x}_d$  sinusoïdale :

$$\ddot{x}_d(t) = A \sin(\omega_d t) \quad (1)$$

Où  $A$  désigne l'amplitude de l'accélération et  $f_d = \omega_d/2\pi$  sa fréquence. La figure 6 schématise l'oscillateur présenté sur la figure 5 [7]. Le récupérateur dans son ensemble peut donc être décrit par les équations suivantes :

$$\begin{cases} \dot{x} = -\frac{\omega_0^2}{2} \left( \frac{x^2}{x_w^2} - 1 \right) x - \frac{\omega_0}{Q} \dot{x} - 2 \frac{\alpha}{ML} x v + A_d \sin(2\pi f_d t) \\ \dot{v} = 2 \frac{\alpha}{LC_p} x \dot{x} - \frac{1}{RC_p} v \end{cases} \quad (2)$$

Où  $x$ ,  $\dot{x}$  et  $v$  représentent le déplacement et la vitesse de la masse et la tension générée par piézoélectricité. Et celles-ci peuvent être réécrites au premier ordre sous la forme suivante :

$$\dot{X} = f(X, t) = \begin{bmatrix} \dot{x} \\ -\frac{\omega_0^2}{2} \left( \frac{x^2}{x_w^2} - 1 \right) x - \frac{\omega_0}{Q} \dot{x} - 2 \frac{\alpha}{ML} x v + A_d \sin(2\pi f_d t) \\ 2 \frac{\alpha}{LC_p} x \dot{x} - \frac{1}{RC_p} v \end{bmatrix} \quad (3)$$

Les récupérateurs d'énergie non linéaires peuvent présenter plusieurs cycles limites stables appelés orbites et ce pour chaque condition d'excitation ( $f_d, A$ ). L'existence des ces dernières est donc fonction la fréquence d'excitation  $f_d$  et de l'amplitude des vibrations ambiantes  $A$ . Avec des conditions initiales données, le système converge naturellement vers l'une ou l'autre de ces orbites. Certaines de ces orbites, appelées orbites hautes, correspondent à un mouvement oscillatoire entre deux positions stables de la masse inertie (voir figure 4). D'autres, appelées orbites basses, correspondent à une situation où la masse inertie oscille faiblement autour d'une seule position stable. Les orbites hautes sont beaucoup plus intéressantes pour la récupération d'énergie, parce qu'elles représentent une puissance récoltée plus importante. Les équations différentielles du système permettent de déterminer les différentes orbites cohabitant sur une fréquence donnée, mais aussi la stabilité desdites orbites grâce à une simulation. Ainsi, pour les fréquences où différentes orbites basses et hautes cohabitent, il est intéressant de mettre en place une stratégie de saut d'orbite afin de forcer l'oscillateur à se positionner sur une orbite haute et récupérer plus d'énergie [6].

### 3.2 Les sauts d'orbite

Le but d'un saut d'orbite est de forcer un oscillateur à passer d'une orbite basse à une orbite haute, et ce en dépensant le moins d'énergie possible. Il existe différentes manières d'effectuer cette manipulation, par exemple en donnant une impulsion venant perturber la trajectoire de la masse, ou encore en utilisant un bruit blanc [3]. Une autre méthode consiste à modifier certains paramètres de l'oscillateur, à l'instar du niveau de flambement  $x_w$ . Il est important de trouver la méthode la moins coûteuse en énergie afin que le système reste rentable. Notons que le niveau de flambement, donné par  $x_w/L$ , sera indirectement

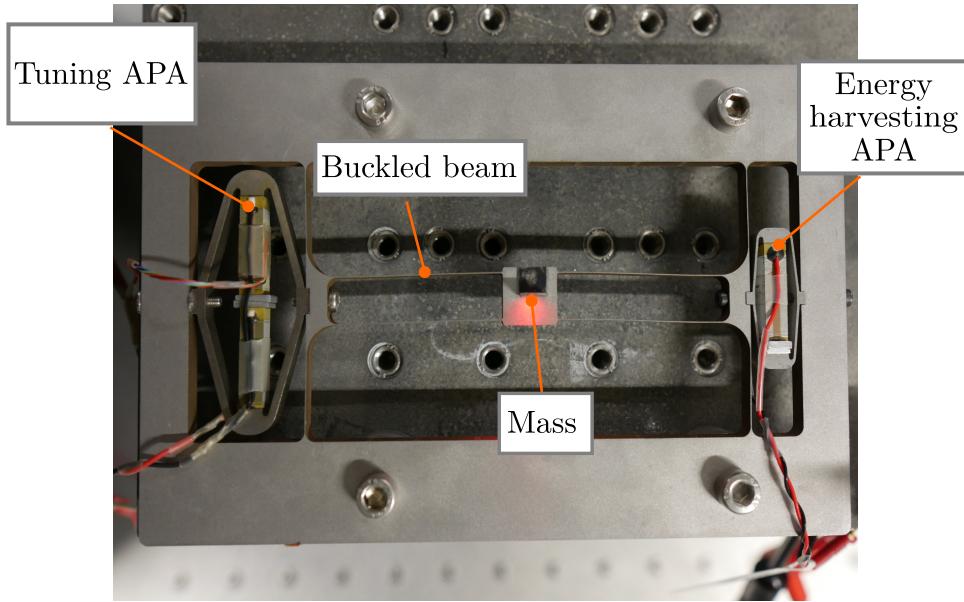


FIGURE 5 – Prototype de dernière génération développé au SYMME dans le cadre du projet H2020 Fast-Smart. Source [7].

modifié par la modification d'autres paramètres. Ainsi, en modifiant l'un de ces paramètres, il est possible de déformer l'énergie potentielle du système et ainsi le faire passer d'une orbite basse à une orbite haute. Afin d'expérimenter ces différentes stratégies, l'équipe de recherche a mis en place un prototype d'oscillateur bistable à ensemble masse-ressort flambé, permettant de modifier facilement certains de ces paramètres, et surtout d'étudier les performances d'énergie de récupération. Une photo du prototype expérimental peut être vue en figure 5. Ce système a l'avantage d'être à la fois compact et facile à mettre en place [5].

C'est ce genre d'oscillateurs qui est utilisé dans le projet de recherche dans lequel s'inscrit ce stage. C'est donc également ce genre d'oscillateurs bistables qui sera utilisé dans les modélisations de l'application à réaliser.

### 3.3 La place de ce stage

Pour mettre en place une stratégie de saut d'orbite efficace, il est donc nécessaire non seulement de connaître la méthode la moins gourmande en énergie, mais également d'intervenir au meilleur moment de la trajectoire de l'oscillateur. La modélisation permet d'observer les comportements des oscillateurs mécaniques afin de les étudier et d'observer leurs réactions face à des perturbations ou des modifications de leurs paramètres. La création d'une application telle que décrite dans le sujet de stage peut permettre de faire ce genre de modélisation. Le but est également de pouvoir observer d'autres aspects, tels que les sections de Poincaré du système, et ainsi obtenir une observation relativement diverse sur un même oscillateur. Cette application pourrait également être utilisée dans un but de vulgarisation scientifique. En effet, son utilisation, qui se veut simple et intuitive, pourrait être une bonne illustration sur les récupérateurs d'énergie, mais aussi sur les oscillateurs mécaniques en général et tous les différents phénomènes observables sur les animations disponibles.

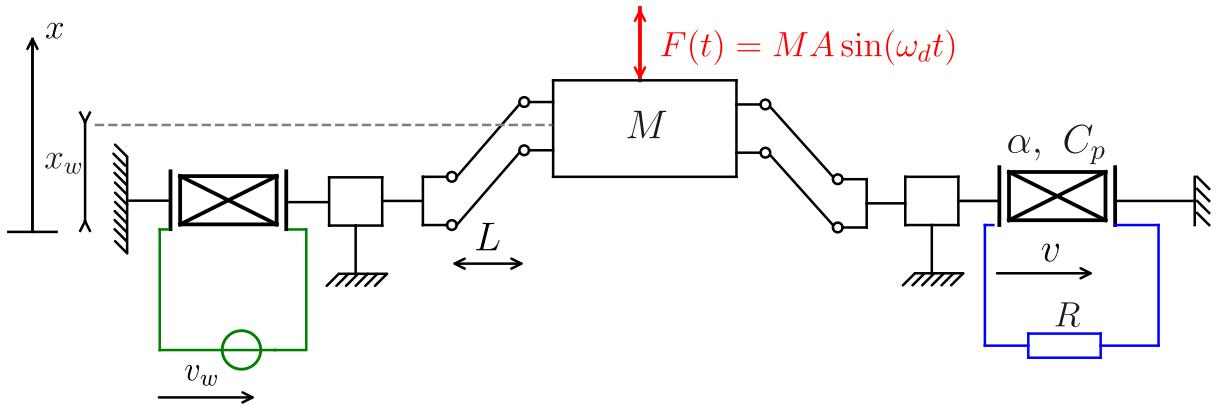


FIGURE 6 – Modélisation électromécanique de l'oscillateur développé au SYMME et présenté dans la figure 5. Source [7].

