# A Study of Kuramoto Oscillators

Aaron Schade

January 27, 2021

## Contents

# 1   Theoretical study

## Lyapunov and stability

### 1.1   Mean natural frequency

Show that

$$\frac{d}{dt}\left(\frac{1}{N}\sum_{i=1}^{N}\omega\right) = \bar{\omega} \tag{1}$$

where

$$\bar{\omega} := \frac{1}{N}\sum_{i=1}^{N}\omega_i \tag{2}$$

---

We already showed this in the lecture for the case with the lowest possible number of oscillators, $N = 2$ (ignoring the trivial case of $N = 1$). Therefore, to show that it holds generally, we have to go to the other extreme and check that it still holds for $N \to \infty$. This in turn means that we will be wanting to take a limit at some point. The simulation later on then fills the gap between $N = 2$ and $\infty$ by running the model for specific values of $N$.

Start by substituting in and expanding the term for $\theta_i$:

$$\frac{d}{dt}\left(\frac{1}{N}\sum_{i=1}^{N}(\theta_i)\right) = \frac{1}{N}\sum_{i=1}^{N}\left(\frac{d\theta_i}{dt}\right)$$

$$= \frac{1}{N}\sum_{i=1}^{N}\left[\omega_i + \frac{K}{N}\sum_{j=1}^{N}\sin(\theta_j(t) - \theta_i(t))\right]$$

$$= \frac{1}{N}\sum_{i=1}^{N}\omega_i + \frac{1}{N}\frac{K}{N}\sum_{i=1}^{N}\sum_{j=1}^{N}\sin(\theta_j(t) - \theta_i(t))$$

$$= \frac{1}{N}\sum_{i=1}^{N}\omega_i + \frac{1}{N}\frac{K}{N}\sum_{i=1}^{N}\left(\sin(\theta_1(t) - \theta_i(t)) + \sin(\theta_2(t) - \theta_i(t)) + \ldots + \sin(\theta_N(t) - \theta_i(t))\right)$$

After expanding the second sum, we have essentially a matrix of all the interaction terms. When $i \neq j$, we can make use of the fact that *sin* is an odd function. The two symmetrical cases will cancel out to zero. All off-diagonal elements in this matrix thus cancel out, only the diagonal elements with $i = j$ remain:

$$\frac{d}{dt}\left(\frac{1}{N}\sum_{i=1}^{N}(\theta_i)\right) = \frac{1}{N}\sum_{i=1}^{N}\omega_i + \frac{1}{N}\frac{K}{N}\left(\sin(\theta_1(t) - \theta_1(t)) + \sin(\theta_2(t) - \theta_2(t)) + \ldots + \sin(\theta_N(t) - \theta_N(t))\right)$$

$$= \frac{1}{N}\sum_{i=1}^{N}\omega_i + \frac{1}{N}\frac{K}{N}\left(0\right)$$

$$= \underbrace{\frac{1}{N}\sum_{i=1}^{N}\omega_i}_{\text{converging sequence}}$$

$$\lim_{N\to\infty}\frac{1}{N}\sum_{i=1}^{N}\omega_i \Rightarrow \bar{\omega}$$

Therefore

$$\frac{d}{dt}\left(\frac{1}{N}\sum_{i=1}^{N}(\theta_i)\right) = \bar{\omega}$$

This means that after the system synchronises the oscillators in its phase-locked component will on average move with the mean natural frequency $\bar{\omega}$. For me, this implication is interesting because of its simplicity.

Despite the system being potentially very large and the mutual updating patterns very complex, the final outcome is pre-destined by the 'natural' properties of the oscillators before the system even starts. They can interact in an almost uncountably-large number of ways, yet in the end all that complexity degenerates and collapses to a simple average. I suppose this is what is called self-organisation, and often in the real world we actually desire this degenerating behaviour of a system. It is reminiscent of the efficient market hypothesis in this sense: a large number of agents and products interact, creating a staggering number of potential behaviours and allocations. Yet in the end, one unique price emerges for each good, balancing all agents' demands and standardising their (buying and selling) behaviour. Like with the oscillators, these prices are implicit in the initial conditions (allocations) in this system, but need a complicated process to manifest themselves.

