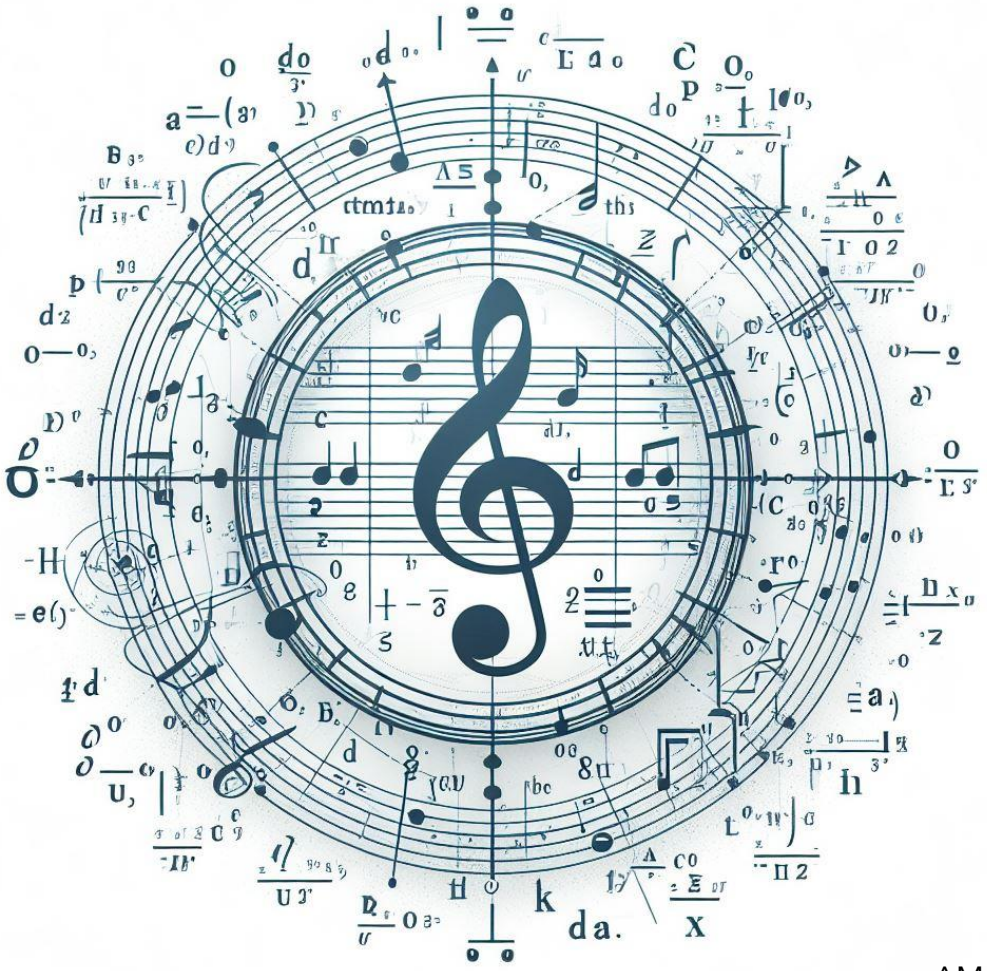
Chaos-ic: tune the chaos, create the symphony!



