The Kuramoto Model

Technical report for python numerical project

Paradis Enzo
Student at the university of Bourgogne Franche-Comté
Master CompuPhys - 1st year

Contents

The Kuramoto Model (Yoshki Kuramoto) is a mathematical model that describes the syncronization of coupled oscillators. This program, written by myself, Paradis Enzo, is based on this model to describe a set of 2D-coupled oscillators in various cases. In particular to reproduce data obtain in the case of the coupled arrays of Josephson junctions [1]. This program was edited for the last time the 10/23/2020. It was written in the language python 3 under ubuntu, so you just have to use `python3` to execute it. There are 6 files. The first one is the main file, `main.py`, it is the only one you have to execute, you could have to comment or uncomment some lines. Instead of that you just have to edit one file, the settings file, `settings.py` were you can find all the settings. Then you have the kuramoto file, `kuramoto.py`, where you can find the `class KuramotoModel` which contains all the functions related with the model or the verifications. The data file, `data.py`, contains the `class Data`, which is used to manage data. The graphs file, `graphs.py`, with the `class Graphs` manages the displaying of the results. And the integrator file, `integrator.py`, where you can find the `class Integration`, which contains the three integrators : Euler, RK2, RK4. There is also the parameters directory which is important to save data.You can find all this files on github.

I Functional requirement of the program

All the functions which are used for the computing or the displaying of the results are called in the main file. At first there are the functions wich create the initial values in respect of the parameters set in the settings file, then this values are stocked in data (.dat) files in a directory named "parameters". With these data files, we don't have to repeat the calculations every time we want to test something. Then you have the functions that compute the results (the phase of the oscillators, the complex mean average, and the Shannon entropy), wich are stocked in data files in the same directory. And finally there are the functions that display the graphs by using the module `matplotlib.pyplot`. In this section we will describe the functions that create data files and their datas. Firstly the initial datas are created throught the `class Data` wich is in the data file. Finally the computing of the other values is in the `class KuramotoModel` wich is in the kuramoto file.

I.1 The initialisation of data

The initial datas are created by the function: `data.init_data(state)`. You can choose if you want to create new initial datas by changing the boolean value of the `newData=False` variable in the settings file.

`data.init_data(state="random")`		
Description	Input	Output
This function is using to create initial values, stocked in data files in the directory parameters, according to the value of the argument `state` set by default to `state="random"`. You can create data for random, chimera, inverse, or josephson states.	The argument `state` is a `string`. By default it takes the value `"random"` but you can give it these values: • `"random"` • `"chimera"` • `"inverse"` • `"josephson"`	This function will retrieve six data file in the parameters directory, computing in according to the `state` argument. The data files are: • `"omega.dat"` • `"theta0.dat"` • `"K.dat"` • `"eta.dat"` • `"alpha.dat"` • `"tau.dat"`

Table 1: function data.init_data()

Each `state` create six variables, wich are defined as follows:

- ω is a list of real of size N. It contains the natural oscillations of each oscillators.
- θ_0 is a list of real of size N. It contains the intial phase of each oscillators. So at the $\theta^i(t = t_0)$.

- K is a matrix of real of size N×N. It contains the coupling coefficients between each oscillators depending on the nearest neighbour. The nearest neighbour of an oscillator are defined by M in the settings file. It is set as 30% of the totality of the oscillators, you can change it.
- α is a matrix of real of size N×N. It is the dephasing matrix of the coupling.
- τ is a matrix of real of size N×N. It is the delay matrix.
- η is a matrix of real of size N×T. It is the representations of external noises for each oscillators at each time $t \in]0, T[$.

They are different for each state, you can see their definitions in the tables 2 and 3. The function `uniform()` is from the module `random`, and provide random real numbers with uniform distribution, in the range given. The function `randint()` do the same things but for integers.

`"random"`	`"chimera"`
This `state` represent the case with random values defined by: • $\omega^i = uniform(0, 3)$ • $\theta_0^i = uniform(0, \frac{2}{\pi})$ • $K_j^i = uniform(0, 1e10)$ • $\eta_j^i = uniform(0, 0.5)$ • $\alpha_j^i = uniform(0, \frac{2}{\pi})$ • $\tau_j^i = randint(0, N//2)$	This `state` represent the case of a quantum chimera state[2] defined by: • $\omega^i = 0.2 + i * 0.4 * sin(\frac{i^2 * \pi}{(2 * N^2)})$ • $\theta_0^i = uniform(0, \frac{2}{\pi})$ • $K_j^i = uniform(0, 1e10)$ • $\eta_j^i = uniform(0, 0.5)$ • $\alpha_j^i = 1.46$ • $\tau_j^i = randint(0, N//2)$