## 1.2 Lyapunov function $\mathcal{H}$

In the rotating reference frame with $\omega = \bar{\omega}$ and $\phi_i := \theta_i - \bar{\omega}$, the Kuramoto model becomes

$$\frac{d\phi_i}{dt} = (\omega_i - \bar{\omega}) + \frac{K}{N}\sum_{j=1}^{N} sin\left(\phi_j - \phi_i\right) \tag{3}$$

Show that

$$\mathcal{H} := -\frac{K}{2N}\sum_{i,j}\cos(\phi_i - \phi_j) - \sum_{i=1}^{N}(\omega_i - \bar{\omega})\phi_i \tag{4}$$

is a Lyapunov function, ie. that

$$\dot{\mathcal{H}} := \sum_{i=1}^{N}\underbrace{\frac{\partial\mathcal{H}}{\partial\phi_i}}_{\text{to be calculated}} \cdot \underbrace{\frac{d\phi_i}{dt}}_{\text{known from (3)}} \leq 0 \tag{5}$$

Start by expanding out the summations and taking the derivative of cos

$$\frac{\partial\mathcal{H}}{\partial\phi_i} = -\frac{K}{2N}\sum_{i}^{N}\sum_{j}^{N}\left(-sin(\phi_i - \phi_j)\right) - \sum_{i=1}^{N}(\omega_i - \bar{\omega})$$

$$= -\frac{K}{2N}\sum_{i}^{N}\sum_{j}^{N}\left(-sin(\phi_j - \phi_i)\right) - \sum_{i=1}^{N}(\omega_i - \bar{\omega})$$

$$= (-1)\left[\frac{K}{2N}\sum_{i}^{N}\sum_{j}^{N}sin(\phi_i - \phi_j) + \sum_{i=1}^{N}(\omega_i - \bar{\omega})\right]$$

$$= -\frac{d\phi_i}{dt}$$

We can substitute this into (5), such that

$$\dot{\mathcal{H}} = \sum_{i=1}^{N} -\frac{d\phi_i}{dt}\frac{d\phi_i}{dt}$$

$$= \sum_{i=1}^{N} -\underbrace{\left[\frac{d\phi_i}{dt}\right]^2}_{> 0, \text{ thus:}}$$

$$\dot{\mathcal{H}} = \sum_{i=1}^{N} -\left[\frac{d\phi_i}{dt}\right]^2 \leq 0$$

The system having a Lyapunov function $\mathcal{H}$ whose derivative $\dot{\mathcal{H}}$ is always negative means that after small perturbations to the stationary solutions $\mathcal{H}$, the system will return to that stationary solution and not spiral outwards and diverge. The solutions are therefore *stable* stationary points. The later numerical studies will confirm this, as once convergence has been reached, the state of the system does not vary strongly but always converges back to the same stationary point (see figure 4).

## 1.3   Extreme points

To actually calculate these stationary points, the gradient of $\mathcal{H}$ must equal zero. For the gradient overall to equal zero, each of the individual rows must equal zero.

Resolving one of the double sums of (4), we get

$$\mathcal{H}_i = -\frac{K}{2N}\sum_j \cos(\phi_i - \phi_j) - (\omega_i - \bar{\omega})\phi_i$$

$$\frac{\partial \mathcal{H}_i}{\partial \phi_i} = \frac{-K}{2N}\sum_j -\sin(\phi_i - \phi_j) - \underbrace{(\omega_i - \bar{\omega})}_{\omega_i = \bar{\omega} \Rightarrow 0} = 0$$

$$0 = \sum_j \sin(\phi_i - \phi_j)$$

$$0 = \sum_j \Big(\sin(\phi_i)\cos(\phi_j) - \sin(\phi_j)\cos(\phi_i)\Big)$$

$$0 = \Big(\sum_j \sin(\phi_i)\cos(\phi_j) - \sum_j \sin(\phi_j)\cos(\phi_i)\Big)$$

$$0 = \Big(\sum_j \sin(\phi_i)\cos(\phi_j) - \sum_j \sin(\phi_j)\cos(\phi_i)\Big)$$

$$\Rightarrow \sum_j \sin(\phi_i)\cos(\phi_j) = \sum_j \sin(\phi_j)\cos(\phi_i)$$

or also

$$0 = \sum_j \sin(\phi_i - \phi_j)$$

A sufficient criterion for reaching a stationary point is thus $\phi_i = \phi_j$.

