Travaux pratiques d'informatique musicale,
TP2 : Plugin synthétiseur

Licence 2, de physique et musicologie (version : 11/01/2025)

Frédéric Faure
Université Grenoble Alpes, France
frederic.faure@univ-grenoble-alpes.fr

Résumé

Dans ces **travaux pratiques** on utilise la librairie JUCE en C++ pour fabriquer des applications et plugins audio et midi.

Dans ce TP2, on propose de créer un plugin **synthétiseur** (Midi→Audio) basé sur un modèle physique d'oscillateur non linéaire comme le modèle de Van der Pol ou l'attracteur étrange du modèle de Rossler. Ce projet s'adresse à des étudiants ayant des bases en musique, en mathématiques et en physique mais pouvant être débutant en programmation informatique.

Table des matières

1 Résumé du projet **2**

2 On part d'un projet 'template' **2**

3 Synthétiseur monophonique avec un son de sinus **5**

4 Synthétiseur polyphonique avec un son sinus **6**

5 Synthétiseur par intégration de l'oscillateur de Van der Pol (optionnel) **7**

6 Synthétiseur basé sur le modèle chaotique de Rossler avec un attracteur étrange (optionnel) **11**

A Solutions aux exercices **13**

Remarque 0.1. Dans la version électronique de ce document (pdf), vous accédez aux pages de cours et autres documents en **cliquant sur les liens de couleur qui entourent le texte**.

Le travail à faire en TP est indiqué sous la forme « Exercice (TP) ». En préalable à la séance, on conseille de lire attentivement ce document, de faire les « Exercices (TD) ». Ne pas hésiter à demander de l'aide technique à ChatGpt par exemple.

1 Résumé du projet

Video de cette section.

On rappelle les faits suivants qui motivent ce projet :

— La perception auditive humaine « apprécie » les **signaux périodiques** en temps dans les intervalles de fréquence $[100\text{Hz}, 3000\text{Hz}]$, cela étant probablement du au fait que la voix humaine émet des signaux périodiques générés par la vibration des cordes vocales (voir [2, chap.3]).

— Par conséquent, les sons utilisés en musique sont de façon préférentielle périodiques (sauf pour les effets de percussions) et les instruments de musique sont conçus pour générer un son périodique (donc musical) selon l'un ou l'autre des mécanismes suivants (voir [2, chap.4]) :

1. « type système dynamique » : un système dynamique dissipatif non linéaire et possédant un attracteur qui est un cycle limite. Le mouvement de l'objet est donc périodique sur le cycle et par contact avec l'air cela produit un signal sonore périodique donc musical. C'est le cas du violon avec archet, trompette, flûte, voix, etc
2. « type onde 1D » : Une équation d'onde à une dimension. Dans une cavité 1D, une onde se propage de façon périodique. C'est le cas de la guitare, piano, etc

L'idée du projet est de simuler un instrument de « type système dynamique » (type 1), c'est à dire un système dynamique dissipatif non linéaire ayant un cycle limite comme attracteur et de l'utiliser pour générer un signal périodique audio en sortie du plugin. En entrée du plugin, ce système dynamique sera déclenché par un message MIDI 'note_on' où le numéro de touche (ex : 'C5') déterminera sa fréquence et la force (la vélocité) déterminera la non linéarité et donc le timbre du son.

Pour commencer on va utiliser le modèle simple et très connu qui est le modèle de Van der Pol (ou plus généralement appelé un système d'oscillations par relaxation, ou cycle limite).

Objectif : L'objectif est de créer un plugin AU (pour MacOS) ou VST3 (pour Windows et Linux) qui réalise ce synthétiseur, qui soit pleinement utilisable dans un DAW, avec une interface qui montre en temps réel la trajectoire sur l'attracteur du système dynamique, et quelques widgets pour contrôler les paramètres.

Remarque 1.1. En compléments, voici des liens sur le sujet de « modèles physiques de synthétiseurs », des exemples de sons à partir de l'attracteur de Lorenz : youtube, produits commerciaux Sur youtube, système chaotique en paramètres : youtube.

2 On part d'un projet 'template'

Video de cette section.

2.1 On télécharge le projet Template

« Template » signifie **modèle**. On va télécharger un projet de base qui est un plugin mais qui ne fait rien d'autre que de montrer les messages midi entrant et de jouer un son sinus fixe, mais que l'on va enrichir petit à petit dans les étapes suivantes.

Exercice 2.1. (TP) (Les explications sont pour Windows, mais avec MacOs ou Linux, c'est équivalent).

1. Sur le site juce_template.git,
   — cliquer sur Code/Télécharger_code_source_zip (par exemple)
   — sur votre ordinateur, avec le bouton droit, 'extraire l'archive' dans un répertoire de travail, par exemple '`$HOME/TP`'
   — Aller dans le nouveau répertoire C :\Users\$env :USERNAME\TP\juce_template-main. Fais un clic droit sur le fichier C1_win.ps1. Va dans les Propriétés. Si tu vois un bouton **"Débloquer"** en bas de la fenêtre, clique dessus et applique les changements. Cela permettra de l'executer.
2. Lancer **Visual Studio Code (VSC)**
   — Ouvrir une fenetre Terminal : Menu/View/Terminal.
   — Dans le terminal écrire (copier/coller) :

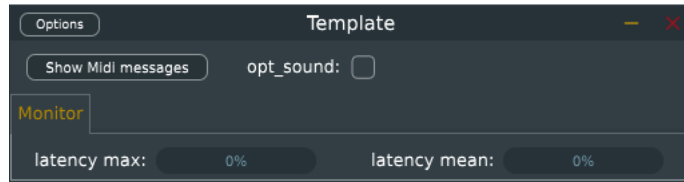
```
cd C:\Users\$env:USERNAME\TP\juce_template-main # change de directory
. .\C1_win.ps1 # execute le script
```

Cela génère et compile le projet pour faire un éxécutable.