Table 2: `states`

`"inverse"`	`"josephson"`
This `state` represent the case with the coupling matrix defined by the inverse of the distance between two oscillators, and the delay matrix is proportionnal to this distance. • $\omega^i = uniform(0, 3)$ • $\theta_0^i = uniform(0, \frac{2}{\pi})$ • $K_j^i = \begin{cases} \frac{1}{\lvert i-j \rvert} \; if \; \lvert i-j \rvert \neq 0 \\ 1e20 \; otherwise \end{cases}$ • $\eta_j^i = uniform(0, 0.5)$ • $\alpha_j^i = 1.46$ • $\tau_j^i = \lvert i-j \rvert$	This `state` represent the case of the array of Josephson, data were drawn from this article [1]. Their datas are defined by: • $\omega^i = 0.2 + 0.4i \times sin(\frac{i^2 \pi}{(2N^2)})$ • $\theta_0^i = uniform(0, \frac{2}{\pi})$ • $K_j^i = \frac{Nr\omega^{i^2} 2\,\mathrm{e} / (\hbar r I_b) - \omega^i}{\sqrt{(L\omega^{i^2} - 1/C)^2 + \omega^{i^2}(R + rN)^2}}$ • $\eta_j^i = uniform(0, 0.5)$ • $cos(\alpha_j^i) = \frac{L\omega^{i^2} - 1/C}{\sqrt{(L\omega^{i^2} - 1/C)^2 + \omega^{i^2}(R + rN)^2}}$ • $\tau_j^i = randint(0, N//2)$

Table 3: `states`

For each `state` choosen you have to define in the settings file the parameters `Nr` and `Nc`, to define the geometry of the system. `Nr` define the number of rows and `Nc` the number of columms, so `N=Nr*Nc` is the number of oscillators that you have. For example if you choose `N=12` in the geometry `Nr=3, Nc=4`, you will have this configuration:

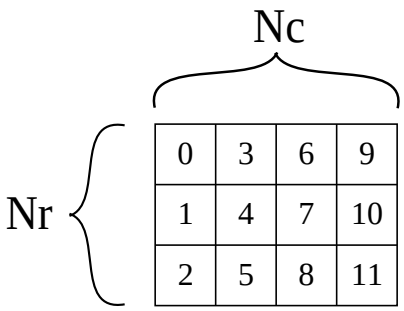

Figure 1: Configuration of the oscillators for `Nr=3, Nc=4`

The labels in the Figure are the labels of the oscillators. If you put a non-positive number you will have some issues, the program will not work. So be careful to respect the physics of the system.

I.2 The computing functions

There are three computing functions that are imported from the `class KuramotoModel`, by the object `kuramoto`. You can choose if you want to compute new datas by changing the boolean value of the `newComputing=False` variable in the settings file.

`kuramoto.integrate(f, theta0, tf=100, integrator="RK4")`		
Description	Input	Output
This function compute integrates the function `f`, with the initial vector `theta0` during a time `tf`, set by default to `tf=100` with 1000 values(T), using the integrator, set by default to `integrator="RK4"`	• `f` is the function to be integrated, such that $f : \mathbb{R}^n \to \mathbb{R}^n$ • `theta0` is the initial vector, such that $\theta_0 \in \mathbb{R}^n$. So it has to be a list of real of size N • `tf` is the duration of the integration. Set to 100 by default. It is an integer • `integrator` is the integrator that you want to choose. It can take only this string values: `"Euler"`, `"RK2"`, `"RK4"`. Among the three integrators, RK4 is the one losing the least energy the longer the integration lasts, therefore it is the default value.	This function will retrieve two data files in the parameters directory. • `"theta.dat"` : contains a matrix of real of size N×T. It is used like a list of vectors. Each vector contains the phase of each oscillators at one time, labeled in the same way as the Figure 1. So the matrix represent the evolution over time of this vector. • `"t.dat"` : contains a list of real of size T, defined by `linspace(0, tf, T)`. `linspace()` is a function added by `numpy`.

Table 4: function kuramoto.integrate()

`kuramoto.orders()`		
Description	Input	Output
This function computes the orders parameters R and Φ, of the θ^i over the time, by the relation : $Re^{i\Phi} = \frac{1}{N}\sum_{i=1}^{N} e^{i\theta^i}$, for one time.	There are no arguments to this function because it takes only data files. It takes the theta file and the t file, see Table 4.	This function will retrieve two data files in the parameters directory. • `"R.dat"` : It contains a list of real of size T. `R` represents the complex mean of magnitudes of the θ^i over the time. • `"phi.dat"` : It contains a list of real of size T. `phi` represents the complex mean of angles of the θ^i over the time.