Jorge Arbesú Nieto

David Calleja Merino

Sandra Domínguez Gómez

Sofía Mesón Pérez

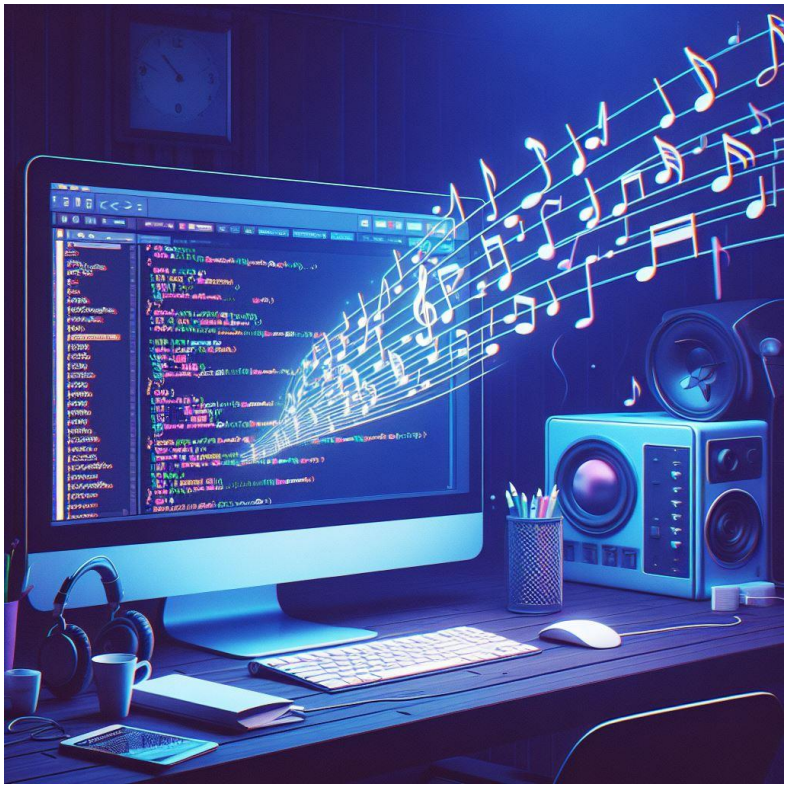
Juan Olalla Pombo

AM1

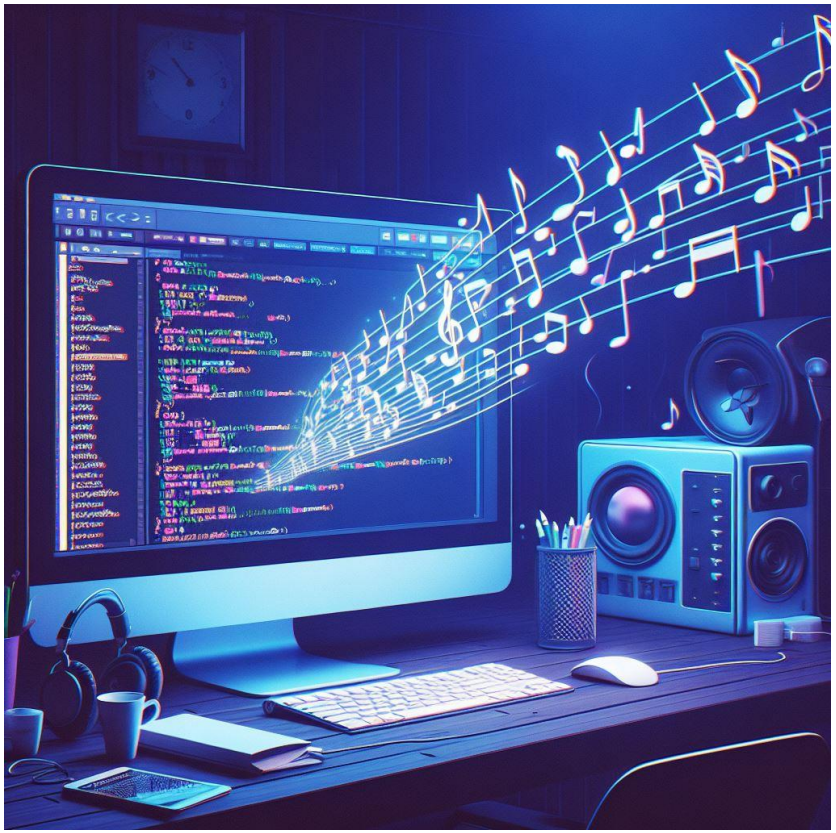
1º MUSE

Curso 2023/24

1.Introduction
2.GUI
3.Algorithm
4.MIDI file
5.Examples
6.Conclusions



- Automatic-generated music:
  - Literature, algorithms, …
- Fractals
- Academic interest: a different way of studying the behaviour of the chaotic dynamic systems



GUI (I)

- Library to develop GUI: Thinker
- Objetives:
  - **Define variables** to be stored.
  - Create **labels** for each block.
  - Use a **combobox** to choose options.
  - Define parameters to be choosen.
  - Define to enter" data.
  - Create dictionaries"float to store variables.
  - Use variable "**self.__**" to assign variables.