## 1.4   Solutions in terms of $r$

Show that the stationary solutions of the Kuramoto model satisfies

$$\Delta(\omega_i) = Kr\sin(\phi_i - \psi) \tag{6}$$

where

$$re^{i\Psi} := \frac{1}{N}\sum_{j=1}^{N} e^{i\phi_j} \tag{7}$$

Starting from $\frac{-K}{2N}\sum_j(-1)\sin(\phi_i - \phi_j) - (\omega_i - \bar{\omega}) = 0$, let's use the identity of a complex exponential function according to which the imaginary part of a function can be defined as

We can rearrange Euler's formula

$$e^{ix} = \cos x + i\sin x$$

into

$$\sin(x) = \frac{e^{ix} - e^{-ix}}{2i}$$

Substituting in $x = \phi_i - \phi_j$:

$$\sin(\phi_i - \phi_j) = \frac{e^{i(\phi_i - \phi_j)} - e^{-i(\phi_i - \phi_j)}}{2i}$$

In the stationary solutions, we can rearrange and substitute as follows:

$$\frac{K}{2N}\sum_j \sin(\phi_i - \phi_j) - (\omega_i - \bar{\omega}) = 0$$

$$(\omega_i - \bar{\omega}) = \Delta\omega_i = \frac{K}{2N}\sum_j \sin(\phi_i - \phi_j)$$

$$\Delta\omega_i = \frac{K}{2N}\sum_j \frac{e^{i(\phi_i - \phi_j)} - e^{-i(\phi_i - \phi_j)}}{2i}$$

$$= \frac{K}{2N}\frac{1}{2i}\sum_j (e^{i(\phi_i - \phi_j)} - e^{-i(\phi_i - \phi_j)})$$

$$= \frac{K}{2}\frac{1}{2i}\frac{1}{N}\sum_j (e^{i(\phi_i - \phi_j)} - e^{-i(\phi_i - \phi_j)})$$

$$= \frac{K}{2}\frac{1}{2i}r(e^{i(\phi_i - \psi)} - e^{-i(\phi_i - \psi)})$$

$$= \frac{K}{2}r\frac{e^{i(\phi_i - \psi)} - e^{-i(\phi_i - \psi)}}{2i}$$

$$= \frac{K}{2}r\frac{e^{i(\phi_i - \psi)} - e^{-i(\phi_i - \psi)}}{2i}$$

$$= \frac{K}{2}r\sin(\phi_i - \psi)$$

# Model with $N$ identical oscillators

## 1.5 Model in terms of r

Show that the Kuramoto model with identical ($\omega_i = \omega$) oscillators

$$\dot{\theta}_i(t) = \omega + \frac{K}{N} \sum_{j=1}^{N} \sin(\theta_j(t) - \theta_i(t)) \tag{8}$$

can be expressed in terms of order parameter $r$ as

$$\dot{\theta}_i(t) = r(t) \sin(\psi(t) - \theta_i(t)) \tag{9}$$

when $K = 1$ and $\omega = 0$.

Using the trigonometric identity that $\sin(\alpha - \beta) = \sin\alpha\cos\beta - \cos\alpha\sin\beta$ again, we can say that

$$\begin{aligned}
\dot{\theta}_i(t) &= 0 + \frac{1}{N} \sum_{j=1}^{N} \sin(\theta_j - \theta_i) \\
&= \frac{1}{N} \sum_{j=1}^{N} \left( \sin\theta_j \cos\theta_i - \cos\theta_j \sin\theta_i \right) \\
&= \frac{1}{N} \sum_{j=1}^{N} \left( \cos\theta_i \sin\theta_j - \sin\theta_i \cos\theta_j \right) \\
&= \frac{1}{N} \left( \sum_{j=1}^{N} \cos\theta_i \sum_{j=1}^{N} \sin\theta_j - \sum_{j=1}^{N} \sin\theta_i \sum_{j=1}^{N} \cos\theta_j \right) \\
&= \frac{1}{N} \left( N\cos\theta_i \underbrace{\sum_{j=1}^{N} \sin\theta_j}_{\sin\Psi} - N\sin\theta_i \underbrace{\sum_{j=1}^{N} \cos\theta_j}_{\cos\Psi} \right) \\
&= \frac{1}{\cancel{N}} \cancel{N} \cos\theta_i \sin\Psi - \frac{1}{\cancel{N}} \cancel{N} \sin\theta_i \cos\Psi \\
&= \underbrace{\cos\theta_i}_{m} \sin\Psi - \underbrace{\sin\theta_i}_{n} \cos\Psi \\
\dot{\theta}_i(t) &= m\sin\Psi - n\cos\Psi \\
&= r\sin(\Psi - \theta_i) = r(t)\sin(\Psi(t) - \theta_i(t))
\end{aligned}$$