## 4 Contenu du stage

### 4.1 Travail effectué

Durant ce stage, différents outils de travail ont été utilisés. La totalité du code écrit se situe dans un repository Git, [NVEH-modelling](#), dont l'utilisation a permis de travailler en équipe, récupérer d'anciennes versions du code, et rendre l'application finale disponible. La programmation a été réalisée sur l'éditeur de code Visual Studio Code, dont les extensions permettent d'utiliser de nombreux langages différents, ainsi que des outils tels que Git, de manière rapide et instinctive. Après plusieurs discussions, il a été décidé que la programmation serait faite en utilisant le langage Julia, plus rapide et performant que Python, et avec des librairies simplifiées sur la résolution d'équations différentielles et l'affichage de graphes interactifs.

Travailler sur ce sujet a impliqué d'essayer différentes librairies Julia afin de mieux se rendre compte de ce qu'il était possible de faire avec ce langage. Il a vite été important de savoir exactement ce que les outils intégrés à Julia permettent de réaliser comme interface graphique. Pour cela, la modélisation des oscillateurs a d'abord été faite « à la main », en suivant une résolution d'équation différentielle numérique à l'aide de la méthode de Runge-Kutta d'ordre 4. Il s'est ensuite avéré que plusieurs paquetages Julia étaient capables de faire cette résolution de manière beaucoup plus rapide et intégrée dans différentes fonctions d'animation. Il existe, en Julia, une librairie de systèmes dynamiques déjà enregistrés [DynamicalSystems.jl](#) [2]. Il est possible de créer son propre système dynamique avec les équations qui le caractérisent afin qu'il se comporte de la même manière que ceux qui ont été pré-enregistrés, ce qui est très utile car certaines fonctions d'animations intéressantes ne fonctionnent qu'avec un système dynamique en entrée. Un oscillateur bistable a donc été codé de cette manière, déclaré comme système dynamique, afin de se servir de ces propriétés. Cependant, la modélisation initiale a servi pour faire différents essais d'interface graphique.

#### 4.1.1 Interface graphique

Il est vite devenu apparent que le paquetage *Gtk.jl* permettait de créer des interfaces graphiques interactives assez facilement. La création de curseurs pour modifier les paramètres et de menus déroulants pour changer de condition initiale a été mise en place afin de réaliser une fenêtre permettant de modifier un graphique. Les graphiques ont été codés à l'aide du paquetage *Plots.jl*. L'utilisation de macros permet de créer des gif ou des animations en seulement une ligne de code. Ces différentes fonctionnalités ont été utilisées pour réaliser une première fenêtre interactive. Cette première fenêtre simule de deux manières la trajectoire d'un oscillateur répondant à l'équation :

$$\ddot{x} = -\delta \dot{x} - \alpha x - \beta x^3 + t \quad (4)$$

Avec  $\delta$  le coefficient d'amortissement,  $\alpha$  le coefficient de raideur linéaire,  $\beta$  le coefficient de raideur non linéaire, et une excitation sinusoïdale d'amplitude  $A$  et de pulsation  $\omega_d$ . Les valeurs par défaut des paramètres  $\alpha$ ,  $\beta$  et  $\delta$  valent 1.

La première fenêtre interactive créée ainsi peut être vue en figure 7. Le graphe de gauche est un graphe statique montrant le résultat de la simulation vue plus haut. Celui de droite montre le même résultat tracé en bleu, et une simulation similaire dont les conditions initiales sont différentes et peuvent être redéfinies par le menu déroulant situé dans le coin en haut à droite de la fenêtre. C'est également un format gif, donc une animation qui se répète dans le temps. À gauche de ce menu déroulant se trouvent trois curseurs correspondant respectivement aux paramètres  $\alpha$ ,  $\beta$  et  $\delta$ .

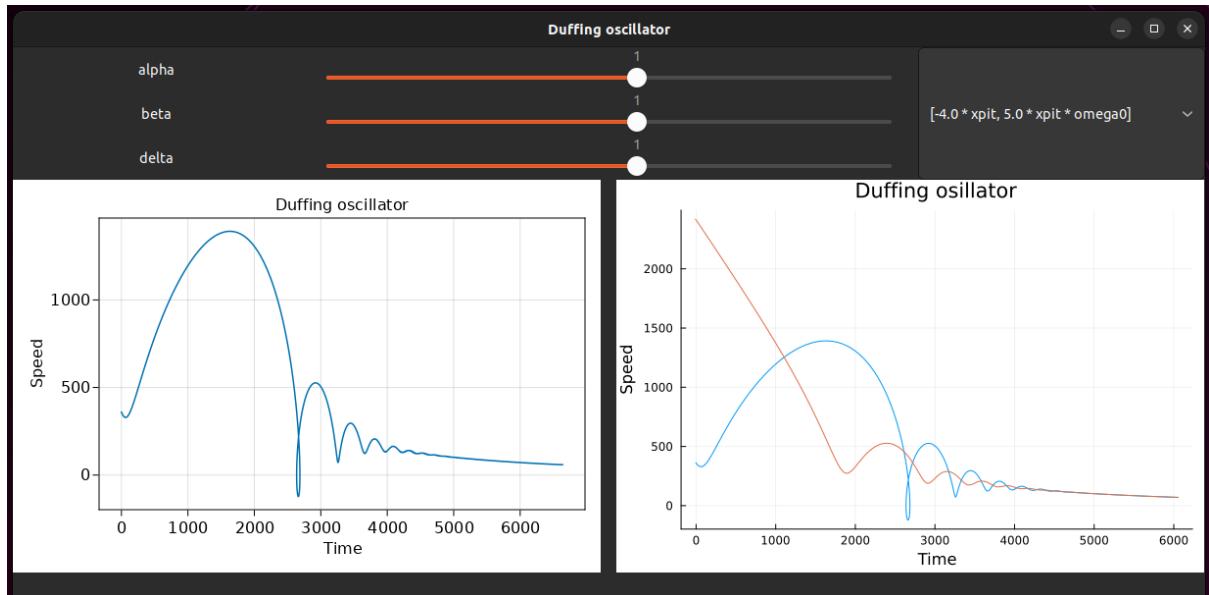


FIGURE 7 – Fenêtre interactive créée avec le paquetage *Gtk.jl* dans sa position par défaut, avec les graphes statique et dynamique de la vitesse en fonction du temps, les curseurs avec leurs labels associés, et un menu déroulant servant à modifier les conditions initiales du tracé orange.

La figure 8 permet d'illustrer le mouvement du graphe de droite : la capture a été prise à un moment où le graphe n'est pas encore complet. Ce graphe se crée en boucle : une fois arrivé à la fin du temps pris pour la trajectoire, il s'efface et recommence.

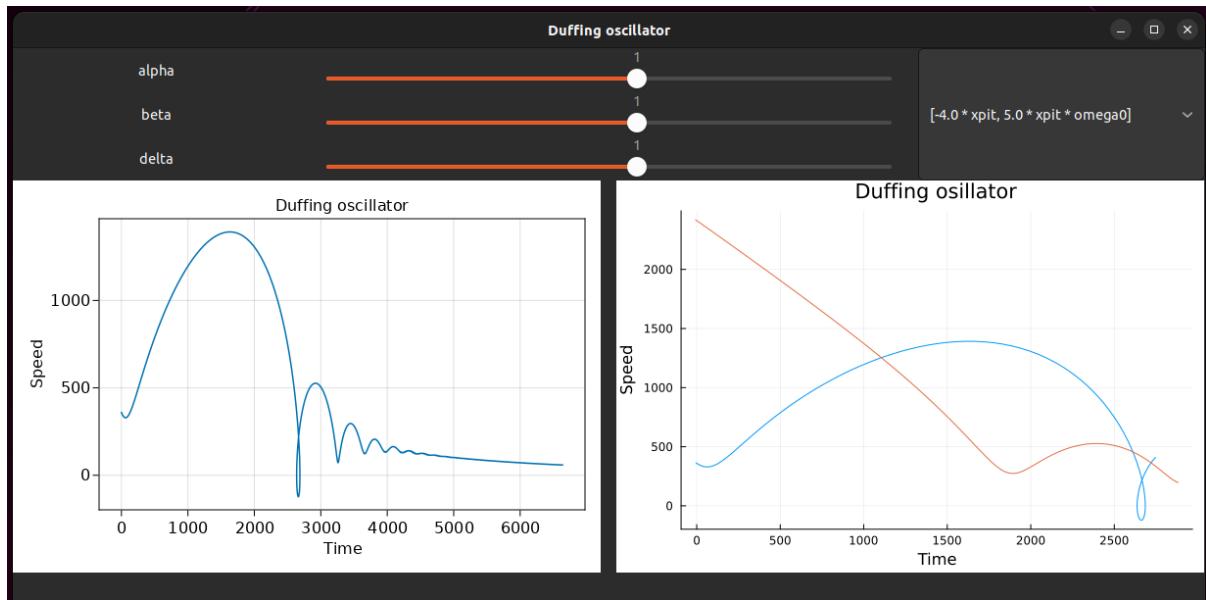


FIGURE 8 – Illustration du mouvement du graphe de droite sur la fenêtre interactive vue plus haut.