3. **Executer le projet** : dans le terminal, écrire

```
. C:\Users\$env:USERNAME\TP\juce_template-main\build\Template_artefacts\Release\Standalone\Fred_Template.exe
```

Cela lance le programme appelé « StandAlone » (cela signifie que c'est un programme indépendant de tout DAW). On voit la fenêtre suivante appelée GUI ('Graphical User Interface') :



4. **Tester ce programme** :
   (a) Options/Audio_Midi_Settings/Output_Test pour entendre un son. Et sur l'interface, cliquer sur opt_sound pour **entendre le son fixe produit**, qui est un sinus de fréquence 440Hz.
   (b) Brancher un clavier externe (ou controller MIDI). Relancer le programme. Dans Options/Audio_midi_settings/Active_Midi_inputs cocher l'instrument. Sur l'interface, cliquer sur « Show_Midi_m et jouer des touches du clavier. **Observer les messages midi entrant.**

— **Remarque :** les deux Jauges du bas (latency) montrent le pourcentage occupé par les calculs par cycle du processor. Ils doivent rester inférieur à 30%. On verra cela plus loin.

2.2 Observation des messages midi entrant

Brancher un clavier externe. Avec le plugin « Template » de la section 2.1, vous pouvez observer les messages midi entrant. Il sont écrit en hexadécimal (voir section 1.5 du TP 1).

Voici tout d'abord des précisions sur quelques messages midi importants pour nous (on utilisera **la notation hexadécimale**) :

Note-on : c'est le message constitué de 3 nombres

$$\{90 + c, k, v\}$$

où

— $c \in \{0, \ldots, f\}$ est le numéro du canal midi utilisé par le clavier (attention sur le clavier l'affichage du canal est décalée de 1 et parcourt donc les nombres $1, \ldots, 16$ en décimal). En général le clavier est configuré sur le premier canal $c = 0$.

— $k \in \{0, \ldots, 7f\}$ est le numéro de touche. Par convention $k = 0$ est un 'C' de l'octave 0, très grave, que l'on notera C_0, et ensuite on augmente k de 1 pour chaque demi-ton. Ainsi $D\flat_0$ est la touche $k = 1$, D_0 est la touche $k = 2$, etc, C_1 est la touche $k = 12$ en décimal, C_5 est la touche $k = 5 \times 12 = 60$ en décimal et $3c$ en hexadécimal. Le La du diapason A_5 de fréquence $f_{A_5} = 440$Hz est la touche $k = 60 + 9 = 69$ en décimal et 45 en hexadécimal.

— $v \in \{0, \ldots, 7f\}$ est la vélocité de la touche. Cela correspondra à l'intensité du son.

Note-off : c'est le message

$$\{80 + c, k, v\}$$

avec c, k, v comme ci-dessus, mais ici la vélocité v correspondra à un temps d'extinction du son.

Exercice 2.2. (TD et TP) Quels seront les messages midi successifs si on joue l'accord de Do Majeur ? vérifier en TP avec le plugin « Template ». **Solution :** A.1.

Remarque 2.3. « Sur la numérotation des octaves ». Attention, on a appelé ici A_5 de l'octave 5 le La du diapason de fréquence $f = 440$Hz, pour la simple raison que sa touche MIDI est $k = 5 \times 12 + 9$. En solfège il est conventionnel de l'appeler A_3 , i.e. il y a un décalage de 2 octaves avec la convention MIDI.

2.3 Présentation rapide du code C++

Dans la suite nous allons lire et éventuellement modifier le code C++ du programme. Pour cela, nous allons utiliser le logiciel Visual Studio Code (VSC).

Nous avons expliqué en section 1.4 du TP1 que le plugin doit gérer les évènements audio et MIDI en temps réel avec une latence inférieur à 10ms (ou durée comparable). Or certains calculs peuvent prendre plus de temps, comme des dessins sophistiqués. Pour cette raison le plugin contient deux programmes qui tournent en parallèle, appelés **processus :**

— Un processus qui s'occupe des taches rapides 'temps réel' qui nécessite une petite latence, comme la gestion des évènements audio et midi. On appellera ce processus le '**processor**'.

— Un processus qui s'occupe des autres taches qui peuvent prendre plus de temps, comme la réalisation d'un dessin ou la communication avec l'utilisateur via des boutons et widgets du GUI ('Graphical User Interface'). On appellera ce processus l'**editor**.

Le code C++ du programme se trouve dans le répertoire **Source**. Vous y trouvez :

— Le sous répertoire **editor** qui contient les programmes pour le processus editor.