tk.Label

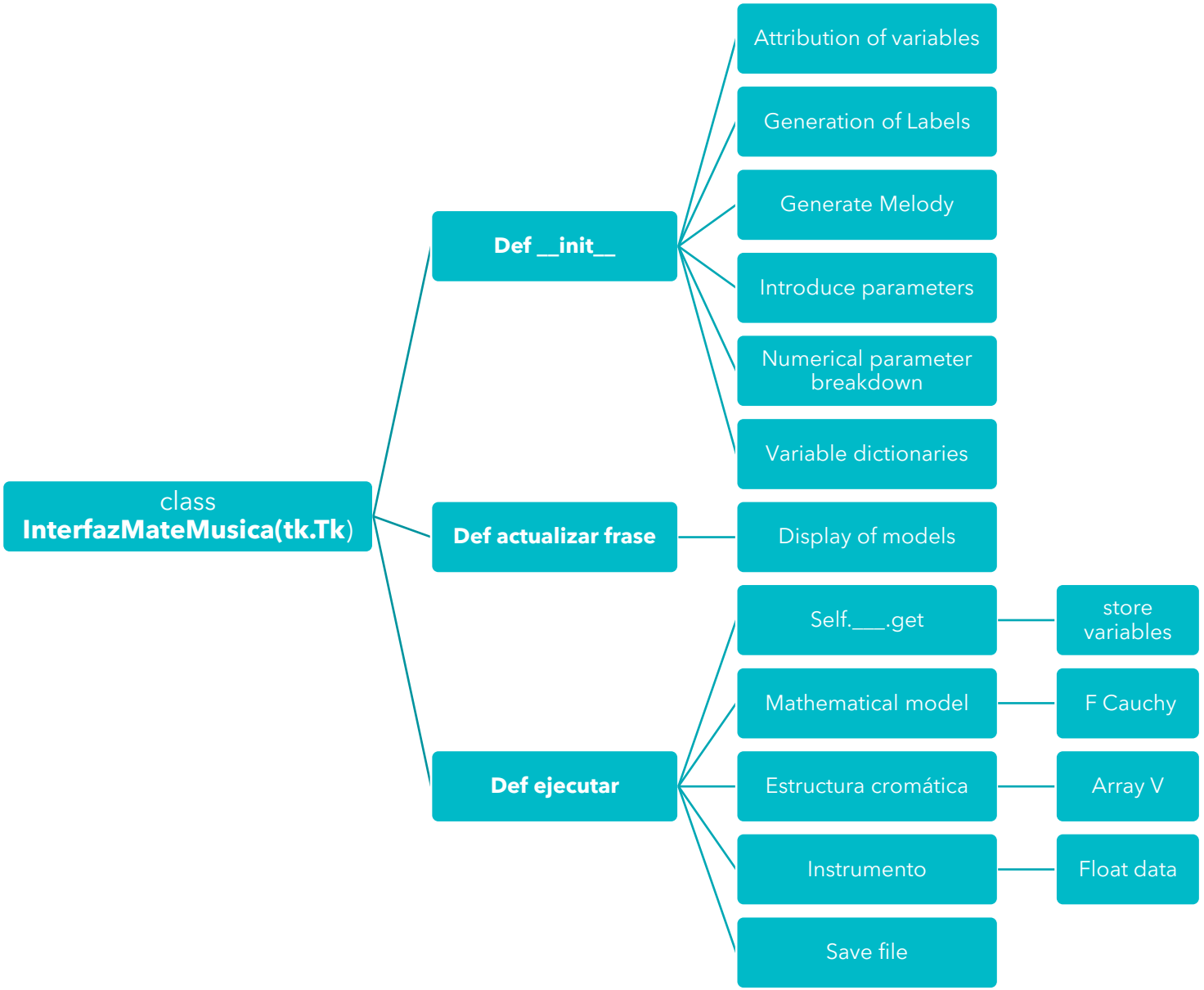
Tk.Combobox

tk.Entry

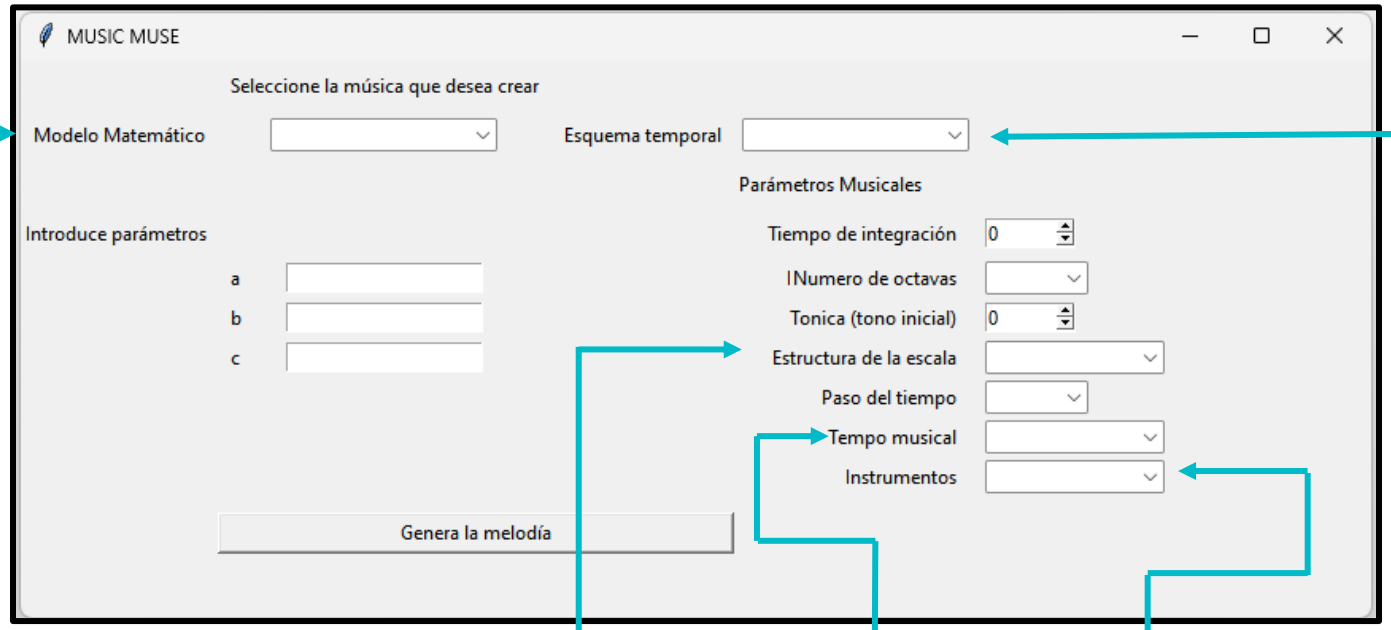
- Code Secuence:

def **__init__** → def **actualizar_frase** → Def **ejecutar**

2 GUI: Function Tree



Combobox



- Rossler
- Rayleigh
- Lorentz
- Van Der Poel
- Duffing

-Cromática
- Mayor
- Menor natural
- Menor armónica
- Menor

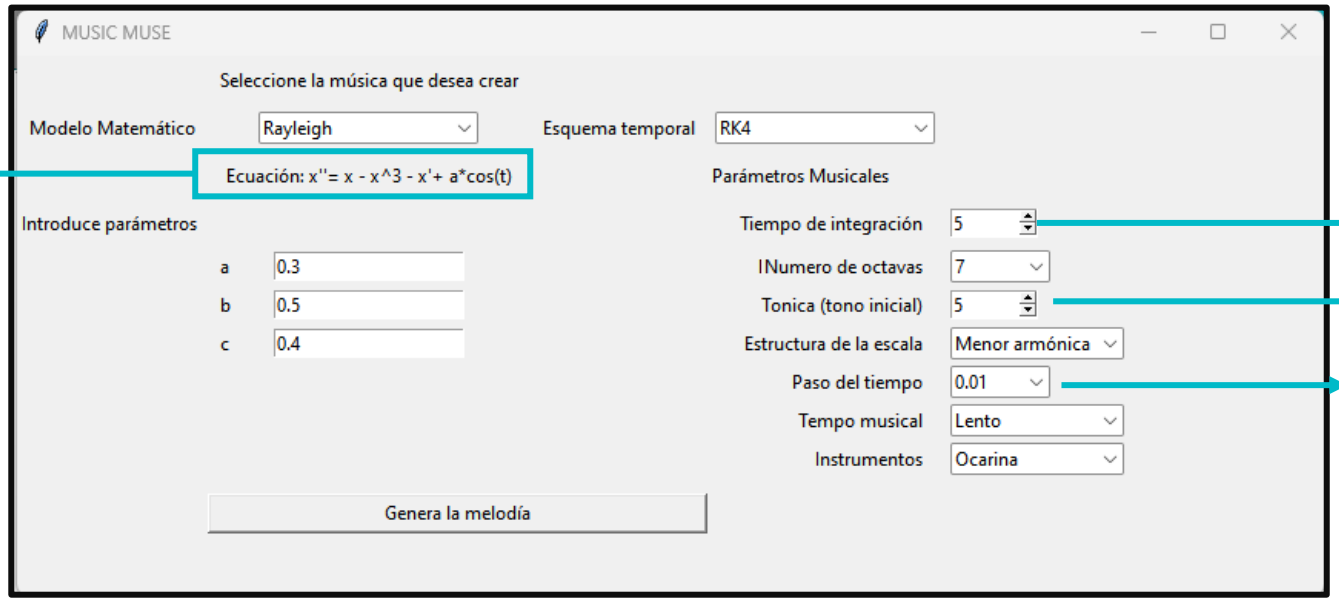
-Rápido
-Suave
-Lento

- Piano
- Guitarra eléctrica
- Ocarina

- Euler
- Euler Inverso
- Runge Kutta 4
- Crank Nicolson

GUI: User Interface Appearance (I)

Grafical User Interface (GUI)



Information about the model

[1,60]

[1,7]

[0.1, 0.01, 0.001]

ALGORITHM

- Main idea:

Transformation

Mathematical Method Solution

- X = ()
- Y = ()
- Z = ()

Musical values

- Frequency
- Rhythm
- Dynamic

ALGORITHM: Frequency

Normalization → VMP → Transformation into MIDI values

```
'''Normalizacion de la variable'''
alpha = (2**k-1)/(max(x)-min(x))
beta = -alpha*min(x)+1

for i in range(0,len(x)):
    x1[i] = alpha*x[i]+beta
```

```
'''Mapeo al valor mas proximo'''
mu=2**(lamnbda/6)-1
D=empty((len(x1),len(E1)))
for i in range(0,len(x1)):
    for j in range (0,len(E1)):
        if abs(x1[i]-E1[j])<=mu:
            D[i,j]=0
        else:
            D[i,j]=x1[i]

L=[]
for i in range(0,len(x1)):
    L.append(min(D[i,:]))
```

```
f=55*2**((tau+12*0-10)/12)

'''Frecuencias de las notas'''
F=[]
for i in range(0,len(x1)):
    F.append(f*L[i])

'''valores de la frecuencia para MIDI'''
theta=[]
for i in range(0,len(F)):
    if abs(F[i]) / 440 <= 0:
        theta.append(69)
    else:
        theta.append(69 + 12 / log(2) * log(abs(F[i]) / 440))
theta2=[]
for i in range(0,len(theta)):
    theta2.append(round(theta[i]))
```

ALGORITHM: Rhythm

Normalization → Round → Transformation into MIDI values

```
'''Normalizacion de la variable'''
alpha = (max(R)-min(R))/(max(y)-min(y))
beta = -alpha*min(y)+min(R)
y1 = zeros(len(y))

for i in range(0,len(y)):
    y1[i] = alpha*y[i]+beta
```

```
'''Redondeo a valor proximo'''
y2=[]
for i in range(0,len(y)):
    y2.append(round(y1[i]))
```

```
'''Conversion a MIDI'''
Y=[]
for i in range(0,len(y2)):
    Y.append(60/tp*2**y2[i]/2**4)
```

```
R=[0,1,2,3,4,5,6]
'''
0 - semifusa
1 - fusa
2 - semichorchea
3 - corchea
4 - negra
5 - blanca
6 - redonda
'''
```

ALGORITHM: Dynamic

Normalization → VMP → Transformation into MIDI values

```
'''Normalizacion de la variable'''
alpha = (max(U)-min(U))/(max(z)-min(z))
beta = -alpha*min(z)+min(U)
z1 = zeros(len(z))

for i in range(0,len(z)):
    z1[i] = alpha*z[i]+beta
```

```
'''Mapeo al valor mas proximo'''
mu=10
D=empty((len(z),len(U)))
for i in range(0,len(z)):
    for j in range (0,len(U)):
        if abs(z[i]-U[j])<=mu:
            D[i,j]=0
        else:
            D[i,j]=z[i]

L=[]
for i in range(0,len(z)):
    L.append(min(D[i,:]))
```

```
Z=[]
for i in range(0,len(L)):
    if max(L) != 0:
        Z.append(L[i]/max(L))
    else:
        Z.append(100)
```

```
U=[10, 30, 45, 60, 75, 92, 108, 127]
'''
0-10 ppp
11-30 pp
31-45 p
46-60 mp
61-75 mf
76-92 f
93-108 ff
109-127 fff
'''
```

get_save_path

From *tkinter*:
Creates a file dialog for choosing the location and name of the MIDI file

```python
def get_save_path():
    root = tk.Tk()
    root.withdraw()  # Ocultar la ventana principal

    # Mostrar el diálogo para elegir la ubicación y el nombre del archivo
    file_path = filedialog.asksaveasfilename(defaultextension=".mid", filetypes=[("MIDI files", "*.mid")])

    return file_path
```