Les curseurs permettent de modifier les paramètres de l'équation différentielle et voir leur impact immédiatement sur les graphes. La figure 9 donne une vue de la fenêtre interactive avec l'un des paramètres modifié.

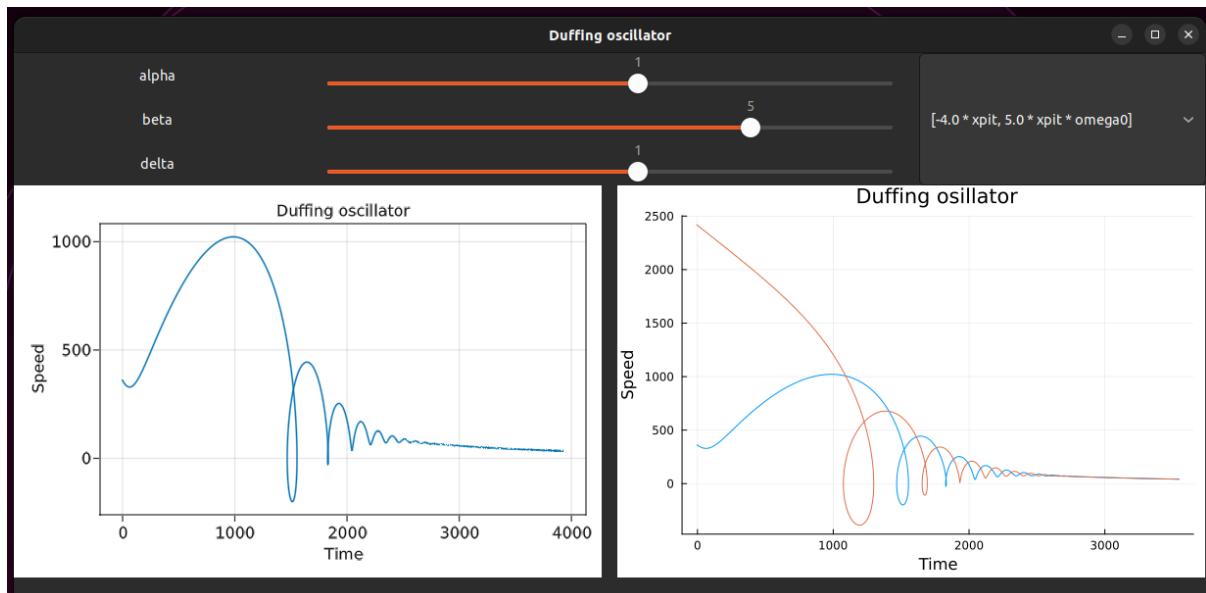


FIGURE 9 – Illustration de l'effet du mouvement des curseurs sur les graphes de la fenêtre créée pour le stage. La modification des paramètres par les curseurs se répercute sur la simulation, et les graphes sont recréés en temps réel.

Le menu déroulant en haut à droite de la fenêtre permet de modifier les conditions initiales du tracé orange sur le graphe de droite. La figure 10 montre ce qu'une modification de ces conditions initiales peut donner.

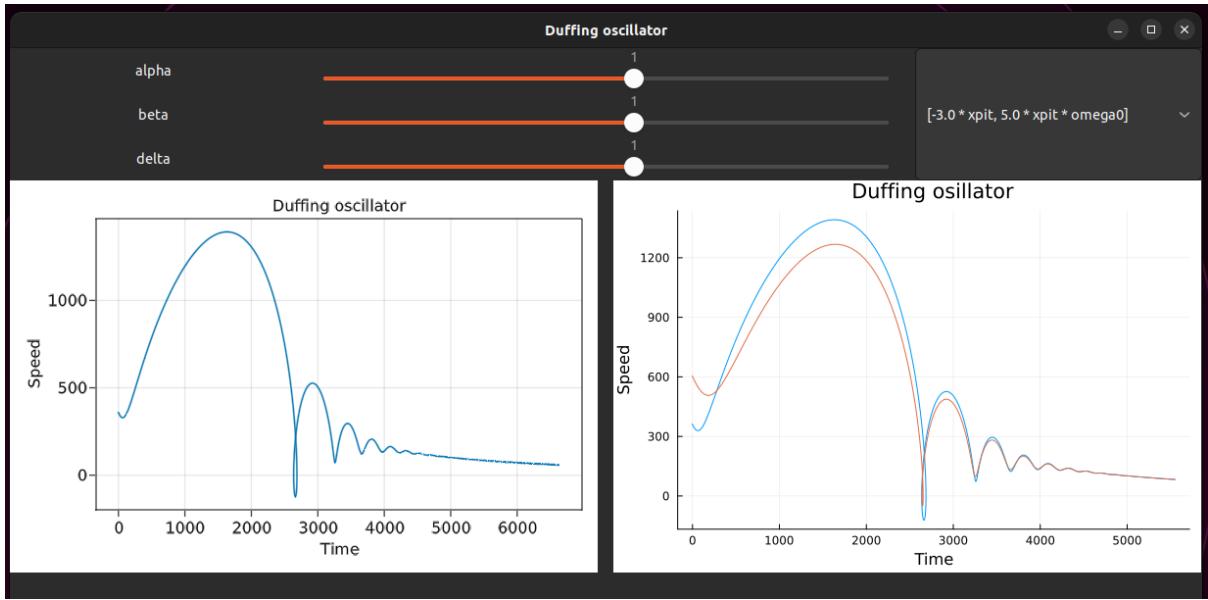


FIGURE 10 – Illustration de l’effet du menu déroulant sur la fenêtre interactive. La modification des conditions initiales entraîne un nouveau calcul et tracé du graphe orange.

Une autre façon de créer des interfaces graphiques en Julia est d’utiliser l’extension Visual Studio Code *Génie Builder*. Cette extension permet d’organiser une fenêtre graphique avec un système de « cliqué-glissé » qui se construit automatiquement en générant un code html. Il est ainsi possible de mettre en place tous les éléments souhaités (par exemple des graphes, boutons, curseurs et menus déroulants mais aussi des textes interactifs) de manière intuitive et rapide. À condition de maîtriser l’outil, cela peut permettre de gagner beaucoup de temps pour le côté purement graphique de l’application. Un fichier *app.jl* contient le code de l’application en elle-même, c’est-à-dire le code des modélisations, calculs et autres programmations nécessaires. Il faut ensuite créer des variables spécifiques contenant, par exemple, le résultat d’un calcul. Ces variables-là communiquent avec l’interface. Dans l’onglet qui sert à créer l’interface par système de cliqué-glissé, il suffit de sélectionner un élément mis en place afin d’en modifier certaines caractéristiques, notamment quelles sont les valeurs ou actions qui lui sont associées. La figure 11 montre la fenêtre de travail de l’extension *Génie Builder*. C’est une fenêtre générée automatiquement lors de la création d’une nouvelle application. Sur la droite, sous la catégorie « Bindings », se trouvent toutes les variables permettant de communiquer entre l’interface et le code du fichier *app.jl*. Sur l’interface de travail, le graphique (dernier élément, en bas) est sélectionné. Dans la colonne de droite, il est possible d’en modifier les propriétés, par exemple les valeurs prises dans le champ « Data ».

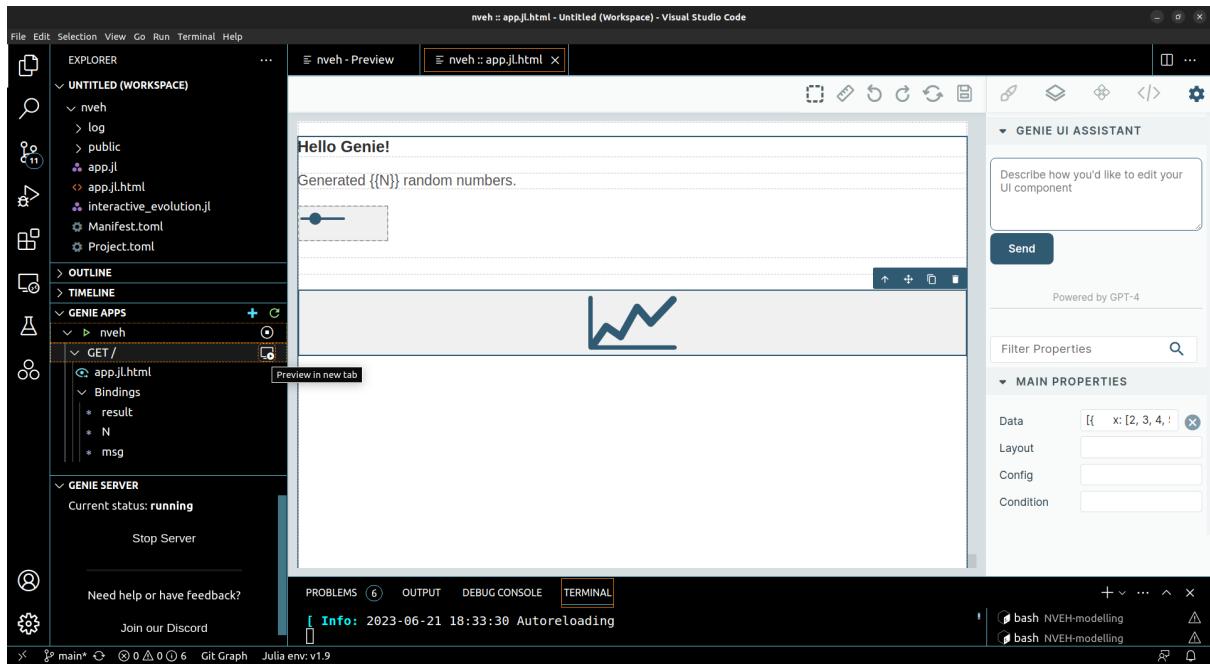


FIGURE 11 – Fenêtre de travail de Génie Builder permettant de créer une interface graphique par système de « cliqué-glissé ». La colonne de droite montre et permet de modifier les paramètres des éléments de la fenêtre, vus dans le carré central.