— Le sous répertoire **processor** qui contient les programmes pour le processus processor en temps réel.

La fonction processBlock : C'est la fonction principale du plugin, où le travail est fait. (Avec VSC, Menu/File/Open_File) ouvrir le fichier **Source/processor/PluginProcessor.cc** et y repérer la ligne suivante qui est le début d'une fonction c++ appelée processBlock :

```
void Processor::processBlock(AudioBuffer<float>& buffer, MidiBuffer& midiMessages)
{
        //.. ici il y a du code c++
}
```

Cette fonction du processus processor du plugin est appelée périodiquement par le DAW. Elle échange avec le DAW en entrée et sortie les objets suivants.

— **buffer** : est un tableau de n nombres réels à valeur dans l'intervalle $[-1, 1]$. Ce sont les données audio (par exemple venant d'un microphone). Ici on ne s'en occupe pas car notre plugin ne prend pas d'audio en entrée. Par contre **en sortie de la fonction on y mettra le signal audio** de notre synthétiseur que l'on souhaite envoyer aux haut parleurs.

— **midiMessages** : est un tableau contenant éventuellement des **messages midi en entrée**. C'est ce qui nous intéresse dans cette première étape du projet.

Remarque 2.4. Pour plus d'informations, la classe Processor est définie dans le fichier **Source/processor/PluginProcessor.h.** Elle dérive de la classe juce::AudioProcessor. Le buffer audio est de type AudioBuffer. Le buffer MIDI est de type MidiBuffer.

2.4 Modification du nom du projet

Dans la suite on va réaliser le projet avec plusieurs étapes. Il sera utile de conserver des étapes intermédiaires. Voici comment 'dupliquer' le projet actuel.

Exercice 2.5. (TP) Considérons ce projet qui est dans le répertoire 'juce_template-main'.

— On va copier le répertoire C :\Users\$env :USERNAME\TP\juce_template-main vers C :\Users\$env :USERNAME\TP\Synthe1. Pour cela utiliser le gestionnaire de fichier ou dans le terminal de VSC, écrire :

```
cd C:\Users\$env:USERNAME\TP
cp juce_template-main Synthe1 -Recurse
rm .\Synthe1\build # efface répertoire build
```

— Dans VSC, ouvrir les fichiers `C:\Users\$env:USERNAME\TP\Synthe1\C1_win.ps1` et `CMakeLists.txt` et partout changer `'juce_template-main'` et 'Template' par `'Synthe1'`

— Comme dans la partie 2, on compile et exécute ce projet. Pour cela, dans le terminal écrire :

```
cd C:\Users\$env:USERNAME\TP\Synthe1
.\C1_win.ps1
. C:\Users\$env:USERNAME\TP\Synthe1\build\Synthe1_artefacts\Release\Standalone\Fred_Synthe1.exe
```

3 Synthétiseur monophonique avec un son de sinus

Objectif : On va partir du projet 'Synthe1' comme expliqué dans la section 2.4. Dans cette première étape on va fabriquer un synthétiseur qui peut émettre une seule note à la fois (monophonique) en réponse à une touche du clavier. Le travail va donc consister à adapter la fréquence et l'amplitude du signal audio produit en fonction des messages midi entrant.

Exercice 3.1. (TD) « **Fréquence d'une note** ». A partir des informations données en section 2.2, trouver la formule mathématique qui donne la fréquence $f \in \mathbb{R}$ à partir du numéro de touche $k \in \mathbb{N}$. On rappelle que la note A_5 est la touche $k = 69$ de fréquence $f_{A_5} = 440\text{Hz}$ et que changer d'octave augmente k de $+12$ et multiplie la fréquence par $\times 2$. **Solution :** A.2.

Exercice 3.2. (TD) « **Synthèse additive, Harmoniques** »

1. Donner la formule qui exprime le signal sinusoïdal $s(t) \in \mathbb{R}$ de fréquence $f \in \mathbb{R}$ et amplitude $A > 0$ en fonction du temps $t \in \mathbb{R}$.
2. (Option) ajouter quelques termes « d'harmoniques », i.e. fréquences $2f$, $3f$, etc. (Cela correspond à faire de la Synthèse additive.)

Solution : A.3.

Exercice 3.3. (TP)

— Dans le fichier **Source/processor/PluginProcessor.cc,** juste avant la fonction processBlock(), ajouter la fonction c++ suivante, et modifier la formule par la formule de l'exercice 3.1, donnant la fréquence f à partir du numéro de touche k. Aide : 2^x s'écrit en c++ : pow(2., x)

```
//===========================
// input: k key MIDI value
// output: f frequency
double key_to_f(int k)
{
        double f = 440; // change this formula
        return f;
}
```