Called from GUI
`def ejecutar(self)`

Output file: the path where the MIDI file will be saved
X : **Frequencies** - notes [0..127]
Y: Note Duration-**rhythm** (seg or beats->BPM)
Z: Speed / intensity and musical **dynamic** [0..127]
Instrument: MIDI code

create_midi_file

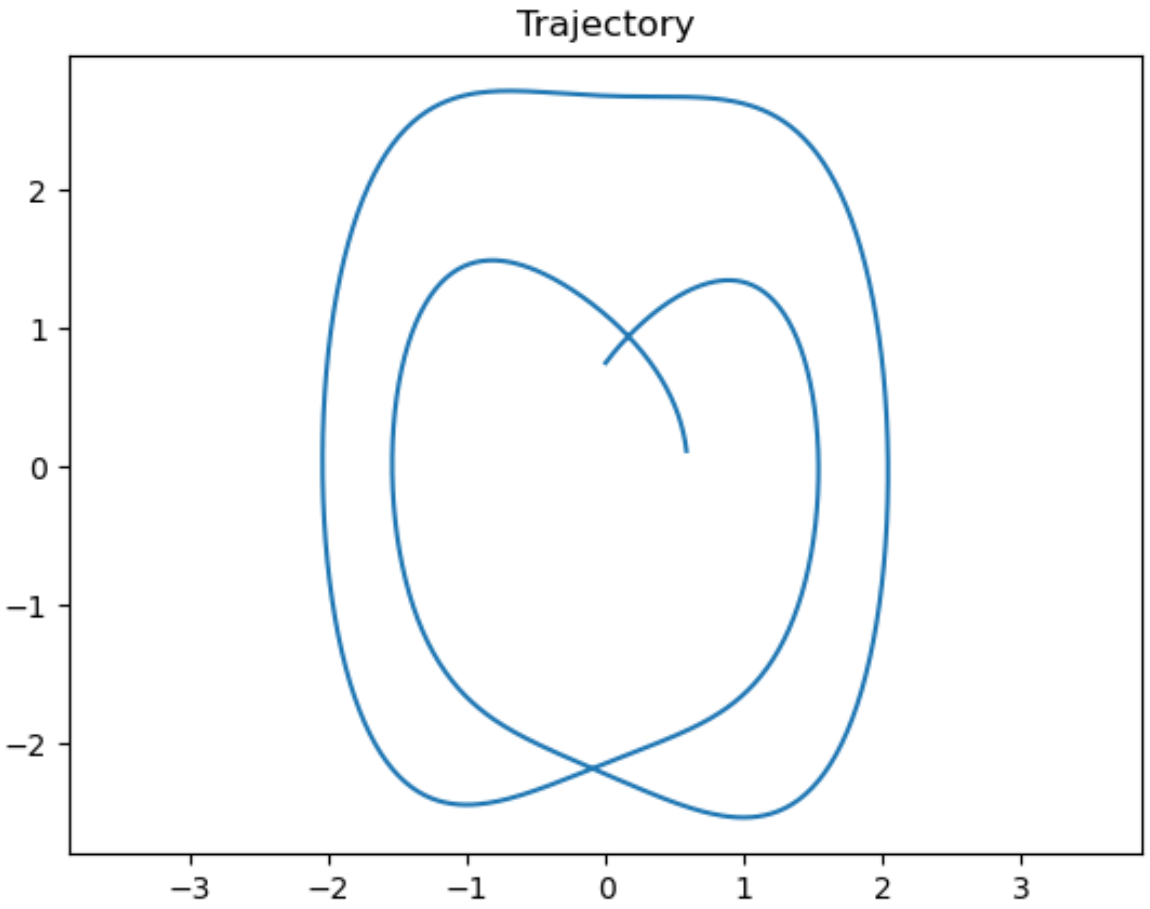
```python
def create_midi_file(output_file, vector_nota, vector_velocidad, vector_bpm, Instrumento):

    # Crear un archivo MIDI
    midi = MidiFile()

    # Agregar una pista al archivo MIDI
    track = MidiTrack()
    midi.tracks.append(track)

    # Configurar el canal y el instrumento (opcional)
    track.append(Message('program_change', program=Instrumento))  # Cambiar el instrumento

    # Agregar mensajes MIDI a la pista (secuencia de notas)
    for nota, velocidad, bpm in zip(vector_nota, vector_velocidad, vector_bpm):
        tiempo_en_segundos =  bpm  # Calcular el tiempo en segundos a partir de bpm
        tiempo_en_tics = int(tiempo_en_segundos * midi.ticks_per_beat)  # Convertir a tics de tiempo MIDI
        track.append(Message('note_on', note=nota, velocity=velocidad, time=0))  # Encender la nota
        track.append(Message('note_off', note=nota, velocity=velocidad, time=tiempo_en_tics))  # Apagar la nota

    # Guardar el archivo MIDI
    midi.save(output_file)
```

From *mido*:

1. Creates a Midifile
2. Adds a track to the file
3. Configures the instrument
4. Add the notes with its duration and speed to the track based on input vectors (X, Y, Z)