La figure 12 montre l’aperçu de la fenêtre générée par le code de la figure 11. Le curseur, interactif, et les deux messages qui se modifient lorsque le curseur est bougé, y sont présents, ainsi que le graphe avec les valeurs vues plus haut.

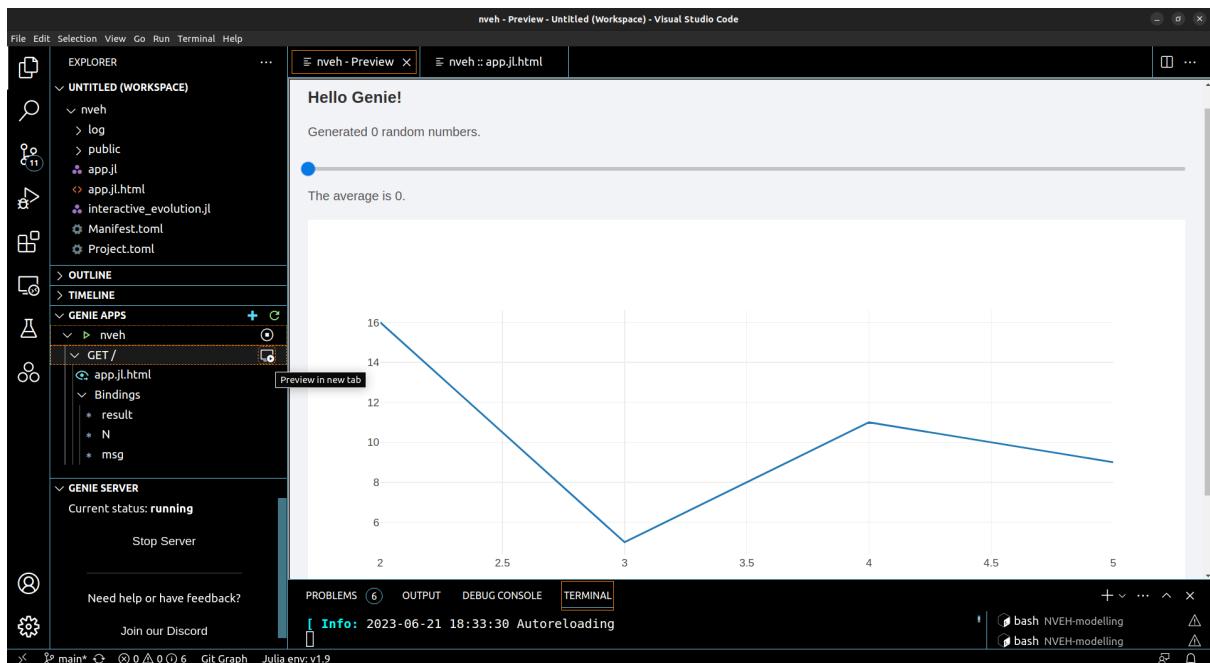


FIGURE 12 – Aperçu du résultat de Génie Builder correspondant à l’interface mise en place dans la figure précédente. Les différents éléments (messages interactifs, curseur, graphe) vus plus haut s’y trouvent.

Ce système a été retenu comme une possibilité pour créer l'application finale car, si maîtrisé correctement, il pourrait permettre de prendre beaucoup moins de temps sur le côté esthétique pour agencer correctement l'interface, et permettre de se concentrer un peu plus le côté modélisation et programmation.

#### 4.1.2 Systèmes dynamiques

En Julia, il existe de nombreux paquetages permettant de manipuler des systèmes dynamiques de manière simplifiée. Le premier qui sera abordé ici est le paquetage [Harmonic-Balance.jl](#) [4]. Ce paquetage utilise la méthode de la balance harmonique pour calculer des solutions périodiques en passant par le domaine fréquentiel. La méthode de la balance harmonique consiste à trouver une solution périodique, notée  $x$  que l'on suppose de la forme suivante :

$$x(t) = u_1 \cos(\omega t) + v_1 \sin(\omega t) \quad (5)$$

Ce paquetage permet par exemple de trouver des solutions pour un oscillateur dont la dynamique est décrite par l'équation :

$$\ddot{x}(t) + \omega_0^2 x(t) + \alpha x(t)^3 + \eta x(t)^2 \dot{x}(t) = F \cos(\omega t) \quad (6)$$

Ainsi, par exemple, le graphe vu en figure 13 présente l'amplitude des solutions en fonction de la fréquence angulaire.

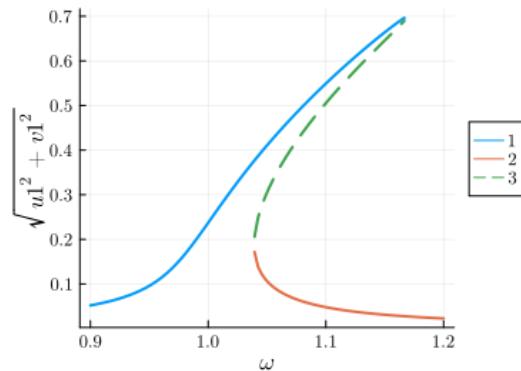


FIGURE 13 – Amplitude en fonction de la fréquence angulaire des solutions de calculées par la méthode de la balance harmonique. Source : [4]

Pour l'obtenir, il faut déclarer l'équation différentielle étudiée ainsi que le paramètre variable souhaité en abscisse. Cette possibilité de graphe ne fait pas encore partie du projet actuel de l'application, mais elle est très intéressante et y sera probablement ajoutée plus tard.

Comme vu plus tôt, Julia présente également une librairie de systèmes dynamiques déjà programmés grâce au paquetage [DynamicalSystems.jl](#) [2]. Il est possible de créer un objet de ce type afin d'utiliser les fonctions qui utilisent des systèmes dynamiques pour de la modélisation, par exemple. Ainsi, même si un système dynamique modélisant un oscillateur

de Duffing existe déjà en Julia, un nouveau système a été créé afin de coller au modèle du récupérateur bistable utilisé par l'équipe de recherche, qui correspond à l'équation suivante :

$$\begin{cases} \ddot{x} = -\frac{\omega_0^2}{2} \left( \frac{x^2}{x_w^2} - 1 \right) x - \frac{\omega_0}{Q} \dot{x} - 2 \frac{\alpha}{ML} x v + A_d \sin(2\pi f_d t) \\ \dot{v} = 2 \frac{\alpha}{LC_p} x \dot{x} - \frac{1}{RC_p} v \end{cases} \quad (7)$$

On pourra par exemple modifier les valeurs des paramètres du système tels que la pulsation propre  $\omega_0$ , les paramètres de l'excitation ( $A, f_d$ ), les positions d'équilibre  $\pm x_w$ , etc... et voir leur influence sur la dynamique.

L'utilisation de la fonction `interactive_evolution()`, du paquetage `InteractiveDynamics.jl`, permet de prendre en arguments un système dynamique, et de le visualiser en temps réel. Dans le cas présent, la trajectoire de plusieurs récupérateurs bistables, dont les conditions initiales sont définies en avance, sont visualisées. La figure 14 représente l'animation dans laquelle on affiche la vitesse  $\dot{x}$  en fonction du déplacement  $x$ . Ce plan 2D est généralement appelé l'espace des phases.