## 1.6 Gradient system

Show that (8) is a gradient system, such that

$$\dot{\theta}_i(t) = \frac{\partial U}{\partial \theta_i} \tag{10}$$

for

$$U(\theta_1, \ldots, \theta_N) := \frac{1}{2N} \sum_{i,j=1}^{N} \cos(\theta_i - \theta_j) \tag{11}$$

Start by expanding $U$

$$U(\theta_1,\ldots,\theta_N) = \frac{1}{2N}\sum_{j=1}^{N}\sum_{i=1}^{N}\cos(\theta_i - \theta_j) = \frac{1}{2N}\sum_{j=1}^{N}\sum_{i=1}^{N}\left(\cos\theta_i\cos\theta_j + \sin\theta_i\sin\theta_j\right)$$

$$= \frac{1}{2N}\sum_{j=1}^{N}\left(\cos^2\theta_i + 2\sin\theta_i\sin\theta_j + \sin^2\theta_j + 2\cos\theta_i\cos\theta_j + \cos^2\theta_j + \sin^2\theta_i\right)$$

From here, we can show that the gradient of this function is equivalent to (8):

$$\frac{\partial U}{\partial\theta_i} = \frac{1}{2N}\sum_{j=1}^{N}\left(\cancel{-2\sin\theta_i\cos\theta_i} + 2\cos\theta_i\sin\theta_j - 2\sin\theta_i\cos\theta_j + \cancel{2\cos\theta_i\sin\theta_i}\right)$$

$$= \frac{1}{2N}\sum_{j=1}^{N}2\left(\cos\theta_i\sin\theta_j - \sin\theta_i\cos\theta_j\right)$$

$$= \frac{1}{2N}\sum_{j=1}^{N}2\left(\sin(\theta_j - \theta_i)\right)$$

$$\frac{\partial U}{\partial\theta_i} = \frac{1}{N}\sum_{j=1}^{N}\left(\sin(\theta_j - \theta_i)\right) = \dot{\theta}_i(t)$$

$$\underbrace{\phantom{\frac{1}{N}\sum_{j=1}^{N}\left(\sin(\theta_j - \theta_i)\right)}}_{\text{with }\omega = 0 \text{ and } k = 1}$$

## 1.7 $U$ in terms of $r$

Using another trigonometric identiy $(\cos(\alpha + \beta) = \cos\alpha\cos\beta - \sin\alpha\sin\beta)$, we can expand $U$ a different way in order to express it in terms of $r$:

$$U(\theta_1,\ldots,\theta_N) = \frac{1}{2N}\sum_{j=1}^{N}\sum_{i=1}^{N}\cos(\theta_i - \theta_j)$$

$$= \frac{1}{2N}\sum_{j=1}^{N}\sum_{i=1}^{N}\left(\cos\theta_i\cos\theta_j\right) + \sum_{j=1}^{N}\sum_{i=1}^{N}\left(\sin\theta_i\sin\theta_j\right)$$

$$= \frac{1}{2N}\left[\sum_{j=1}^{N}\left(\cos\theta_i\right)^2 + \sum_{j=1}^{N}\left(\sin\theta_i\right)^2\right]$$

$$= \frac{1}{2N}\left[N(\cos\theta_i)^2 + N(\sin\theta_i)^2\right]$$

$$= \frac{1}{2}\frac{\cancel{N}}{\cancel{N}}\left[(\cos\theta_i)^2 + N(\sin\theta_i)^2\right]$$

$$= \frac{1}{2}\left[\underbrace{(\cos\theta_i)^2}_{a^2} + \underbrace{(\sin\theta_i)^2}_{b^2}\right]$$

$$\underbrace{\phantom{(\cos\theta_i)^2 + (\sin\theta_i)^2}}_{a^2 + b^2 = r^2}$$

$$= \frac{1}{2}r^2$$

## 1.8 Conservation of mean phase

# 2   Numerical study

*Normally distributed $\omega$*

## 2.1   $r_\infty(K)$ – simulated vs predicted

### 2.1.1   Euler method

**Design and initiation**
My implementation of the Euler method is object-oriented. This may not be the computationally most efficient way, but I'm still somewhat coding beginner, so I appreciate the clarity that object-oriented programming affords. The first type of object is simply an `Oscillator` with three fields: $\omega, \theta_{s-1}$ and $\theta_s$. In the Euler method it is enough to store the present and the last value of the state variable $(\theta)$. In the rest of this documentation, I will refer to *simulation* time steps with the variable $s$ to differentiate it from the *equation* time step $t$.