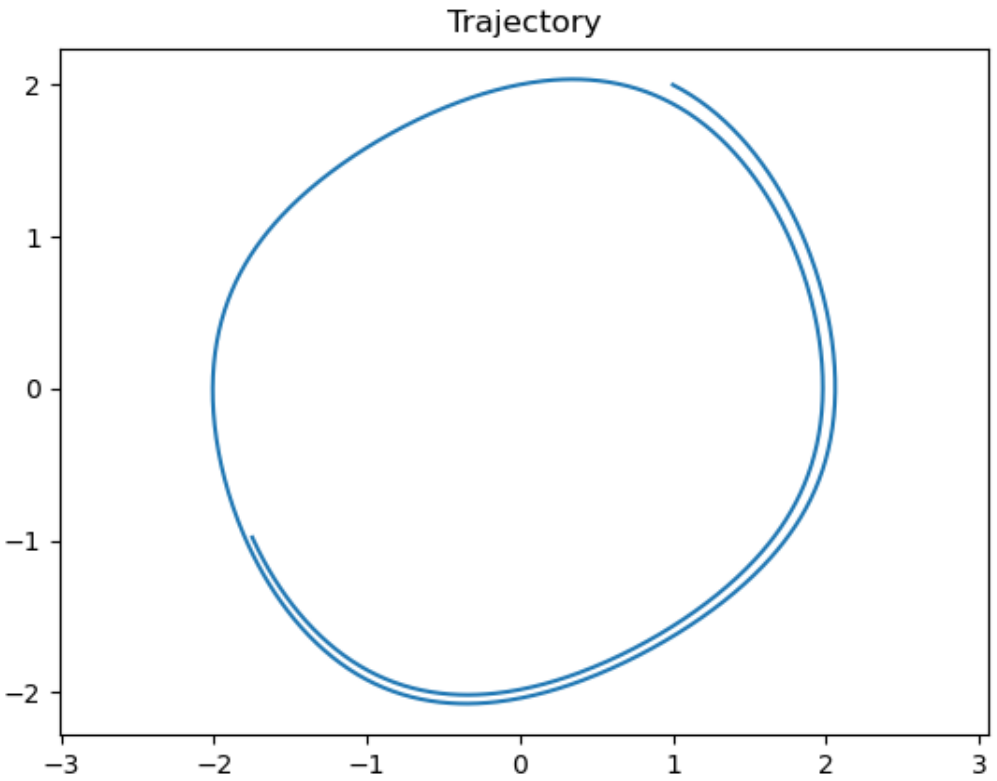
EXAMPLES: Duffing

- Integrated with **Euler method**
- Initial Conditions: (0, 0.75)
- K parameter: 0



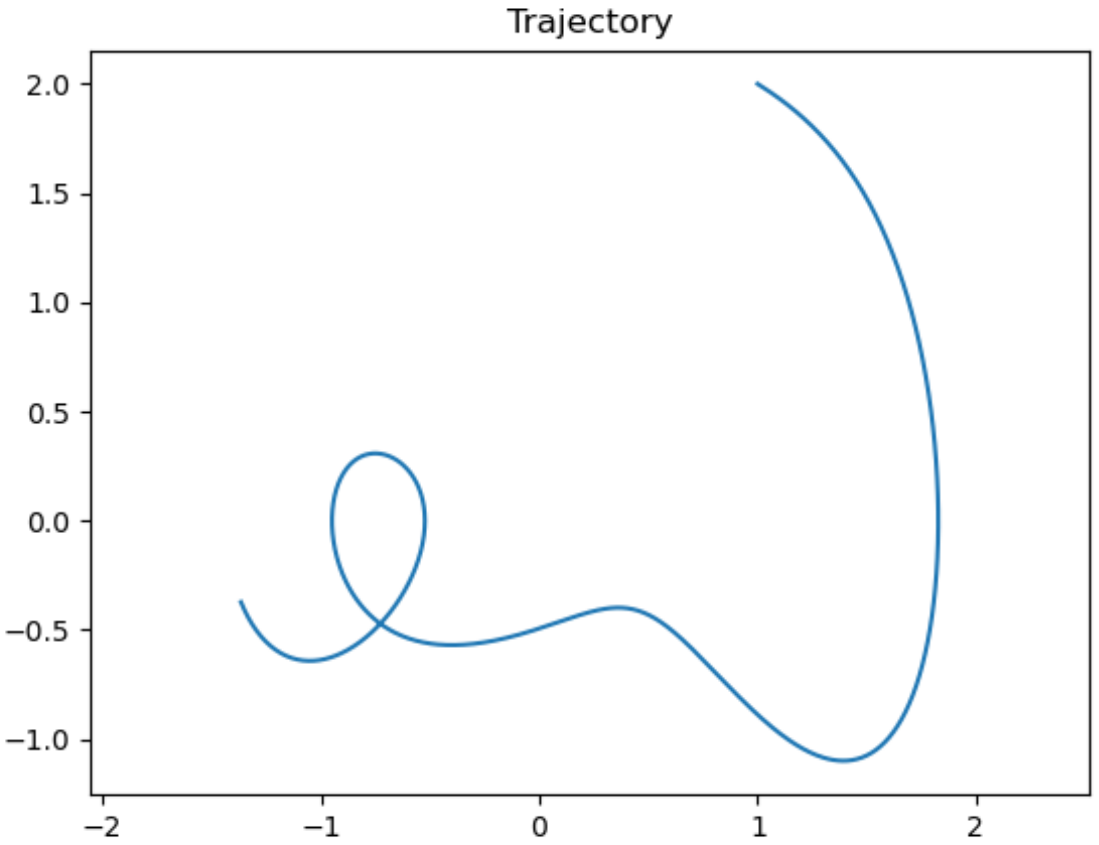
EXAMPLES: Van der Poel

- Integrated with **Inverse Euler method**
- Initial Conditions: (1, 2)
- μ Parameter = 0.2