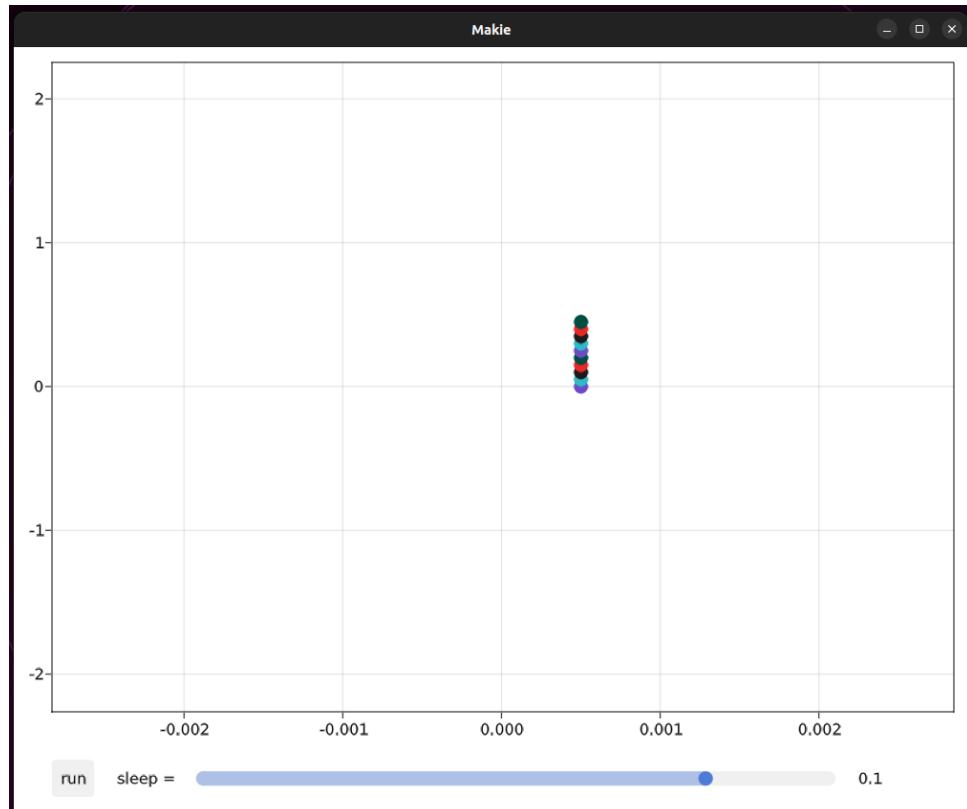


FIGURE 14 – Fenêtre initiale créée par la fonction `interactive_evolution()` du paquetage `InteractiveDynamics.jl`, montrant les conditions initiales des différents récupérateurs bistables modélisés dans le plan.

La figure 15 montre la même fenêtre un peu plus tard dans le temps. Le curseur situé en-dessous du graphique permet de gérer le temps de pause entre deux itérations. Contrairement aux animations au format gif vues plus haut, cette animation évolue en temps réel

et ne se reforme pas après un certain temps, ce qui correspond beaucoup mieux aux attentes du sujet de stage.

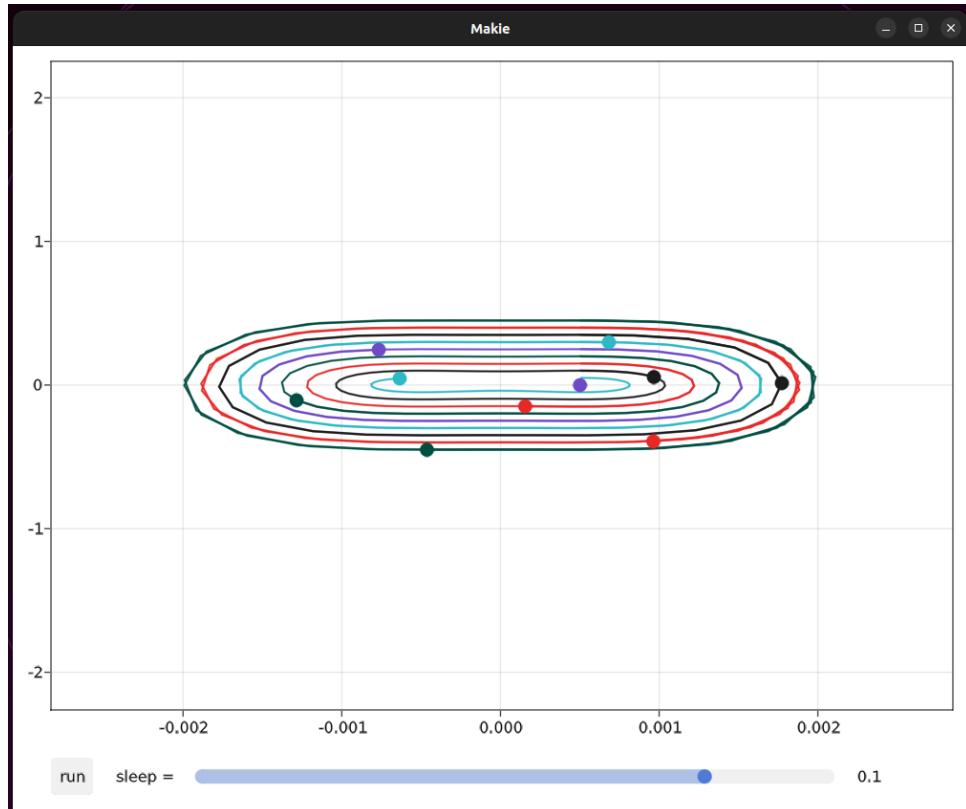


FIGURE 15 – Fenêtre créée par la fonction `interactive_evolution()` du paquetage `InteractiveDynamics.jl`, montrant les états des oscillateurs vus dans la figure 14 un peu plus tard dans le temps.

Selon les versions<sup>1</sup> du paquetage de `InteractiveDynamics.jl`, il est possible d'entrer d'autres arguments en paramètres de la fonction `interactive_evolution()` permettant de créer des curseurs supplémentaires pour changer par exemple les valeurs des paramètres du système. Ces curseurs sont alors automatiquement liés à l'expression du système dynamique, et permettent de modifier des paramètres en temps réel. Ce genre de fonctionnalité permet de mettre en place une visualisation comme décrite dans le sujet de stage, et ce avec un code très simple et intuitif. En revanche, la version de paquetage nécessaire n'est pas compatible avec la plupart des autres paquetages ou fonctionnalités utilisées.

Une autre fonctionnalité du paquetage `InteractiveDynamics.jl` permet de faire un scan en trois dimensions des sections de Poincaré d'un système. En dynamique des systèmes, les sections de Poincaré forment un outil de visualisation des trajectoires dans l'espace d'état. Elles permettent de simplifier l'étude d'un système de dimension  $N$  en traçant les points traversant un hyperplan de dimension  $N - 1$ . La figure 16 illustre la construction des sections de Poincaré dans le cas d'un système dynamique de dimension 3. À nouveau, son utilisation est très simple, puisqu'il suffit de définir le système dynamique étudié et lancer la fonction `brainscan_poincaresos()`. Le résultat est une fenêtre interactive, avec sur la gauche une vi-

---

1. La version v0.22.1 par exemple permet l'ajout des curseurs.

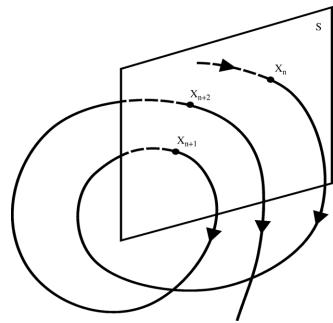


FIGURE 16 – Illustration du principe de construction des sections de Poincaré. [Source](#)

sion globale montrant la surface étudiée, sur la droite la surface de Poincaré scannée, et en bas un curseur permettant de modifier la position de la surface en question. Les figures 17 et 18 montrent ces résultats, avec différentes positions de la surface scannée. Le graphique de gauche représente le plan 3D ( $x, \dot{x}, v$ ).

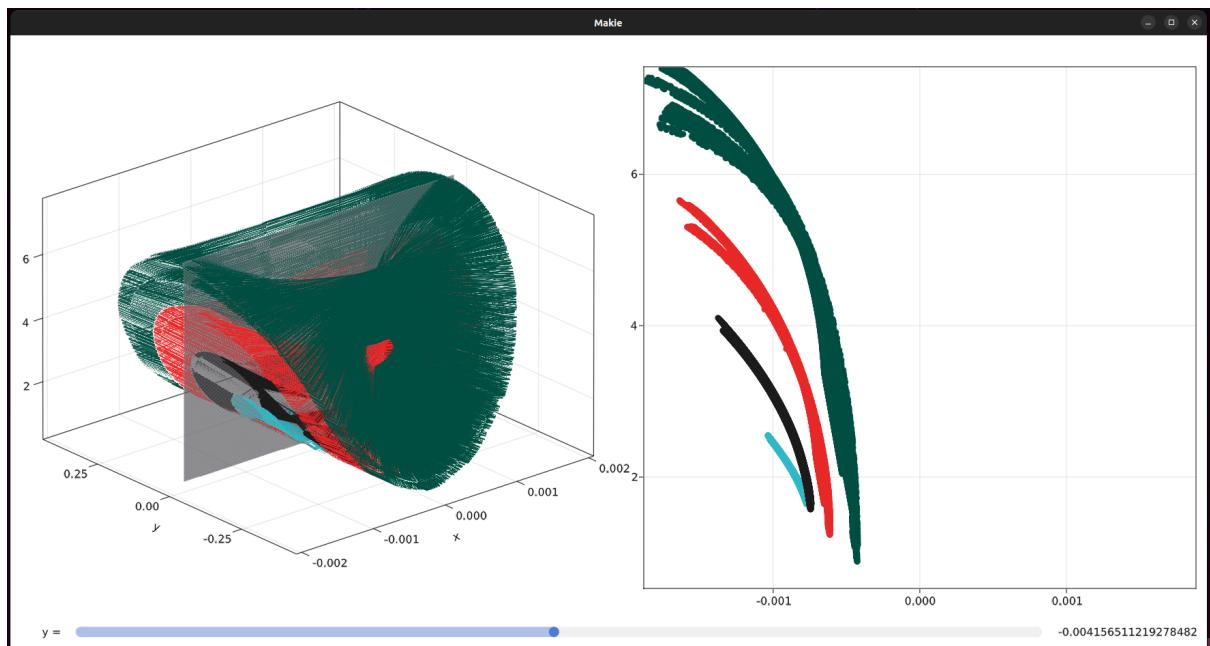


FIGURE 17 – Scan des surfaces de Poincaré créé par une animation de Julia dans une première position