— Dans la fonction processBlock, à la place de la partie existante intitulée « `//- genere un son sinus` » ajouter les lignes de code suivantes

```
//----- Loop over input MIDI messages
for (const MidiMessageMetadata &metadata : midi_buffer)
{
        MidiMessage message = metadata.getMessage();

        if (message.isNoteOn())
        {
                int c = message.getChannel() - 1; // shift channel
                int k =  message.getNoteNumber();
                int v =  message.getVelocity();

                f = key_to_f(k);
                A = 0.1 *  v/127.;
                t = 0;
        }
        else if (message.isNoteOff())
        {
                int c = message.getChannel() - 1; // shift channel
                int k = message.getNoteNumber();
                int v = message.getVelocity();
                A = 0;
        }
}

//--- Audio parameters
int N = getSampleRate(); // nombre echantillons / sec.
int n = audio_buffer.getNumSamples(); // nombre echantillons dans le buffer
int nchan = audio_buffer.getNumChannels(); // 1: mono, 2: stereo

//.......fill audio buffer ..........
```

```
int ch = 0; // channel, mono
float* bufferPtr = audio_buffer.getWritePointer(ch); // pointeur du buffer,

for (int i = 0; i< n; i++) // audio samples
{
        bufferPtr[i] =  0; // put here the formula s(t)
        t = t + 1./N; // increment time
} // for i
```

— Modifier la ligne ci-dessus : `bufferPtr[i] = 0;` avec la formule de l'exercice 3.2.
— Dans le fichier **Source/processor/PluginProcessor.h,** ajouter la déclaration des variables f, A, t après le mot 'public' : (attention la ligne : double t = 0 ; est déjà présente, ne pas l'écrire deux fois).

```
public:
        double f = 440.; // frequency
        double A = 0.1; // amplitude
        double t = 0; // time
```

— Compiler et exécuter ce programme. Vérifier qu'il marche bien.

3.1 Solutions

Le projet demandé dans cette section se trouve ici : solution vanderpol_1.

4 Synthétiseur polyphonique avec un son sinus

Il nous faut gérer une liste des notes actives (qui sonnent). Après un message 'note_on' le programme doit ajouter une note à cette liste et après un message 'note_off' il doit enlever la note si elle est présente dans la liste. Pour chaque note active, le programme doit mémoriser :
— Le numéro de touche k et la vélocité v de la note qui sont des entiers.
— La date t depuis le début de la note et sa fréquence f qui sont des nombres réels.
Au niveau programmation c++ on va utiliser la classe vector qui permet de gérer des listes de nombres (ou autre).

4.1 Programmation

Exercice 4.1. (TP)

1. Dans le fichier **Source/processor/PluginProcessor.h,** enlever la déclaration des variables f, A, t qui ne servent plus et ajouter à la place les déclarations de listes qui contiendront les valeurs de k, v, t, f :

```
public:
        vector<int> L_k, L_v;
        vector<double> L_t, L_f;
```

2. Dans le fichier **Source/processor/PluginProcessor.cc,** avant la fonction processBlock, ajouter la fonction c++ suivante qui permet de savoir si une valeur k est présente ou non dans une liste L et à quel endroit.

```
//==========================
// input: k key MIDI value
// ouput:  index of the value k in vector L if found,
//          -1 if not found.
int Is_note_in_the_list(int k, vector<int>& L)
{
    auto it = std::find(L.begin(), L.end(), k);
    return (it != L.end()) ? std::distance(L.begin(), it) : -1;
}
```

3. Dans la fonction processBlock() si on détecte un évènement 'note_on', ajouter les lignes de code suivantes à l'endroit adéquate

```
if(Is_note_in_the_list(k, L_k) == -1) // note is not in the list
{
  L_k.push_back(k); // add k to the list L_k
```

```
    L_v.push_back(v);
    L_t.push_back(0);
    L_f.push_back(key_to_f(k));
  }
```

4. De même si on détecte un évènement 'note_off', ajouter les lignes de code suivantes à l'endroit adéquate

```
int pos = Is_note_in_the_list(k, L_k);
if(pos >= 0) // note is  in the list
{
  L_k.erase(L_k.begin() + pos); //remove element at position pos in the list L_k
  L_v.erase(L_v.begin() + pos);
  L_t.erase(L_t.begin() + pos);
  L_f.erase(L_f.begin() + pos);
}
```

5. Un peu plus loin dans le code c++, au moment de remplir la valeur bufferPtr[i] du buffer audio, écrire les lignes suivantes

```
bufferPtr[i] = 0; // init buffer
for(int ik = 0; ik < L_k.size(); ik++) // loop on active notes
{
        double f = L_f[ik];
        double A = 0.1 * L_v[ik] / 127.;
        double t = L_t[ik]; // time from start of the note
        bufferPtr[i] +=  0; // put here the formula s(t)
        L_t[ik] = t + 1./N; // increment time
} // for ik
```

   en remplaçant la ligne bufferPtr[i] += 0 par la formule de l'exercice 3.2.

6. Compiler et exécuter ce programme. Vérifier qu'il marche bien.

4.2 Solutions

Le projet demandé dans cette section se trouve ici : solution vanderpol_2.

4.3 Variantes pour aller plus loin :

1. Ajouter quelques termes « d'harmoniques », i.e. fréquences $2f$, $3f$, etc. (Cela correspond à faire de la Synthèse additive.)
2. Ajouter un vibrato et un tremolo aux notes (de fréquence 5 Hz et amplitude relative 0.2).
3. Ajouter une attaque des notes plus douce, et une fin de notes plus douce aussi.

5 Synthétiseur par intégration de l'oscillateur de Van der Pol (optionnel)

Dans cette section nous allons modifier le programme précédent et remplacer la formule explicite du signal sinusoidal $s(t)$ par la solution d'un modèle physique d'oscillations. On va commencer avec le modèle de Van der Pol. Il va falloir que le programme résolve (i.e. intègre) numériquement le champs de vecteur du modèle de Van der Pol et transforme la trajectoire en signal audio.

