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 the datas 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 the datas. 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. 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 the datas 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 the datas

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, the datas 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:

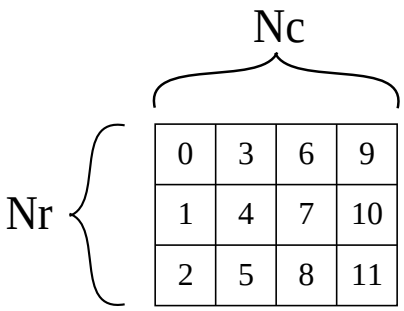

Figure 1: Configuration of the oscillators for `Nr=3, Nc=4`

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 the datas 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 would make no sense. If you want to save or use other files than the predifined one you can modify it in the dictionnary `FILE`. You doesn't have to modify anything else that is not in the settings file. If you do it be careful, because you can broke the program. For the rest of the main file there are twelve 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 function will be explain later.

References

[1] Pere Colet Kurt Wiesenfeld and Steven H. Strogatz. 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.