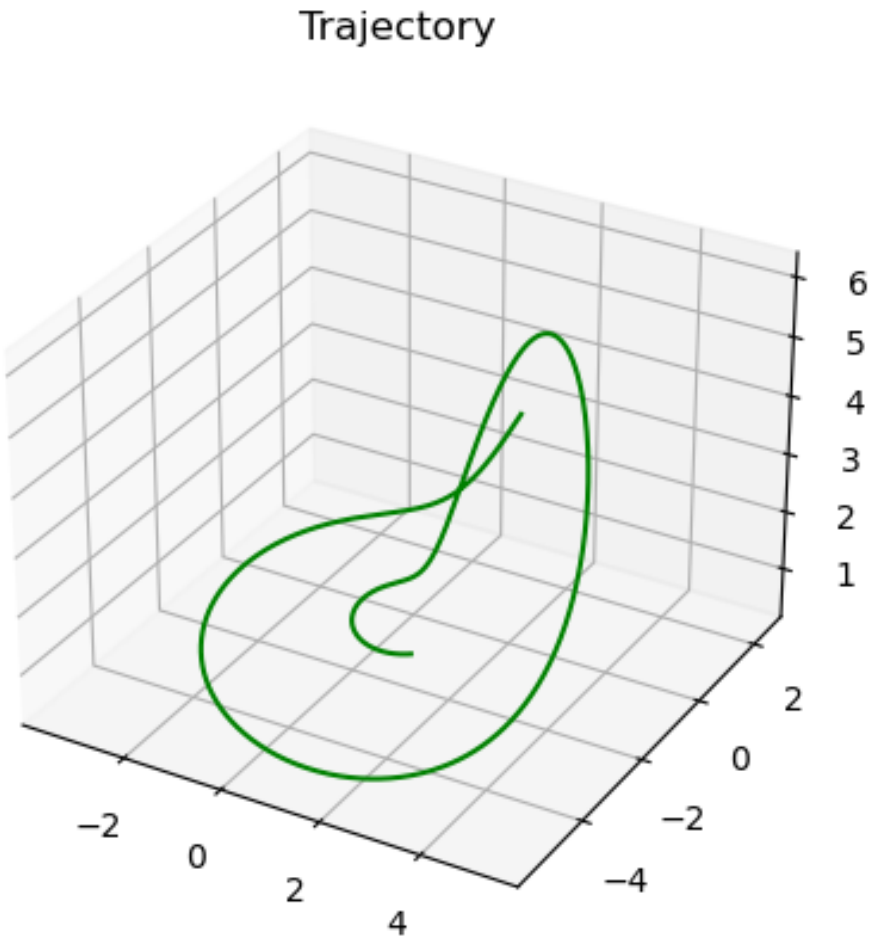
EXAMPLES: Rayleigh

- Integrated with **Runge-Kutta order 4 method**
- Initial Conditions: (1,2)
- F_0 parameter: 0.75



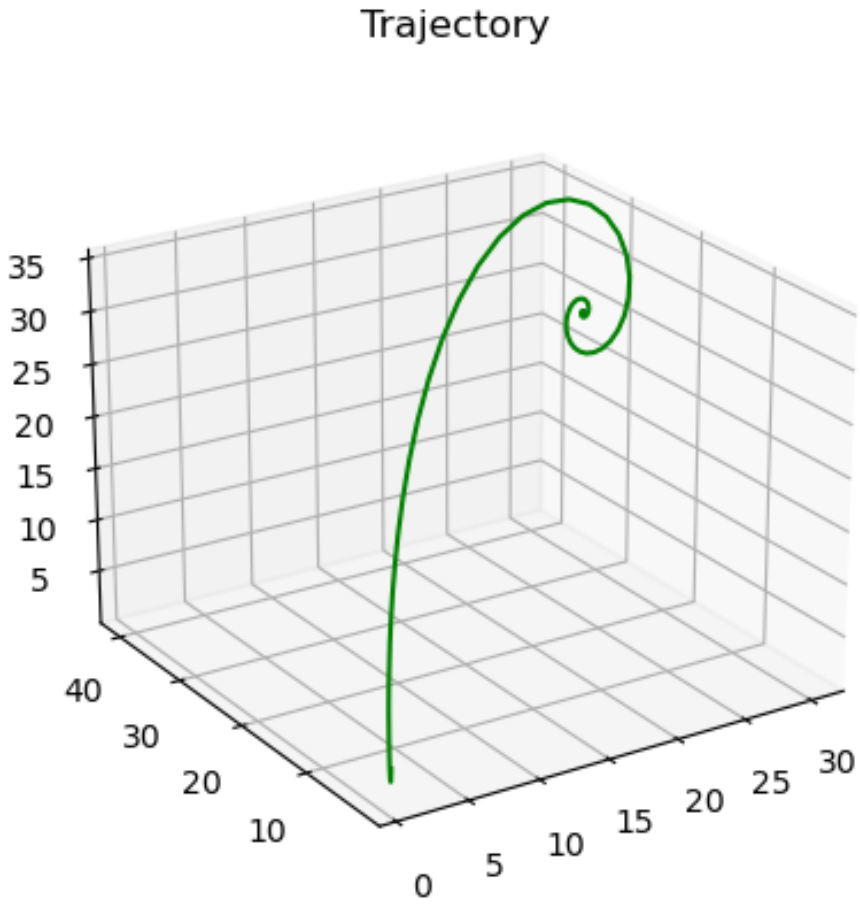
EXAMPLES: Rossler

- Integrated with **Crank-Nicolson method**
- Initial Conditions: (1,2,3)
- Parameters:
  - A = 0.398
  - B = 2
  - C = 4