5.1 Champ de vecteur de Van der Pol

Pour un paramètre $\mu \in \mathbb{R}$ fixé, c'est un champ de vecteur dans le plan $\vec{x} = (x_1, x_2) \in \mathbb{R}^2 \rightarrow \vec{V} = (V_1, V_2) \in \mathbb{R}^2$ dont voici l'expression des composantes :

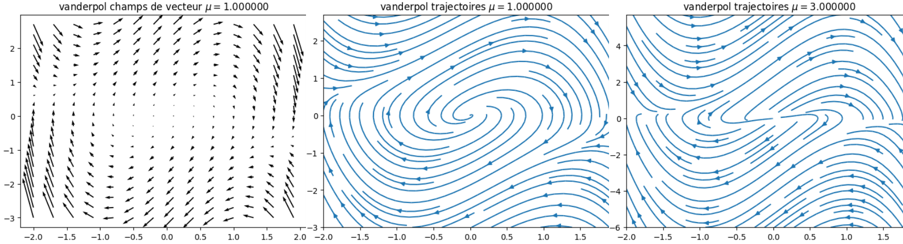
$$\begin{aligned} V_1(x_1, x_2) &= x_2 \\ V_2(x_1, x_2) &= \mu\left(1 - x_1^2\right) x_2 - x_1 \end{aligned} \tag{5.1}$$

Ce champs de vecteur détermine des **trajectoires** (ou flot) $\tau \in \mathbb{R} \rightarrow (x_1(\tau), x_2(\tau)) \in \mathbb{R}^2$ partant de conditions initiales quelconques $(x_1(0), x_2(0)) \in \mathbb{R}^2$ et d'après les équations de mouvement

$$\frac{dx_j}{d\tau} = V_j(x_1, x_2), \quad j = 1, 2. \tag{5.2}$$

Un tel modèle est appelé « **système dynamique différentiel** ou flot, défini par un champs de vecteur ».

Remarque 5.1. Voici le champs de vecteur et des portions de trajectoires dessinées dans le plan (x_1, x_2) par ce programme python, par exemple sur ce site en ligne, pour $\mu = 1$ et $\mu = 3$.



On observe que toutes les trajectoires sont attirés vers une trajectoire périodique, appelée cycle limite, qui se déforme si μ augmente. Plus tard on verra que sa période T_τ augmente avec μ, et $T_\mu \underset{\mu \to 0}{\rightarrow} 2\pi$.

Pour avoir un effet sonore intéressant, on décide de fixer le paramètre μ à la vélocité v de la note par la formule (par exemple)

$$\mu = 10(v/127) + 0.03. \tag{5.3}$$

5.2 Résolution numérique par la méthode de Euler

Numériquement on va résoudre les équations de mouvement (5.2) par la méthode de Euler. Pour cela on fixe un petit intervalle de temps $\delta_\tau = 10^{-2} \ll 1$, et on approxime

$$V_j \underset{(5.2)}{=} \frac{dx_j}{d\tau} \approx \frac{x_j(\tau + \delta_\tau) - x_j(\tau)}{\delta_\tau}, \quad j = 1, 2. \Leftrightarrow x_j(\tau + \delta_\tau) = x_j(\tau) + \delta_\tau \times V_j \tag{5.4}$$

Donnant l'algorithme suivant.

- On pose $X_1 = x_1(0), X_2 = x_2(0)$: conditions initiales
- On itère N_τ fois les étapes suivantes
  1. On calcule V_1, V_2 d'après (5.1)
  2. On déduit les nouvelles valeurs d'après (5.4) :

$$X_1 = X_1 + \delta_\tau V_1$$
$$X_2 = X_2 + \delta_\tau V_2$$

5.3 Relation entre le temps τ du modèle et le temps t du signal

Le paramètre d'évolution du modèle noté τ est différent du temps t utilisé pour le signal audio. On va relier τ et t par $t = c\tau$ avec une constante $c > 0$ qui va dépendre de la fréquence f souhaitée de la note. On note T_τ la période du cycle limite et $T = \frac{1}{f}$ la période de la note. On souhaite donc $T = cT_\tau$ donc $c = \frac{T}{T_\tau} = \frac{1}{fT_\tau}$.

Rappelons que le temps entre deux échantillons audio est noté

$$\delta = 1/N$$

avec $N = 44100$Hz. Pour l'intégration numérique, on va **diviser cette durée δ en $M \in \mathbb{N}^*$ parties.**

Exercice 5.2. (TD) « Intégration, méthode de Euler »

1. Déduire l'expression de δ_τ à partir de f, T_τ, M, N.
2. On imposera que δ_τ soit le plus grand avec la contrainte $\delta_\tau < \frac{1}{N_{\text{Euler}}}$ avec un entier $N_{\text{Euler}} = 100$ fixé. Déduire l'expression de M.

Solution : A.4

Dans un premier temps, on ne connaît pas la période T_τ et on va donc mettre la valeur arbitraire $T_\tau = 2\pi$. Plus tard on la mesurera précisément.