An `Oscillator` gets initiated with its natural frequency and the current-period $\theta_s$. The last-period $\theta_{s-1}$ is initiated as zero, since the first part of making an 'Euler step' forwards in time is handing over $\theta_s$ to $\theta_{s-1}$.

The other object type is a population of oscillators, `OscPopulation`. It consists of a list of `Oscillators` and a construction mode, namely the probability distribution of the $\omega$. When an `OscPopulation` is initiated, for each `Oscillator` a natural frequency and initial phase are drawn from the respective types of random distribution. An `Oscillator` object is then initiated with those values for $\omega$ and $\theta_s$ and assigned to a place in the list within the `OscPopulation` object. The `OscPopulation` object thus possess an indexable list of all oscillators.

**Running the simulation**
The simulation has two functions, to run in two modes, `OscPopulation.runK()` and `OscPopulation.runT()`. I will get into the latter in subsection 2.2.

To start a simulation for finding $r_\infty(K)$, a new `OscPopulation` object gets created, passing the desired distribution type of $\omega$ to it. Then `runK()` is run on this object. It loops through the different values for K, starting with resetting the oscillators in the population, then Euler-stepping all oscillators through time until T and finally calculating $r_\infty(K)$. Take each in turn.

The oscillators are reset by by finding new random values for $\omega$ and $\theta_s$ for all `Oscillators` and setting their field value equal to the newly calculated value. Same as during the original initiation, $\theta_{s-1}$ is set to zero.

The Euler-step function `OscPopulation._oneStepForAll()` is straight-forward in principle, but needs to be optimised for reasonable computation times. Originally, running task 1 with $N = 100$, $K = [0, 4]$ and $dK = 0.1$ took about 45 minutes. Since the computation time likely increases exponentially with $N$ (more neighbouring oscillators have to be considered at *each* step) the full compute time for $N = 1000$ would have been unreasonable. Thus, I make use of the `numba` package which offers 'decorators' for functions. Decorators are functions which take functions as inputs and return modified functions. In this case, the `@jit` decorator converts my Euler-step computation function into optimised machine code using the LLVM compiler. This can yield computation speeds similar to C or FORTRAN.[1] It reduced the compute time for task 1 (with $N = 100$) from 45 minutes down to 4 minutes. To make it work, however, the function needs to take `nympy.ndarrays` as inputs, not objects, like originally the case in my object-oriented code. Therefore, I wrote the wrapper function `oneStepForAll` around `_oneStepForAll` (which actually does the computation). It retrieves the values from the `OscPopulation` object (a getter), puts them into temporary variables and then calls the optimised function `_oneStepForAll` using those temporary variables (namely, arrays for $\omega, \theta_{s-1}, \theta_s$). Finally, a setter writes the returned arrays for $\theta_{s-1}, \theta_s$ onto the `Oscillator` objects in the `OscPopulation` .
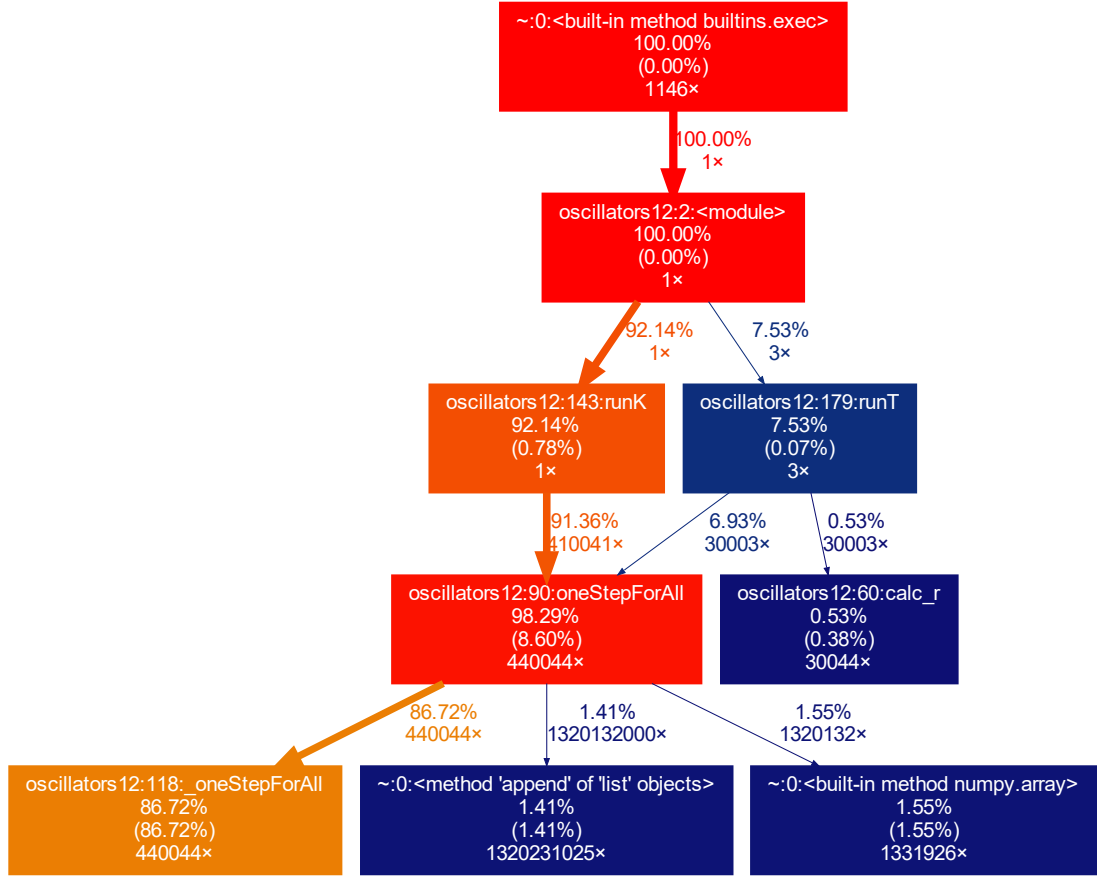
---