Table 5: function kuramoto.orders()

`kuramoto.shannon_entropies()`		
Description	Input	Output
This function computes the Shannon entropy over the time of θ. So for each vector of the list of vectors `theta`	There are no arguments to this function because it takes only data files. It takes the theta file and the t file, see Table 4.	This function will retrieve two data files in the parameters directory. • `"S.dat"` : It contains a list of real of size T. `S` represents the Shannon entropy over the time of each vector of `theta`

Table 6: function kuramoto.shannon_entropies()

To summarize in the main file there are 4 functions that are called for the computing. The first one is for initializing data according to the state choosed by the user. The following three are here to compute the main datas by using the Kuramoto model. You can choose if you want to initialized new datas (1) or do a new computing (4, 5 & 6) by modifying the logical variables (`newData`, `newComputing`) in the settings file. In this file you can modify the `state` variable (2 & 3). You can also modify the geometry of the system by changing `Nr` and `Nc`, don't forget to redo the initial datas and the computing after that. Be careful not to set the number of rows or columns to a negative or zero number, it wouldn't make any sense. If you want to save or use other files than the predifined one you can modify it in the dictionnary `FILE`. You don't have to modify anything else that is not in the settings file. If you do it be careful, because you can break the program. For the rest of the main file there are 12 functions that are here to display the results. The last three are here to create gifs, they are the begining of the gold version. These displaying functions will be explained later.

II Internal structure of the program

II.1 Description of the physical model

In this program we had to describe 2D-coupled oscillators and tried to reproduce the results obtain in the article *Frequency locking in Josephson arrays: Connection with the Kuramoto model* [1]. So we will use the Kuramoto Model. Proposed by Yoshiki kuramoto this model is used to describe the syncronization of a large set of coupled oscillators. It is a mathematical model and the form that we have used is :

$$\dot{\theta}^i(t) = \omega_i + \frac{1}{N}\sum_{j=0}^{N-1} K_{ij}\, sin(\theta^j(t-\tau_{ij}) - \theta^i(t) + \alpha_{ij}) + \eta_i(t)$$

where :

- $\dot{\theta^i}(t)$ represent the varation of the phase of the i-th oscillator over the time.
- ω_i represent the natural frequency of the i-th oscillator.
- N is the total number of oscillators in the system.
- K_{ij} is the coupling matrix. It represent the coupling between each oscillators depending on the inverse of the physical distance between them. Sometimes you can consider only the nearest neighbour.
- θ^i is the phase of the i-th oscillator.
- τ_{ij} is the delay matrix. It is to take into account the phase propagation of the oscillators which have an influence on the i-th oscillator.
- α_{ij} is the dephasing matrix. It represent the dephasing due of the coupled oscillators.
- $\eta_i(t)$ represent the influence of the environement, the possible external noises.

As you can see this model is pretty complete. The formula is the sum of all the influences due to the environment and the coupled oscillators occuring on each oscillators at a given given time. That why it was chosen to build θ like a list of vectors.

II.2 Desciption of the used scientific computation algorithms

During the computation we have used 3 algorithm for the integration of the kuramoto model II.1, the Euler, RK2 and RK4 algorithm. But you can choose the one that you want to use. It is recommended to use the RK4 method because it loses less energy over time than the others, so it is more precise but if you want less computing time it is a good thing to try the other. You can see all these integrators in the `class Integration` in the integrator file.
We are looking to integrate $\dot{\theta} = f(\theta)$.
Let $\theta \in \mathbb{R}^n$ and $f : \mathbb{R}^n \to \mathbb{R}^n$. We wil note θ^i the i-th component of θ and θ_n the vector θ at the time t_n, $\theta_n \equiv \theta(t_n)$. So we have θ_n^i the i-th component at a time t_n of θ.
The Euler and Runge-Kutta methods are integrators of first-order differential equations and are based on the definition of the derivative:

$$\begin{aligned} \dot{\theta}(t_n) &= \frac{\theta(t_n + h) - \theta(t_n)}{h} \simeq \frac{\theta_{n+1} - \theta n}{\Delta t} \\ \iff \theta_{n+1} &= \theta_n + \dot{\theta}(t_n)\Delta t = \theta_n + f(\theta_n)\Delta t \end{aligned} \tag{II.1}$$

with $\Delta t = \frac{T}{N_t}$ the step of the simulation ($N_t \in \mathbb{N}$ the number of steps of the simulation and T the time of the simulation). We need Δt sufficiently small so N_t enought large and T sufficiently thin.