5.4 Transformation de la trajectoire en signal audio

A la fin, on transforme la trajectoire obtenue $\vec{x}(t) = (x_1(t), x_2(t))$ en signal audio en prenant la première composante par exemple $s(t) = Ax_1(t)$, avec une constante A correspondant au volume (on pourrait aussi considérer une formule $\vec{x}(t) \to s(t)$ plus compliquée ou au choix de l'utilisateur).

5.5 Programmation

Exercice 5.3. (TP)

1. Pour **dessiner la trajectoire du système dynamique** sur le GUI, il faut au préalable faire les étapes suivantes
   - Télécharger les fichiers graphics.cc et graphics.h et les sauvegarder dans le répertoire « Source/editor/ »
   - Télécharger les fichiers com.cc et com.h et les sauvegarder dans le répertoire « Source » (en replaçant les fichiers existants).
   - Avec le logiciel Projucer, ouvrir le fichier *.jucer et dans la colonne de gauche « File_Explorer » avec le bouton droit sur « Source » cliquer sur « Add Existing Files » et ajouter ces deux fichiers graphics.cc et graphics.h.
   - Au début du fichier Source/editor/manager.h sous l'instruction #pragma once, ajouter la ligne :

     #include "Source/editor/graphics.h"
2. Dans le fichier **Source/processor/PluginProcessor.h,** ajouter les déclarations suivantes

```
//... for VanDerPol
double mu = 4; // parameter of the model
int N_Euler = 100; // number of samples in a period for Euler integration method
vector<double> L_x1, L_x2; // position of the point for each active note

//.... for copy of the trajectory to the editor
int N_tau = 1000; // number of iterations
vector<double> L_x1_copy, L_x2_copy, L_x3_copy; // accessed with atomic variable changes_Lx_copy
int i_copy; // index
atomic<bool> changes_Lx_copy = false; // 1: ask to refresh display
```

3. Dans le fichier **Source/processor/PluginProcessor.cc,** dans la fonction processBlock, à l'endroit où il y a détection d'une nouvelle note ('note_on'), ajouter les lignes suivantes :

```
L_x1.push_back(0.1);
L_x2.push_back(0);
i_copy = 0;
```

et si il y a un message 'note_off', ajouter les lignes suivantes :

```
L_x1.erase(L_x1.begin() + pos);
L_x2.erase(L_x2.begin() + pos);
```

Ajouter aussi dans la fonction processBlock,

```
//-----for editor --------------------
if(L_x1_copy.size() != N_tau)
{
        L_x1_copy.resize(N_tau);
        L_x2_copy.resize(N_tau);
}
```

Un peu plus loin, dans la boucle sur les notes actives décrite en 5, écrire à la place :

```
for(int ik = 0; ik < L_k.size(); ik++) // loop on active notes
{
        double f = L_f[ik];
        double T_tau = 2*M_PI; // approximate period of the cycle

        double mu = (L_v[ik] / 127.) * 10. + 0.03;  // map v \in [0,127] to mu \in [0,10]
        int M = floor(N_Euler * f * T_tau / N); // subdivision of time step
        if(M<1)
                M=1;

        double x1 = L_x1[ik], x2 = L_x2[ik];
        double  delta_tau = f*T_tau/(M*N);

        for(int im=0; im<M; im++) // subloop
        {
```

```
                        //. formula for the vector field
                        double V1 = x2;
                        double V2 = mu * (1 - x1*x1) * x2 -x1;
                        double x2_old = x2;
                        x1 = x1 + delta_tau * V1;
                        x2 = x2 + delta_tau * V2;

                        //... copy to the editor
                        if(ik == (L_k.size()-1)) // consider last note only
                        {
                                if(changes_Lx_copy.load()  == false) // editor needs a new copy
                                {
                                        L_x1_copy[i_copy]= x1;
                                        L_x2_copy[i_copy]= x2;
                                        i_copy++;
                                }

                                if(i_copy > N_tau) // end of the copy
                                {
                                        i_copy = 0;
                                        changes_Lx_copy.store(true); // copy is done
                                }
                        }
                } // for im

                L_x1[ik] = x1; // memorizes
                L_x2[ik] = x2;

                double A = 0.02 * (L_v[ik] / 127.);
                bufferPtr[i] += A* L_x1[ik];
        } // for ik
```

4. Dans le fichier **Source/editor/manager.h,** ajouter les déclarations suivantes

```
//------
void Dessin(juce::Graphics& g); // make_gui =  nl Window2(ZC, "Phase space", 500, 400)  help="Show
TCanvas c; // window associated to Dessin()
```

5. Dans le fichier **Source/editor/manager.cc,** ajouter les lignes suivantes

```
//======================
void Manager::Dessin(juce::Graphics& g)
{
        c.cx = p_com->Manager_Dessin->getX();
        c.cy = p_com->Manager_Dessin->getY();
        c.wx = p_com->Manager_Dessin->getWidth();
        c.wy = p_com->Manager_Dessin->getHeight();
        g.fillAll(Colour((uint)kBlack));
        c.Range(-2, -10, 2, 10); // x1,y1,x2,y2

        //-----Draw the trajectory-------------
        for(int j=0; j< processor->L_x1_copy.size(); j++)
        {
                TMarker m(processor->L_x1_copy[j], processor->L_x2_copy[j], 3);
                m.SetMarkerColor(kYellow);
                m.Draw(c, g);
        }
}
```

6. **Compiler et exécuter ce programme.** Observer que les **notes ne sont pas justes** et si on appuie plus fort sur une touche, la fréquence diminue. Cela est dû au fait que le paramètre μ augmente d'après la remarque 5.1, et donc la période T_τ de la trajectoire augmente d'après la remarque 5.1. La période $T = 1/f$ du signal augmente donc et sa fréquence $f = 1/T$ diminue. Pour corriger cela et stabiliser les fréquences de notes à la valeur souhaitée, il va falloir mesurer la période T_τ et compenser cet effet.