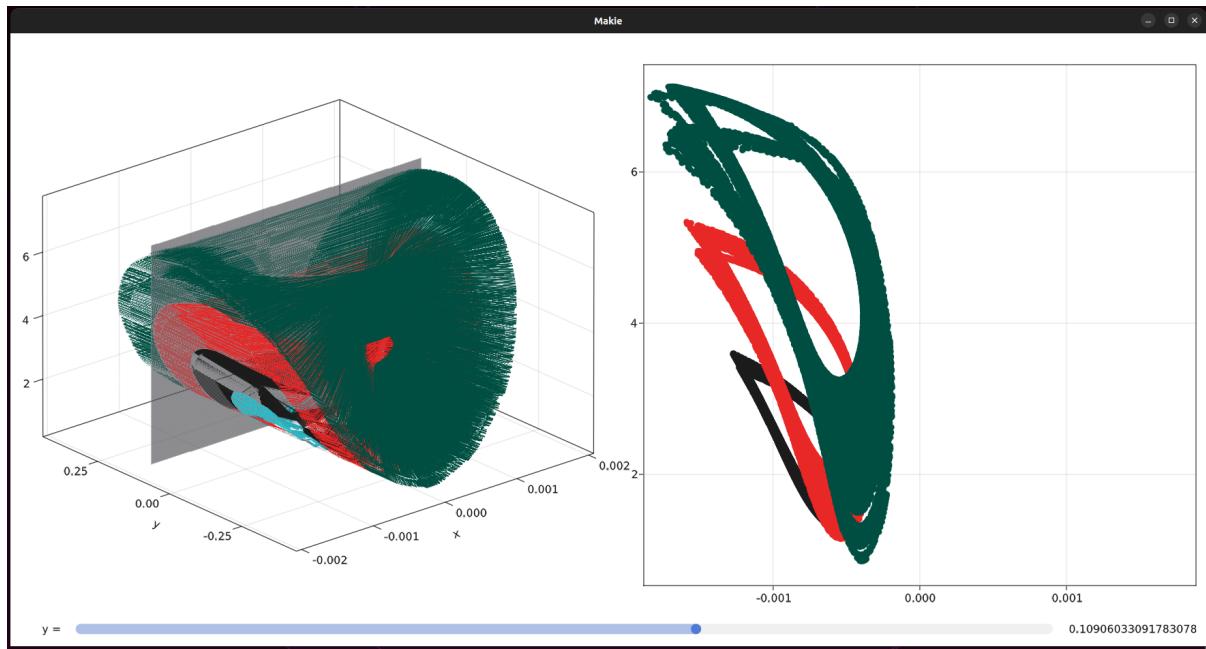


FIGURE 18 – *Scan des surfaces de Poincaré créé par une animation de Julia dans une deuxième position*

Pour rester sur le thème des sections de Poincaré, une autre animation intégrée dans le paquetage *DynamicalSystems.jl* permet de créer une carte interactive pour explorer une section de surface de Poincaré. La création de la fenêtre utilise les conditions initiales définies dans le code, et affiche toutes les solutions du système. Il suffit de cliquer sur un endroit du graphe pour créer de nouvelles conditions initiales et observer les solutions. Deux curseurs permettent de définir le temps de simulation et la taille des points. La figure 19 montre la fenêtre créée ainsi avec les résultats de trois conditions initiales différentes.

L’animation telle que vue sur la figure 19 utilise non pas un récupérateur bistable comme vu précédemment, mais un système de Hénon Heiles, qui était l’exemple dans la documentation de la fonction. Un système de Hénon-Heiles représente le mouvement non linéaire d’une étoile autour d’un centre galactique. Le fonctionnement avec un système bistable défini par l’équipe de travail n’a pas encore été mis en place.

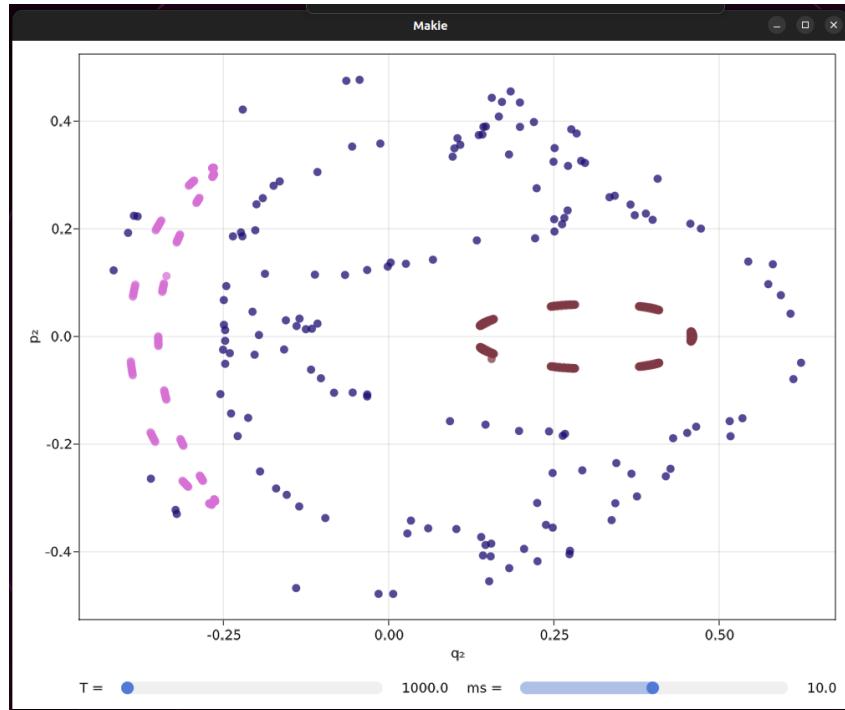


FIGURE 19 – Carte des surfaces de Poincaré associée au système de Hénon-Heiles créée avec le paquetage `DynamicalSystems.jl`, montrant les résultats correspondant à trois conditions initiales différentes, et différencierées par couleurs.

Ces différentes animations sont très intéressantes pour le développement de l’application désirée, car elles permettent de modéliser et afficher des résultats rapidement, avec un effort minimal. Elles ne sont, en revanche, pas forcément toutes compatibles avec les systèmes de création d’interfaces vus plus haut, puisqu’elles ne peuvent pas être intégrées directement dedans, et que les différentes versions nécessaires ne sont pas forcément compatibles entre elles.

## 4.2 Perspectives

Afin de pouvoir utiliser des animations qui requièrent des versions de paquetages incompatibles les unes avec les autres, il a été décidé de faire une interface principale sur laquelle l’utilisateur pourrait décider quelle animation observer.

Le choix d’une animation spécifique se ferait à l’aide d’un clic sur un bouton. C’est ce clic qui pourrait commander la lecture du fichier contenant le code de l’animation en question. En haut de ce fichier, la commande `@quickactivate()` pourrait permettre d’activer un environnement spécifique de Julia, contenant les versions de paquetage nécessaires au fonctionnement de l’animation souhaitée. Il faut donc mettre en place les différents environnements nécessaires, et terminer de créer les animations afin de répondre au cahier des charges. Par exemple, la carte de Poincaré interactive n’est pas encore codée pour utiliser un oscillateur bistable correspondant à celui utilisé par l’équipe de recherche.

Pour ce qui est de la fenêtre principale, elle sera probablement réalisée à l’aide du paquetage `Gtk.jl`. Un essai a été fait avec l’extension `Génie Builder`, mais l’intégration du fichier

contenant l'animation se faisait mal : l'enregistrement du fichier lançait l'animation, et le clic sur le bouton renvoyait une erreur. Probablement qu'il est possible de le faire tout de même, mais le temps dépensé à trouver la solution risque de ne pas être rentable par rapport à celui gagné pour l'agencement des différents composants de la fenêtre. En effet, si la fenêtre principale ne contient que quelques explications et des boutons, mais pas ou peu d'animations, l'expérience déjà acquise permettra de la réaliser rapidement, et l'aide d'une interface par cliqué-glissé devient négligeable. De plus, l'extension *Génie Builder* a été peu utilisée jusqu'à maintenant, et n'est pas encore maîtrisée.