II.2.1 Euler's method

The Euler algorithm is directly an application of the Equation II.1:

$$\theta_{n+1} = \theta_n + f(\theta_n)\Delta t \tag{II.2}$$

This method will diverge quicker than the two other but the computing time is the shortest.

II.2.2 RK2 method

The Runge-Kutta algorithm (RK2) is like the Euler's method but with a double use of the Equation II.1. There is a middle step to refine the estimation of f.

$$\theta_{n+1} = \theta_n + f(\theta_n + f(\theta_n)\frac{\Delta t}{2})\Delta t \tag{II.3}$$

This method will diverge quicker than the RK4 method and slower than the Euler's method. Its computing time will be longer than the one of the Euler's method but remains shorter than the one of RK4 method.

II.2.3 RK4 method

RK4, the fourth order Runge-kutta algorithm, is similar to the RK2 algorithm but with a refinement around the middle point.

$$\theta_{n+1} = \theta_n + (K_{1,n} + 2K_{2,n} + 2K_{3,n} + K_{4,n})\frac{\Delta t}{6} \tag{II.4}$$

with

$$\begin{aligned} K_{1,n} &= f(\theta_n) \\ K_{2,n} &= f\left(\theta_n + K_{1,n}\frac{\Delta t}{2}\right) \\ K_{3,n} &= f\left(\theta_n + K_{2,n}\frac{\Delta t}{2}\right) \\ K_{4,n} &= f\left(\theta_n + K_{3,n}\Delta t\right) \end{aligned} \tag{II.5}$$

Even if this is the slowest method of three orders of the Runge-Kutta algorithm, it is the most accurate one. You can find all this algorithm in the integrator file in the `class Integration`.

II.3 List of the constitutive elements of the program

The Program includes 4 classes. There are 3 that are used in the main file and 1 in the `class KuramotoModel`, see Figure 3.

II.3.1 The `class Integration`

This class doesn't have any attributes. It is just used to store the integrators. It can be used in an other program just by writting `import integrator` at the begining of you code, and create an object with the class. This class is using the module `numpy`. Here is a table of the class:

`class Integration`	
Attributes	Methods
∅	This class contains 3 methods, see subsection II.2: • `euler(f, x0,t)` • `RK2(f, x0,t)` • `RK4(f, x0,t)`

Table 7: class Integration

II.3.2 The `class Data`

This class is used to create the initial data. Here is a table of the class:

`class Data`	
Attributes	Methods
This class contains 8 attributes, see subsection I.1: • `M,type:int` • `N,type:real` • `K,type:ndarray,shape:(N,N)` • `alpha,type:ndarray,shape:(N,N)` • `eta,type:ndarray,shape:(N,1000)` • `omega,type:ndarray,shape:(N,N)` • `tau,type:ndarray,shape:(N,N)` • `theta0,type:ndarray,shape:(N,N)`	This class contains 5 methods: • `init_data(state)`, see Table 1 • `chimera_states()`, see Table 2 • `inverse_states()`, see Table 3 • `random_states()`, see Table 2 • `josephson_matrice()`, see Table 3

Table 8: class Data

You can add a new method in this class to add new configurations. But don't forget to add a new possible value to the `state` variable in the `init_data()` function. As follows :

```
if state == "random":
    self.random_states()
elif state == "chimera":
    self.chimera_states()
elif state == "inverse":
    self.inverse_states()
elif state == "josephson":
    self.josephson_matrice()
elif state == "new method":
    self.new_method()
```

II.3.3 The `class Graphs`

This class has no attributes. It is used to store all the displaying functions and the function to create gifs. Here is a table of the class:

`class Graphs`	
Attributes	Methods
∅	This class contains 13 methods: • `graphs(x, y, title, xlabel, ylabel, pol)` • `kuramoto(theta, pol, integrator)` • `anim_kuramoto(theta)` • `dens_kuramoto(N)` • `dens_kuramoto_coord(t, show)` • `anim_dens_kuramoto_coord(t)` • `orders()` • `shannon(pol, t)` • `dens_shannon(N)` • `dens_shannon_coord(t, show)` • `anim_dens_shannon_coord(t)` • `connectivity(K, kmin)` • `graph_connectivity(kmin)`

Table 9: class Graphs

II.4 Functional diagram of the program

The program is cut out in 6 modules: main, graphs, kuramoto, data, integrator, and settings. You can see how they interact with each other in the following figure: In these modules data, integrator graphs and kuramoto