5.6 Mesure de la période T_τ du cycle limite

D'après les figures de la remarque 5.1, on observe que le cycle limite tourne dans le sens indirect. Pour mesure la période du cycle limite, on va détecter les instants lorsque la trajectoire traverse le demi axe $x_2 = 0$ avec $x_1 > 0$.

Exercice 5.4. (TP)

1. Dans le fichier **Source/processor/PluginProcessor.h,** ajouter les déclarations suivantes

```
vector<double> L_tau, L_tau_cross_old, L_T_tau; // to measure the period
```

2. Dans le fichier **Source/processor/PluginProcessor.cc,** dans la fonction processBlock, à l'endroit où il y a détection d'une nouvelle note ('note_on'), ajouter les lignes suivantes :

```
L_tau.push_back(0); // time from start of note
L_T_tau.push_back(2*M_PI); // period of the cycle
L_tau_cross_old.push_back(0); // date of cross section
```

et si il y a un message 'note_off', ajouter les lignes suivantes :

```
L_tau.erase(L_tau.begin() + pos);
L_T_tau.erase(L_T_tau.begin() + pos);
L_tau_cross_old.erase(L_tau_cross_old.begin() + pos);
```

Juste après le calcul de la position (x_1, x_2) du point, ajouter les lignes suivantes :

```
//.. look if we cross the positive axis x1
if(x1 >0 && (x2*x2_old <0))
{
        double tau_cross =  L_tau[ik] + (x2/(x2_old-x2))*delta_tau; // date of cross section
        double T_tau  = tau_cross - L_tau_cross_old[ik]; // measured period
        if(T_tau<0.1)
                T_tau = 2*M_PI;
        L_T_tau[ik]= T_tau; //memorize period
        L_tau_cross_old[ik] = tau_cross; // memorize date of cross section
} // if
L_tau[ik] += delta_tau; // increment effective time
```

et plus haut, remplacer la ligne

```
double T_tau = 2*M_PI; // approximate period of the cycle
```

par

```
double T_tau = L_T_tau[ik]; // measured period of the cycle
```

3. **Compiler et exécuter ce programme.** Observer que maintenant les **notes sont justes** ! et que le timbre des notes dépend de μ et donc de la force de la frappe (vélocité).

5.7 Variantes pour aller plus loin :

— Modifier la formule $v \to \mu$ en (5.3) pour que le son soit plus doux et seulement très timbré si on appuye fort. Par exemple

$$\mu = 10\,(v/127)^E + 0.01.$$

avec un exposant $E > 0$. Avant on avait $E = 1$. Comme $(v/127) < 1$, on déduit que prendre $E > 1$ va rendre le son plus doux (par exemple $E = 3$), et $E < 1$ le rendre plus timbré.

5.8 Solutions

Le projet demandé dans cette section se trouve ici : solution vanderpol_3.

6 Synthétiseur basé sur le modèle chaotique de Rossler avec un attracteur étrange (optionnel)

Le système dynamique précédent de Van Der Pol a un attracteur qui est un cycle limite. Cela fait que le son produit sera un signal parfaitement périodique et donc peut être un peu ennuyeux d'un point de vue musical. On peut améliorer cela en considérant un système dynamique où l'attracteur sera plus plus complexe comme un « attracteur étrange » mais avec un comportement « presque périodique dominant » afin de donner une impression de note musicale.

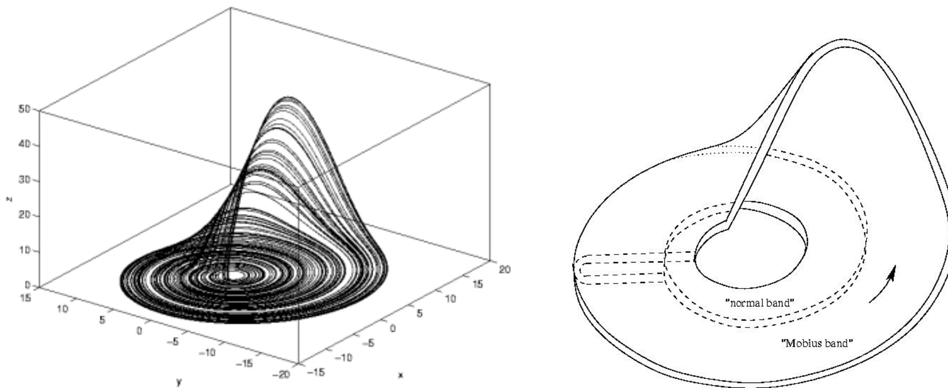
6.1 Champ de vecteur de modèle de Rossler