[1] http://numba.pydata.org/

Figure 1: Profile of code for task 1 and 2

The core `_oneStepForAll` function implements a standard Euler approximation. First, the $\theta$ are stepped through time by handing the value of $\theta_s$ over to $\theta_{s-1}$. In the next step, the sum of the the $sin$ of the differences between the last-period $\theta$ of oscillator $n$ ($\theta_{s-1}^n$) and all other oscillators $j$ ($\theta_{s-1}^j$) is computed:

$$sum = \sum_{j=0}^{N} = \sin(\theta_{s-1}^j - \theta_{s-1}^n)$$

From this, we can calculate the discrete $\Delta\theta_s^n$

$$\Delta\theta_s^n = \omega_n + \frac{K}{N} * sum$$

The new $\theta_s$ of oscillator n is then

$$\theta_s^n = \theta_{s-1}^n + \delta t \cdot \Delta\theta_s^n$$

This is done for all N oscillators.

After all oscillators have been stepped one simulation step $\delta s$ forward, this is repeated $\frac{T}{\delta t}$ times to get to the final time point $T$. At this point, the limiting coherence parameter $r_{(}T)$ is calculated for the current $K$ and appended to a list of all $r_\infty(K)$.

A new loop with the next $K$ begins until all $K$ values have been exhausted. The full results are returned as a list which is then graphed. A scatter plot of $r_\infty(K)$, as opposed to a continuous line, has been chosen since a straight connection between points cannot be assumed.

### 2.1.2   Program profile

To make the structure of the code more clear and see which parts of the Euler-based simulation take the longest to compute, I profiled my python code. This profile shows which functions called which others as well as information about the execution of the respective function. After the name of the function, the first line shows the total time spent in this function (in itself and its children), the second how much time was spent only in itself and the third how many times the function was called.

The profile of the code for tasks 1 and 2 can be seen in figure 1. Even when optimised, the computation time for these tasks is still 1h 30min. As the profile shows, almost all of this time (87%) is spent in the main computation method `_oneStepForAll`, which was called almost half a million times. The wrapper method retrieved 1.3bn and set over 800m values.