À présent que tous les éléments permettant de créer l'application sont connus, il ne reste donc plus qu'à finir de mettre en place correctement les animations et faire le lien entre elles et la fenêtre graphique principale. Si un cahier des charges a été établi, il a été fait avant de savoir exactement ce que Julia permettrait de réaliser. Il faudra donc vérifier au fur et à mesure de l'avancement ce que l'ensemble de l'équipe souhaite mettre en place précisément dans l'application.

La figure 20 présente l'organisation actuelle des différents fichiers de l'application.

- *main.jl* : fichier principal qui sert à lancer l'application ;
- *window.jl* : fichier qui crée la fenêtre interactive et qui définit la position des différents éléments qui y sont créés (par exemple les curseurs) ;
- *plottings.jl* : fichier qui construit les différents graphes ;
- *modelisation.jl* : fichier qui contient les codes définissant les systèmes dynamiques étudiés ;
- *animations* : dossier contenant les codes des différentes animations présentées dans ce rapport.

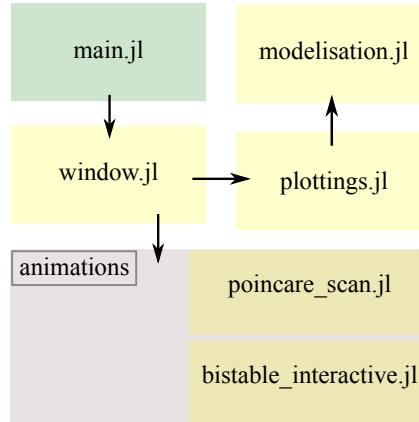


FIGURE 20 – Schéma de l'organisation des fichiers de l'application.

## 5 Conclusion

La réalisation de ce stage a été une occasion d'apprendre le langage Julia et d'acquérir de nouvelles compétences en gestion de projet et en informatique. Si l'application n'est pas encore terminée au moment de la rédaction de ce rapport, le temps conséquent passé à explorer les différents outils mis à disposition par Julia et ses nombreux paquetages a permis d'avoir une idée assez claire du chemin qu'il reste à parcourir.

À présent que la création d'interfaces graphiques, de graphes interactifs, et les animations pré-programmées dans les librairies Julia sont maîtrisées, il ne reste plus qu'à assembler le tout dans une application cohérente, en utilisant le système dynamique créé par l'équipe de recherche correspondant à l'oscillateur bistable étudié. La principale difficulté qu'il reste à surmonter est sans doute la gestion des environnements, qui a déjà posé problème plusieurs fois durant ce stage. Une fois cette problématique réglée, la fin du code de l'application devrait se dérouler facilement dans les deux semaines de stage qu'il reste.

Il est également possible que l'équipe encadrante de ce stage demande la rédaction d'un manuel d'utilisation au terme de la programmation de l'application, ce qui serait une nouvelle expérience. Le fait de déployer cette application de manière publique sur internet a été discuté, mais aucune décision n'a été prise pour le moment, et les moyens de le faire n'ont pas été explorés si l'utilisation de *Génie Builder* est annulée.

## 6 Bibliographie

### Références

- [1] Ludovic Charleux. *Modélisation et simulation de systèmes mécaniques pour la mécatronique*. Habilitation à diriger des recherches mécanique, Université Savoie Mont Blanc, Juin 2023.
- [2] George Datseris. Dynamicalsystems.jl : A julia software library for chaos and nonlinear dynamics. *Journal of Open Source Software*, 3(23) :598, mar 2018.
- [3] Thomas Huguet. *Vers une meilleure exploitation des dispositifs de récupération d'énergie vibratoire bistables : Analyse et utilisation de comportements originaux pour améliorer la bande passante*. PhD thesis, Université de Lyon, 2018.
- [4] Jan Košata, Javier del Pino, Toni Louis Heugel, and Oded Zilberberg. Harmonicbalance.jl : A julia suite for nonlinear dynamics using harmonic balance. *SciPost Physics Codebases*, page 006, 2022.
- [5] Weiqun Liu. *Conception d'un dispositif de récupération d'énergie vibratoire large bande*. PhD thesis, 2014. Thèse de doctorat dirigée par Formosa, FabienAgbossou, Amen et Badel, Adrien Sciences pour l'ingénieur Grenoble 2014.
- [6] Camille Saint-Martin, Adrien Morel, Ludovic Charleux, E Roux, Aya Benhemou, and Adrien Badel. Power expectation as a unified metric for the evaluation of vibration energy harvesters. *Mechanical Systems and Signal Processing*, 181 :109482, 2022.
- [7] Camille Saint-Martin, Adrien Morel, Ludovic Charleux, Emile Roux, David Gibus, Aya Benhemou, and Adrien Badel. Optimized and robust orbit jump for nonlinear vibration energy harvesting, 2023.

## Table des figures

1	Organisation du laboratoire SYMME. . . . .	6
2	Décomposition d'un système de récupération d'énergie vibratoire en plusieurs sous-systèmes. Ce stage se concentre sur la partie « Generator », c'est-à-dire l'oscillateur mécanique et le transducteur électromécanique. La charge électrique est considérée comme une simple charge résistive. Source : [5]. . . . .	8
3	Présentation des avantages des oscillateurs non-linéaires dans le cadre de la récupération d'énergie vibratoire. En haut à gauche, deux paysages vibratoires sont représentés par leur densité spectrale d'énergie. Le signal orange présente une grande accélération à basse fréquence et le bleu une accélération répartie sur une plus grande plage de fréquences avec un pic marqué. Dans le cadre « Linear solution », les performances d'un oscillateur linéaire sont présentées. Celui-ci est optimisé pour le signal bleu et sa fréquence propre est donc ajustée à l'endroit du pic. La réponse en puissance est représentée sur la droite du cadre et la réponse bleue est très bonne. Lorsque ce même oscillateur est soumis au signal orange, la réponse est très mauvaise. En dessous, dans le cadre « Wideband solution », la même démarche est effectuée avec un oscillateur non-linéaire. Il est optimisé pour le signal bleu pour lequel il fournit une bonne puissance. Par contre, il s'adaptera aussi relativement bien au signal orange. Cela illustre le caractère large bande du système non-linéaire. Source : [5]. . . . .	9
4	Représentation schématique de l'énergie potentielle des systèmes de récupération d'énergie vibratoire développés au SYMME en fonction du déplacement de la masse $x$ . Le potentiel présente deux minimums qui sont aussi des positions d'équilibre en $x = \pm x_w$ . Un point col est présent en $x = 0$ . Les zones rouges de basse énergie représentent le lieu des orbites dites basses. La zone verte de haute énergie est celui des orbites hautes. Source : [1]. . . . .	9
5	Prototype de dernière génération développé au SYMME dans le cadre du projet H2020 Fast-Smart. Source [7]. . . . .	11
6	Modélisation électromécanique de l'oscillateur développé au SYMME et présenté dans la figure 5. Source [7]. . . . .	12
7	Fenêtre interactive créée avec le paquetage <i>Gtk.jl</i> dans sa position par défaut, avec les graphes statique et dynamique de la vitesse en fonction du temps, les curseurs avec leurs labels associés, et un menu déroulant servant à modifier les conditions initiales du tracé orange. . . . .	13
8	Illustration du mouvement du graphe de droite sur la fenêtre interactive vue plus haut. . . . .	14
9	Illustration de l'effet du mouvement des curseurs sur les graphes de la fenêtre créée pour le stage. La modification des paramètres par les curseurs se répercutent sur la simulation, et les graphes sont recréés en temps réel. . . . .	14
10	Illustration de l'effet du menu déroulant sur la fenêtre interactive. La modification des conditions initiales entraîne un nouveau calcul et tracé du graphe orange. . . . .	15

11	Fenêtre de travail de <i>Génie Builder</i> permettant de créer une interface graphique par système de « cliqué-glissé ». La colonne de droite montre et permet de modifier les paramètres des éléments de la fenêtre, vus dans le carré central. . . . .	16
12	Aperçu du résultat de <i>Génie Builder</i> correspondant à l'interface mise en place dans la figure précédente. Les différents éléments (messages interactifs, curseur, graphe) vus plus haut s'y trouvent. . . . .	16
13	Amplitude en fonction de la fréquence angulaire des solutions de calculées par la méthode de la balance harmonique. Source : [4] . . . . .	17
14	Fenêtre initiale créée par la fonction <i>interactive_evolution()</i> du paquetage <i>InteractiveDynamics.jl</i> , montrant les conditions initiales des différents récepteurs bistables modélisés dans le plan. . . . .	18
15	Fenêtre créée par la fonction <i>interactive_evolution()</i> du paquetage <i>InteractiveDynamics.jl</i> , montrant les états des oscillateurs vus dans la figure 14 un peu plus tard dans le temps. . . . .	19
16	Illustration du principe de construction des sections de Poincaré. Source . . .	20
17	Scan des surfaces de Poincaré créé par une animation de Julia dans une première position . . . . .	20
18	Scan des surfaces de Poincaré créé par une animation de Julia dans une deuxième position . . . . .	21
19	Carte des surfaces de Poincaré associée au système de Hénon-Heiles créée avec le paquetage <i>DynamicalSystems.jl</i> , montrant les résultats correspondant à trois conditions initiales différentes, et différenciées par couleurs. . . . .	22
20	Schéma de l'organisation des fichiers de l'application. . . . .	23

## 7 Annexes

### A. Cahier des charges pour l'interface de visualisation des oscillateurs

**Objectif :** Développer une interface graphique pour comprendre l'influence des paramètres d'un oscillateur mécanique

#### Fonctionnalités générales

- L'application doit être conçue pour fonctionner sur des systèmes d'exploitation variés (Windows, MacOS, Linux), avec une installation minimale nécessaire.