On propose d'utiliser le modèle de Rossler qui dépend de trois paramètres $a, b, c \in \mathbb{R}$. C'est un champ de vecteur dans l'espace $\vec{x} = (x_1, x_2, x_3) \in \mathbb{R}^3 \to \vec{V} = (V_1, V_2, V_3) \in \mathbb{R}^3$ dont voici l'expression des composantes :

$$V_1(x_1,x_2,x_3) = -x_2 - x_3 \tag{6.1}$$
$$V_2(x_1,x_2,x_3) = x_1 + a x_2$$
$$V_3(x_1,x_2,x_3) = b + x_3(x_1 - c) \tag{6.2}$$

Comme expliqué dans la section 5.1, ce champs de vecteur génère des trajectoires $t \to \vec{x}(t)$ d'après les équations de mouvement $\frac{d\vec{x}}{dt} = \vec{V}(\vec{x}(t))$. Numériquement on intègre ces équations par la méthode de Euler.

Voici une image de l'attracteur pour les paramètres $a = b = 0.2$, $c = 5.7$. Les trajectoires, tournent dans le sens direct. Observer leur devenir qui semble imprévisible (et qui l'est !). Malgré l'apparence simple des équations, le comportement des trajectoires est très subtile, « chaotique » et fait l'objet de recherches actuelles.



Si on change les paramètres a, b, c, la géométrie de l'attracteur change, il y a des phénomènes appelés « bifurcations ». Voici une description de la géométrie de l'attracteur si $a = b = 0.1$ et c varie. D'après ces figures, pour avoir de la dynamique dans le son, on pourra relier la velocité v de la note au paramètre c selon

$$c = 4 + 16\left(\frac{v}{127.}\right)^3. \tag{6.3}$$

Référence : voir les animations et le cours de système dynamique [1, chap. Lorenz]. Voici une page interactive sur le flot de Lorenz.

6.2 Programmation

Exercice 6.1. (TP)

1. Dans le fichier **Source/processor/PluginProcessor.h,** ajouter les déclarations suivantes
```
double a=0.1, b=0.1, c=18; // parameters of the model Rossler
```
2. Dans les fichiers **Source/processor/PluginProcessor.h,** et **Source/processor/PluginProcessor.cc,**
   (a) ajouter tout ce qu'il faut pour travailler maintenant avec trois composantes (x_1, x_2, x_3), (V_1, V_2, V_3) alors qu'avant on n'avait que deux composantes.
   (b) Remplacer les expressions du champ de vecteur par (6.1).
   (c) Ajuster le paramètre c à la velocité selon (6.3).
3. Pour le dessin de l'attracteur,
   (a) dans le fichier **Source/editor/manager.cc,** dans la fonction Dessin(), ajuster le cadre x_1, y_1, x_2, y_2 du dessin d'après les figures ci-dessus.
   (b) dans **Source/editor/manager.h,** augmenter le nombre de points : int N_tau = 10000 ;
4. **Compiler et exécuter ce programme.**

6.3 Variantes pour aller plus loin

— Pour améliorer le son, on peut ajouter une enveloppe, une réponse au pitch-bend, au breath controler, et autres effets.

— On peut explorer l'espace des paramètres a, b, c et leur effets sur l'attracteur pour modifier le son de façon dynamique.

6.4 Solutions

Le projet demandé dans cette section se trouve ici : solution vanderpol_4.

A Solutions aux exercices

Solution A.1. de l'exercice 2.2, « messages MIDI »

```
90,40,36,
90,43,33,
90,3c,37,
80,40,40,
80,43,40,
80,3c,40,
```

Solution A.2. de l'exercice 3.1, « Fréquence d'une note »

$$f = 440 \times 2^{(k-69)/12.0}$$

```
double f = 440.0 * pow( 2.0, (k-69)/12.0 );
```

Solution A.3. de l'exercice 3.2, « Synthèse additive, Harmoniques »

1. C'est
$$s(t) = A\sin(2\pi f t)$$
2. Par exemple pour les trois premières harmoniques
$$s(t) = A\left(\sin(2\pi f t) + \frac{1}{2}\sin(2\pi(2f)t) + \frac{1}{3}\sin(2\pi(3f)t)\right)$$

Solution A.4. de l'exercice 5.2, « Intégration, méthode de Euler »

1. Le pas d'intégration δ_τ utilisé en (5.4) est
$$\delta_\tau = \frac{1}{c}\left(\frac{\delta}{M}\right) = \frac{fT_\tau}{MN}, \tag{A.1}$$
2. On choisit $M \geq 1$ le plus petit entier de sorte que
$$\delta_\tau < \frac{1}{N_{\text{Euler}}} \Leftrightarrow \frac{fT_\tau}{MN} < \frac{1}{N_{\text{Euler}}} \Leftrightarrow M > N_{\text{Euler}}\frac{fT_\tau}{N}.$$
Donc
$$M = \max\left(1, \left[N_{\text{Euler}}\frac{fT_\tau}{N}\right]\right),$$
où $[.]$ signifie la partie entière.

References

[1] F. Faure. *Cours de systémes dynamiques.* link, 2016.

[2] F. Faure. *Cours d'acoustique musicale. Niveau Licence 3.* link, 2020.