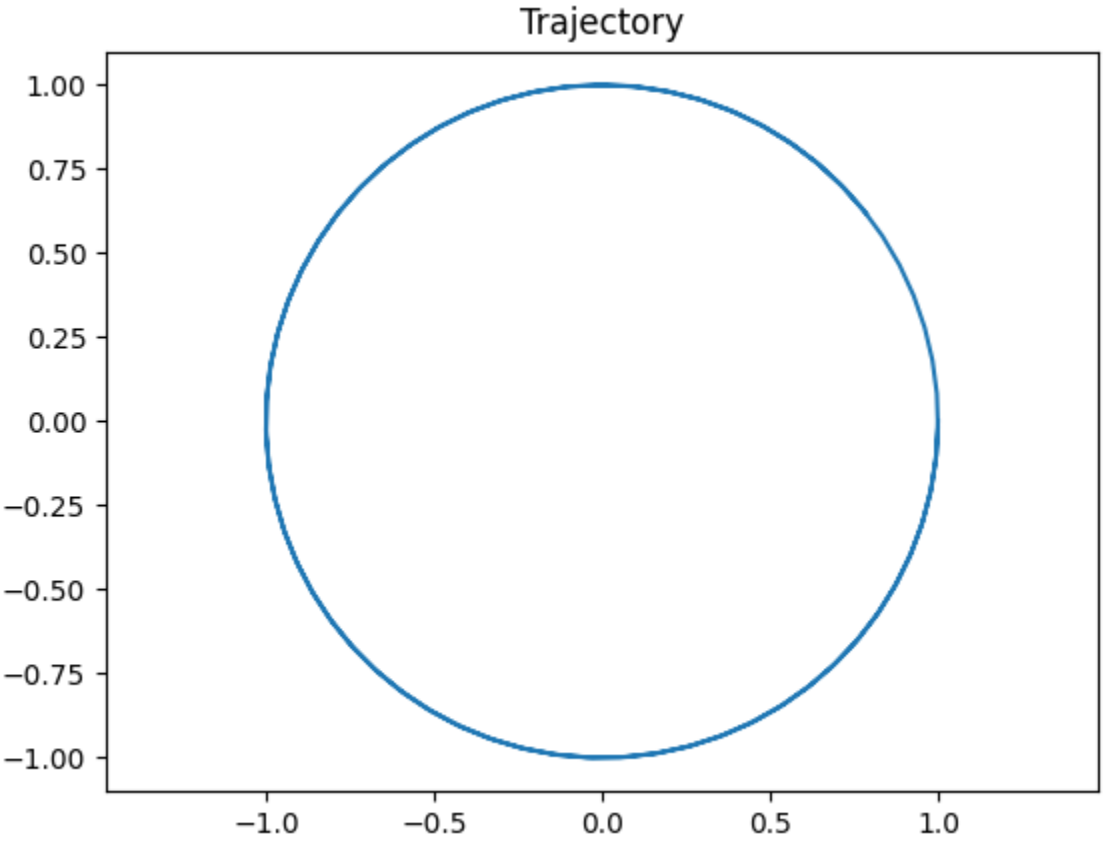
EXAMPLES: Lorentz

- Integrated with **Crank-Nicolson method**
- Initial Conditions: (1,2)
- Parameters:
  - R = 28
  - B = 8/3
  - C = 10



EXAMPLES: Harmonic Oscillator

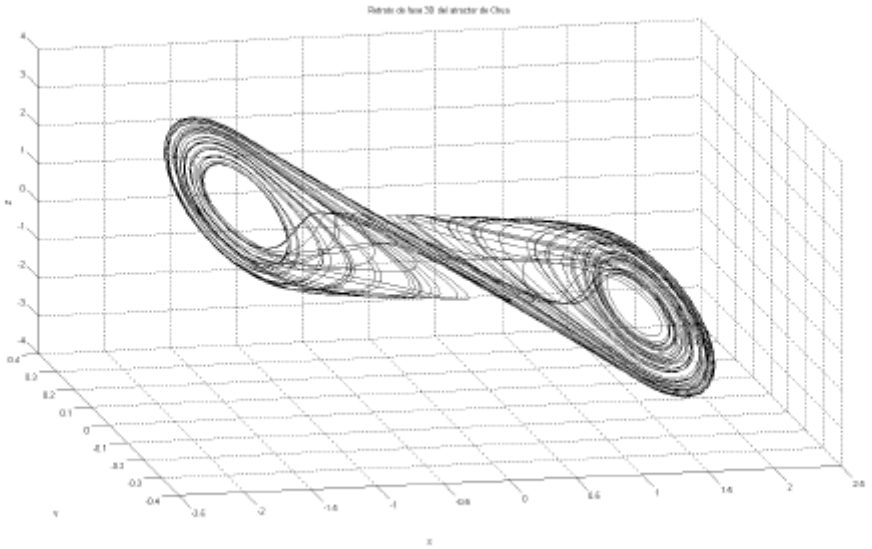
- Integrated with **Runge-Kutta order 4 method**
- Initial Conditions: (1,0)
- Parameters: n/a



Equation:

$$\ddot{x} + x = 0$$

- The results match the concepts studied in the course's classes.
- Future work:
  - Modify the MIDI module (more than 1 instrument in the same MIDI file)
  - Implement AI to add drum and bass (https://magenta.tensorflow.org/drumbot)
  - Add other dynamic systems to study their evolution (e.g.: Chua model)



ANY QUESTIONS?