#### Fonctionnalités spécifiques

##### Priorité

- L'utilisateur doit pouvoir ajuster les paramètres suivants pour chaque oscillateur :
  - amortissement
  - raideur
  - amplitude, fréquence de l'excitation
  - niveau de flambement
- Ces paramètres doivent pouvoir être ajustés dynamiquement, avec une visualisation en temps réel des modifications sur les trajectoires et les potentiels élastiques.
- L'application doit afficher les trajectoires dans le plan de phase pour chaque oscillateur.
- L'application doit afficher la forme de l'énergie potentielle élastique pour chaque oscillateur. Leur forme doit changer en temps réel suivant les variations induites par l'utilisateur.
- L'utilisateur doit pouvoir zoomer et dézoomer sur le plan de phase pour une visualisation plus détaillée ou plus large.

##### Analyse de l'énergie

- L'application doit fournir une option pour ajouter et visualiser des quantités d'analyse telles que l'énergie totale récupérée au cours du temps et l'espérance de puissance (dans le cas de plusieurs oscillateurs, cette quantité va évoluer au cours du lancement de l'application).

##### Option

- L'utilisateur doit pouvoir choisir de lancer un ou plusieurs oscillateurs simultanément.
- L'utilisateur peut choisir quelle variable tracée en fonction du temps ou d'un autre paramètre

- Possibilité de sauvegarder les valeurs des paramètres, les puissances des orbites, l'espérance, etc... associées aux simulations numériques sous forme d'un fichier csv ou hdf5
- L'utilisateur pourra alors charger une configuration de paramètre sauvegardée et reprendre à partir de celle-ci (au lieu de partir des conditions par défaut définies par le concepteur)

### **Livrables**

- Code source de l'application
- Documentation technique décrivant le fonctionnement interne de l'application (rapport de stage ?)

## B. Rapports d'avancement

### Rapport d'avancement 1 - Semaines 1 et 2

Ceci est le premier rapport d'avancement de mon stage. Pour rappel, le sujet de mon stage est la création d'une interface graphique permettant de visualiser les comportements d'un oscillateur mécanique de façon interactive, c'est-à-dire de façon à ce que l'utilisateur puisse modifier certains paramètres et voir leur influence en temps réel.

Pour commencer mon stage et me mettre dans le sujet, j'ai commencé par lire plusieurs articles me permettant de mieux comprendre les enjeux et l'intérêt de mon stage. J'ai ainsi pu me familiariser avec le principe de récupération d'énergie vibratoire et d'orbites. Pour chaque condition initiale, plusieurs cycles limites, appelés orbites, peuvent apparaître. Parmi ces orbites, certaines, appelées "orbites hautes" permettent de récupérer plus d'énergie que les autres. J'ai eu l'occasion de faire un TP sur la simulation d'oscillateurs de Duffing, qui m'a introduit plusieurs notions comme les sections de Poincaré. J'ai également utilisé ce TP pour effectuer plusieurs tests sur l'utilisation de Numba, dont j'aurai probablement à me servir afin d'accélérer les parties simulation de mon code.

Une fois le sujet du stage et les différentes notions bien assimilés, j'ai pu me pencher sur l'interface en elle-même. Ma tutrice de stage, Madame Camille Saint-Martin, avait déjà réalisé une interface en utilisant Python et sa librairie PyQt. Nous avons également parlé du langage Julia. Mon travail pour les jours à venir est d'essayer les deux langages pour pouvoir les comparer et décider lequel des deux sera le plus approprié. Pour ce faire, j'ai déjà commencé à me familiariser avec le langage Julia et la manière d'afficher des images, statiques ou dynamiques, et d'intégrer du code en Python (servant à la simulation) à l'intérieur.

J'ai créé un repository Git sur lequel je mettrai mon code. Pour l'instant, j'ai ajouté au fichier 'tests\_julia' mes essais sur le langage Julia. En voici le lien : <https://github.com/SalsyUniate/NVEH-modelling>

## Rapport d'avancement 2 - Semaines 3 et 4

Ceci est le deuxième rapport d'avancement de mon stage. Pour rappel, le sujet de mon stage est la création d'une interface graphique permettant de visualiser les comportements d'un oscillateur mécanique de façon interactive, c'est-à-dire de façon à ce que l'utilisateur puisse modifier certains paramètres et voir leur influence en temps réel.

Au début de mon stage, j'avais commencé par entrer doucement dans le sujet en lisant divers papiers et en faisant un TP sur la modélisation de l'oscillateur de Duffing.

Durant les deux semaines qui viennent de s'écouler, j'ai eu l'occasion d'explorer le langage informatique Julia, plus performant et plus rapide que Python, et bien documenté. J'ai commencé par faire de nombreux essais pour voir ce qu'il m'était possible de faire, et ai créé une fenêtre interactive dans laquelle j'ai pu afficher des graphes, statiques ou dynamiques, et modifier des paramètres, modifications qui se répercutent directement sur les affichages en cours. Si j'ai réussi à afficher des graphes dynamiques en créant des gif qui se jouent en boucle, nous préférerions avoir un affichage "en temps réel", ce qui est possible avec le langage Julia mais je n'ai pas encore réussi à le mettre en place.

Mes objectifs pour les semaines à venir sont donc de réussir à créer un affichage dynamique en temps réel, et avancer l'application en elle-même. J'ai commencé à créer un dossier src contenant un code plus propre pour commencer l'application, mais ce n'est encore que le tout début. J'ai également découvert une extension Visual Studio Code permettant de créer des interfaces graphiques en Julia sans avoir à les coder, avec un système graphique de "drag-and-drop". Après discussion avec ma tutrice entreprise, nous avons décidé que si l'outil est pratique, cela pourrait me permettre de gagner du temps et de me concentrer plus amplement sur le code des simulations en lui-même. J'explorerais donc ceci.

Voici le lien de mon repository Git, qui contient mes différents essais et mon fichier src : <https://github.com/SalsyUniate/NVEH-modelling>

## Rapport d'avancement 3 - Semaines 5 et 6

Ceci est le troisième rapport d'avancement de mon stage. Pour rappel, le sujet de mon stage est la création d'une interface graphique permettant de visualiser les comportements d'un oscillateur mécanique de façon interactive, c'est-à-dire de façon à ce que l'utilisateur puisse modifier certains paramètres et voir leur influence en temps réel.

Durant les quatre premières semaines de mon stage, j'avais commencé par étudier l'aspect théorique du sujet en lisant différents documents et en faisant quelques essais de code en Python. Suite à cela, j'avais commencé à explorer Julia, un langage de programmation proche de Python avec de nombreuses librairies permettant de faire des modélisations et des animations facilement. En utilisant ce langage, j'avais réussi à créer des graphiques, statiques et dynamiques, interactifs, dans une fenêtre permettant de modifier en temps réel certains paramètres.

Ces méthodes faites « à la main » étaient lentes et ne correspondaient pas exactement à ce qui était attendu pour l'application finale. Dans les deux semaines qui viennent de s'écouler, j'ai pris connaissance de manières bien plus simples de faire un travail plus complet. Certains paquetages Julia utilisent des objets de type systèmes dynamiques pour faire des animations interactives extrêmement complètes à l'aide de fonctions déjà programmées. Il est donc très simple de mettre en place des animations complètes et intéressantes pour notre travail. La principale difficulté vient du fait que certaines de ces fonctions nécessitent des versions de paquetages qui ne sont pas toujours compatibles avec les autres. Une solution envisagée est de créer une interface principale avec des boutons menant à des animations. L'appui sur ces boutons activerait l'environnement nécessaire et permettrait ainsi à l'utilisateur d'avoir accès à de nombreuses animations différentes.

L'utilisation de l'outil Génie Builder a finalement été écartée. Si l'interface graphique est aussi simple, le temps gagné par le système de cliqué-glissé ne rattrapera pas le temps perdu à maîtriser l'outil.

Mon objectif pour les deux semaines à venir est de terminer l'application. Ces deux semaines sont les dernières de mon stage. A présent que j'ai accès à tous les outils nécessaires (interface graphique, graphiques interactifs,

animations de systèmes dynamiques...) il ne me reste plus qu'à assembler le tout en respectant les indications de ma tutrice entreprise.

Pour rappel, voici le lien de mon repository Git, qui contient l'ensemble de mon code : <https://github.com/SalsyUniate/NVEH-modelling>