Package draw_circle_fourier

Support visuel pour notre présentation

KOAN, CORDOVAL, BURGAT, GUILLAUMONT

Université de Montpellier

April 25, 2021

BUT

Le but de ce projet est de créer un module Python capable de dessiner une animation de n'importe quelle image, en utilisant les séries de Fourier. Ceci est inspiré de la vidéo 'Mais qu'est ce qu'une série de Fourier' par 3Blue1Brown sur youtube.
Nous allons en particulier travailler sur les degrés d'approximation de la reproduction de limage, ainsi que la vitesse des vidéos . Des tests et des analyses sur diverses images sont aussi étudiés. Dans un premier temps, nous allons faire des exemples pour tracer des images faciles à dessiner, puis de plus en plus complexes.
DOC

Répartition du travail

[Kenjy] Traitement des images et conversion en points :

- Extraction des coordonnées à partir d'une image
- Coordonnées X et Y -> Nombres Complexes

[Chloë] Trouvez les coefficients de Fourier pour localiser approximativement des points donnés:

- Fonction pour générer x + iy à un instant t donné
- Fonction pour évaluer y au temps t en utilisant l'approximation de Fourier du degré N
- Fonction pour calculer les coefficients
- Partie approximation

[Pierre + Paul] Créer une animation à partir des coefficients calculés:

- Fonction qui modifie les coefficients afin de tracer des cercles dans la dernière fonction.
- Fonction qui affiche le rayon de chaque cercle ainsi que le cercle

Equation complexe générique

Si :
$Z(t) = a_0 + a_1 e^{it} + a_2 e^{2it} + \cdots + a_N e^{Nit} + a_{-1} e^{-it} + a_{-2} e^{-2it} + \cdots + a_{-N} e^{-Nit}$
avec

- n : vitesses des cercles
- a : rayons et phases

Alors : $\forall n \in \mathbb{Z}, a_n = \frac{1}{\tau} \int_0^\tau Z(t) e^{-int}\, dt$

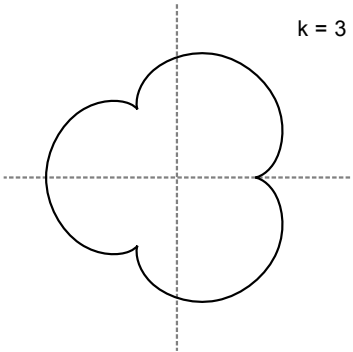


Figure 1: Epicycloïde

Manipulation de l'image

Support visuel pour notre présentation

Kenjy Koan

Université de Montpellier

April 25, 2021

Idée générale

- Dessin linéaire en 2D $\rightarrow$ fonction paramétrique $f(t) = (\, x(t), y(t) \,)$
- Chemin paramétrique passant par les pixels $\rightarrow$ composantes x et y
- Approximation par la transformée de Fourier
- Reconstitution fidèle de l'image

Manipulation de l'image

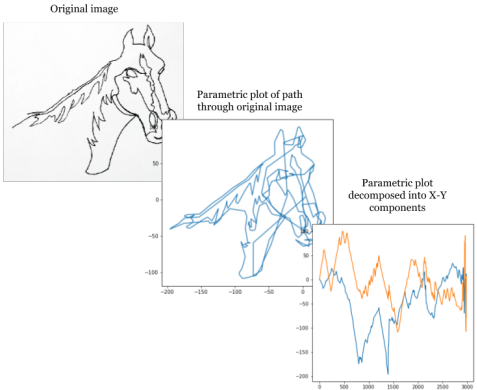


Figure 2: Illustration de la méthode

class **Package.draw_circle_fourier.ImageReader.Image.** **ImageReader** (*url*)

Read the url that contains an image in JPG, JPEG or PNG format then converting into coordinates.

get_tour (*level=[200]*)

Find the contour of image Split the points and store in x_table and y_table Centering the points to origin: (0,0) Make a time table from 0 to 2PI

Parameters

- x_table (*float*) – coordinate of the X axis
- y_table (*float*) – coordinate on the Y axis
- time_table (*float*) – time list from 0 to 2PI

Figure 3: Classe ImageReader

Coefficients de Fourier

Support visuel pour notre présentation

Chloë Cordoval

Université de Montpellier

April 25, 2021

Petite introduction

Dans cette partie, nous trouverons les coefficients de Fourier pour localiser approximativement des points donnés.
Pour cela, nous avons créé une classe de fonctions nommée **Fourier** pour notre Package.

Séries de Fourier

Pour une période L, les séries de Fourier complexes sont :

$$f(t) = \sum_{i=-\infty}^{\infty} C_n e^{int\frac{2\pi}{L}}$$

.