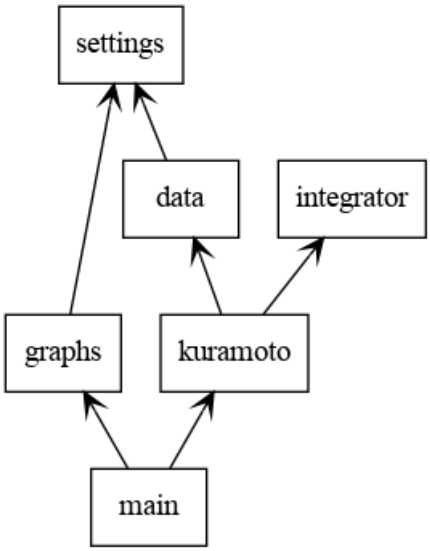

Figure 2: Diagram of the interactions between the modules.

contain a class. These classes are depending to each other like in the following diagram:

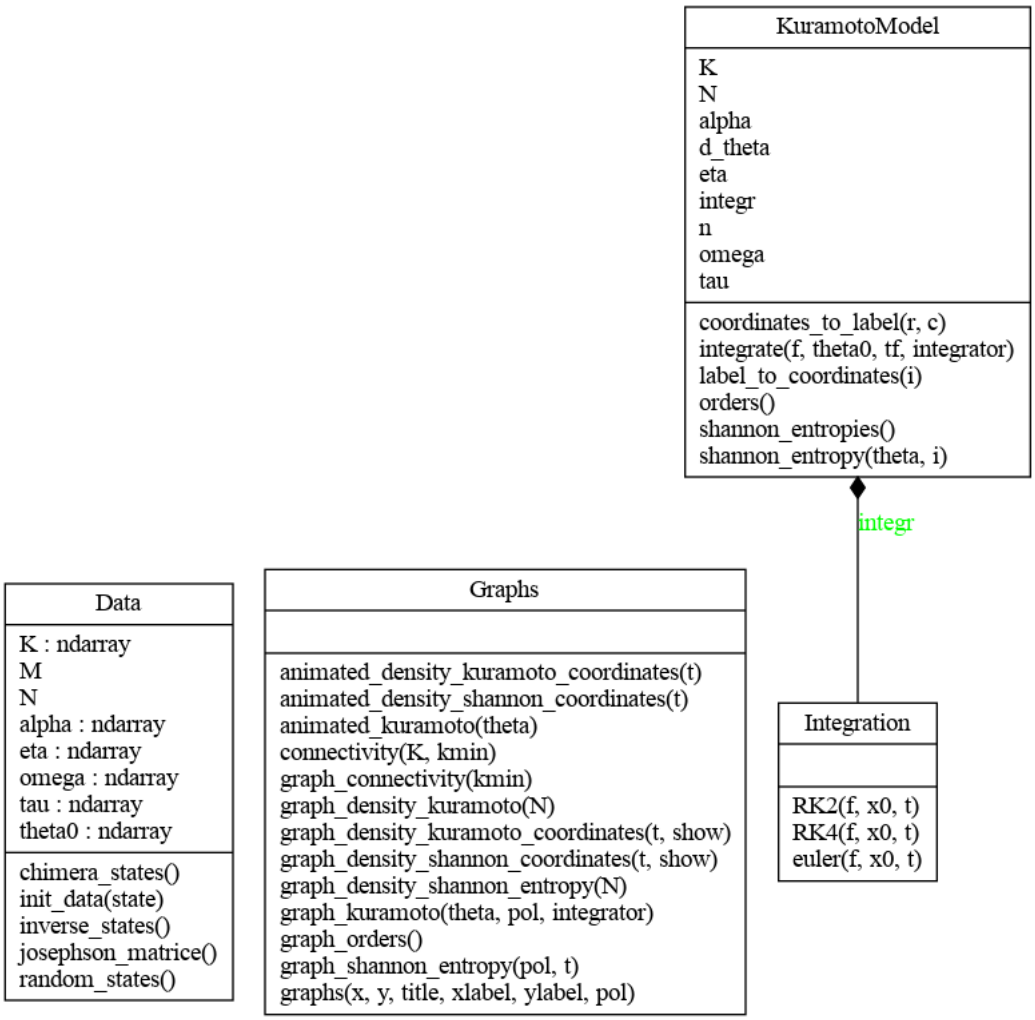

Figure 3: Diagram of the dependences of the classes.

References

[1] Steven H. Strogatz Kurt Wiesenfeld, Pere Colet. Frequency locking in josephson arrays: Connection with the kuramoto model. 57(2), February 1998.

[2] David Viennot. Chaos et chimères quantiques. http://perso.utinam.cnrs.fr/ viennot/publi/chaos.pdf. En ligne; dernière visite : 31 octobre 2020.