Séries de Fourier

On retrouve cette fonction à partir **des coefficients de Fourier** :

$$Cn = \frac{1}{L} \int_{\pi}^{-\pi} e^{-int\frac{2\pi}{L}} f(t)dt$$

```python
import matplotlib.pyplot as plt
import numpy as np
from math import tau
from scipy.integrate import quad
from numpy import interp

def f(t, time_table, x_table, y_table):
        X = np.interp(t, time_table, x_table)
        Y = 1j*np.interp(t, time_table, y_table)
        return X + Y

class Fourier:
    def __init__(self, time_table, x_table, y_table, order) :
        self.order = order
        self.time_table = time_table
        self.x_table = x_table
        self.y_table = y_table

    def coef_list(self,time_table, x_table, y_table, order) :

        coef_list = []
        for n in range(-order, order+1):
            real_coef = quad(lambda t: np.real(f(t, time_table, x_table, y_table) * np.exp(-n*1j*t)), 0, tau, limit=100, full_output=1)[0]/tau
            imag_coef = quad(lambda t: np.imag(f(t, time_table, x_table, y_table) * np.exp(-n*1j*t)), 0, tau, limit=100, full_output=1)[0]/tau
            coef_list.append([real_coef, imag_coef])
            return np.array(coef_list)

    def DFT(self, t, coef_list, order):

        kernel = np.array([np.exp(-n*1j*t) for n in range(-order, order+1)])
        series = np.sum( (coef_list[:,0]+1j*coef_list[:,1]) * kernel[:])

        return np.real(series), np.imag(series)
```

Figure 4: Classe Fourier

Description de la classe

Dans cette classe, nous retrouvons deux fonctions qui sont :

- **Coeff_List**
- **DFT**

La fonction **Coeff_List** prend comme paramètres d'entrée, les valeurs **time_table**, **x_table**, **y_table**, **order**.

Ces valeurs sont les paramètres de sortie de la fonction **get_tour** de la classe **ImageReader** décrite précédemment.

Cette fonction a pour but de calculer les coefficients de Fourier complexes et les décomposer en une partie réelle et imaginaire.

C'est pour cela que nous avons introduite hors du package une fonction **f** qui va convertir les coordonnées trouvées précédemment en nombres complexes par rapport à un temps t donné.

Nous allons le retrouver dans notre boucle "for" dans la fonction **Coef_List** car c'est elle qui va être décomposée en partie réelle et imaginaire.

Notre fonction retournera les valeurs de **coef_list** sous forme d'un array qui a été initialisé avant notre itération comme une liste vide.

Nous aurons donc les coefficients de Fourier complexes voulus sous forme d'un tableau (array) de valeurs élémentaires.

Séries de Fourier

Soit une fonction f de $\mathrm{L}^1(\mathbb{R})$. On appelle **transformée de Fourier** de f, notée F(t), la fonction :

$$F(t) = \frac{1}{\sqrt{2\pi}}. \int_{-\infty}^{\infty} e^{-ixt} f(x) dx$$

La fonction **DFT** pour "Discrete Fourier Transform", prend comme paramètres d'entrée, les valeurs **coef_list**, **t**, **order**.

coef_list est le paramètre de sortie de la fonction **Coef_List** définie dans notre classe.

Cette fonction va simplement calculer les transformées de Fourier de nos coefficients trouvés précédemment.

Elle va retourner comme paramètres de sortie, la partie réelle et la partie imaginaire de la transformée.

Visualisation des cercles de rotation de Fourier

Support visuel pour notre présentation

Burgat Paul et Guillaumont Pierre

Université de Montpellier

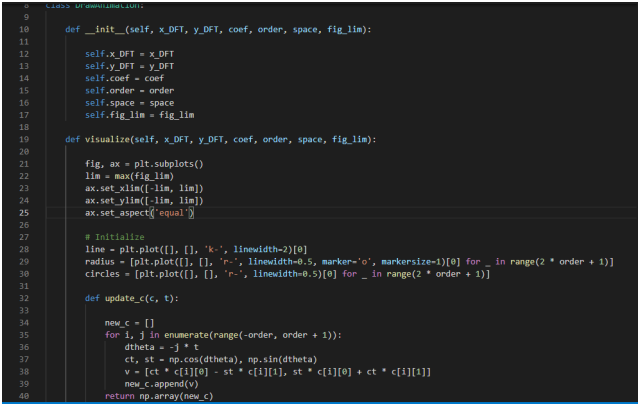
April 25, 2021

Introduction

Dans cette partie, à partir des coefficients de Fourier trouvés pour localiser approximativement des points donnés, nous allons faire une animation qui va tracer une courbe représentant une image choisie. Pour se faire, nous avons créé une classe de fonctions **DrawAnimation**.

La classe **DrawAnimation** est composée de 4 fonctions :

- **visualize**
- **update_c**
- **sort_velocity**
- **animate**



```
class DrawAnimation:

    def __init__(self, x_DFT, y_DFT, coef, order, space, fig_lim):

        self.x_DFT = x_DFT
        self.y_DFT = y_DFT
        self.coef = coef
        self.order = order
        self.space = space
        self.fig_lim = fig_lim

    def visualize(self, x_DFT, y_DFT, coef, order, space, fig_lim):

        fig, ax = plt.subplots()
        lim = max(fig_lim)
        ax.set_xlim([-lim, lim])
        ax.set_ylim([-lim, lim])
        ax.set_aspect('equal')

        # Initialize
        line = plt.plot([], [], 'k-', linewidth=2)[0]
        radius = [plt.plot([], [], 'r-', linewidth=0.5, marker='o', markersize=1)[0] for _ in range(2 * order + 1)]
        circles = [plt.plot([], [], 'r-', linewidth=0.5)[0] for _ in range(2 * order + 1)]

        def update_c(c, t):

            new_c = []
            for i, j in enumerate(range(-order, order + 1)):
                dtheta = -j * t
                ct, st = np.cos(dtheta), np.sin(dtheta)
                v = [ct * c[i][0] - st * c[i][1], st * c[i][0] + ct * c[i][1]]
                new_c.append(v)
            return np.array(new_c)
```

Figure 5: Class_DrawAnimation

Fonction visualize

La fonction **visualize** prend comme variables x_DFT et y_DFT qui proviennent de la fonction **DFT** de la classe **Fourier**, ensuite elle prend comme paramètres **coef** et **order**.
coef représente les coefficients de Fourier et **order** nous donnera le nombre de coefficients que l'on souhaite, si **order**=50, nous aurons 100 coefficients de Fourier. La variable **space** représente une liste de données allant de 0 à **tau**, **tau** est une constante de cercle égale à 2π, le rapport de la circonférence d un cercle à son rayon, et pour finir **fig_lim** représente les limites des coordonnées x et y de notre figure.
Cette fonction a pour but de faire une visualisation qui représente une image choisie à l'aide des cercles de rotation de Fourier.

Pour réaliser cela, nous définissons la fonction **update_c** qui prend comme variables **c** et **t**. La variable **c** est un ndarray qui nous le verrons plus tard correspond au paramètre **coef** de la fonction **visualize** et la variable **t** est un entier que nous définierons plus tard. Ensuite on itinialise un nouveau ndarray **new_c** dans cette fonction, le but de cette fonction est d'arranger les parties réelles et imaginaires des coefficients de Fourier afin de pouvoir afficher le rayon de chaque cercle ainsi que le cercle dans la fonction **animate**.

Manipulation de l'image

```
    def sort_velocity(order):

        idx = []
        for i in range(1,order+1):
            idx.extend([order+i, order-i])
        return idx

    def animate(i):
        # animate lines

        line.set_data(x_DFT[:i], y_DFT[:i])
        # array of radius of each circles
        r = [np.linalg.norm(coef[j]) for j in range(len(coef))]
        # position is on the last circle
        pos = coef[order]
        c = update_c(coef, i / len(space) * tau)
        idx = sort_velocity(order)
        for j, rad, circle in zip(idx,radius,circles):
            new_pos = pos + c[j]
            # plot radius in each circles
            rad.set_data([pos[0], new_pos[0]], [pos[1], new_pos[1]])
            theta = np.linspace(0, tau, 50)
            # plot each circles
            x, y = r[j] * np.cos(theta) + pos[0], r[j] * np.sin(theta) + pos[1]
            circle.set_data(x, y)
            # increase pos for plot from the last circle displayed
            pos = new_pos

    # Animation
    ani = animation.FuncAnimation(fig, animate, frames=len(space), interval=5)
    return ani
```

Fonction sort_velocity

La fonction **sort_velocity** prend en entrée la variable **order** définie dans la fonction **visualize**. Elle permet de créer une variable **idx** qui est utilisée dans la fonction **animate**. On initialise **idx** comme une liste vide puis on la remplit grâce à une boucle for pour i=1 de **order**+1 à **order**-1 jusqu'à pour i=**order**+1 de $2 \times$ **order** à 0.

Fonction animate

La fonction **animate** prend en entrée un entier *i*. Elle permet d'afficher le rayon de chaque cercle ainsi que le cercle grâce à une double boucle **for** imbriqué. A chaque itération, on affiche un cercle supplémentaire et son rayon.
Duquel on affichera le prochain cercle via les variables **pos** et **new_pos** Cette fonction trace aussi l'approximation de l'image en utilisant la commande **line.set_data**.

Pour avoir un rendu animé, on utilise la fonction **FuncAnimation** du package **matplotlib.animate**.

Exemples

Support visuel pour notre présentation

Koan Kenjy, Cordoval Chloë, Burgat Paul, Guillaumont Pierre

Université de Montpellier

April 25, 2021

EXEMPLES

Coeur-1 Coeur-2

Star-1 Star-2

Horse-1 Horse-2

Velo-1 Velo-2

Fish-1 Fish-2

Bart-1 Bart-2

Papillon-1 Papillon-2

Akatsuki-1 Akatsuki-2

Apple-1 Apple-2

Rolex-1 Rolex-2

Puma-1 Puma-2