### 2.1.3   Numerical integration

The strategy to integrate the consistency equation is to loop through different values of $r$ and check which ones make the right-hand side approximately equal to one. The precision of this approximation can be adjusted by changing the `rHysteresis` variable, which sets the acceptable range of values around 1 for the right-hand side of the consistency equation. This is the `calc_r_integ(K)` function which takes $K$ as an input.

We now cycle through the different values for $K$ in a separate loop to get the values of the desired $r(K)$ relationship. Since this is not a computationally-intensive step, I increased the resolution of $K$ to $\delta K = 0.01$. The hysteresis around the left-hand side value is 0.01 and the step size of candidate $r$ is $\delta r = 0.001$. The results are then graphed in figure 2 together with the simulated points.
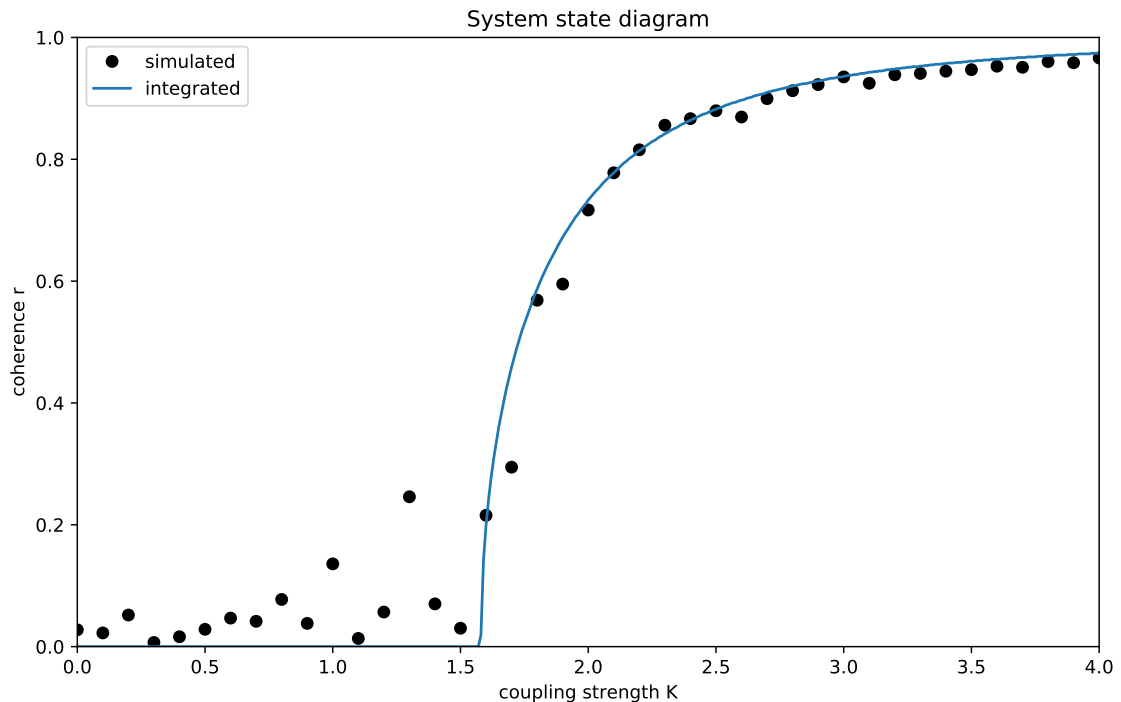
### 2.1.4   Results



Figure 2: Diagram of limiting $r_\infty$ for different values of K

The first observation is that, on the whole, the simulation values track the theoretical values quite closely. More precisely, the fit is *very good* for $2 < K$, *good* in $0 < K < 1$ but has large deviations in $1 < K < 2$.

My first impulse for improving the fit of the simulation lies in shrinking $\delta t$ to reduce the local and global error (which are proportional to $(\delta t)^2$ and $\delta t$, respectively). Such errors happen because ODEs *on paper* are continuous and essentially have infinite precision, which is impossible to achieve *in silico*. In some non-linear dynamical models, such deviations in the discretised simulation can lead to *qualitatively* different outcomes, especially when the system exhibits sensitive dependence on initial conditions (ICs). I do not believe the global error is large enough to cause the large positive deviations we observe just before the critical value $K_{crit} = 1.5958$.
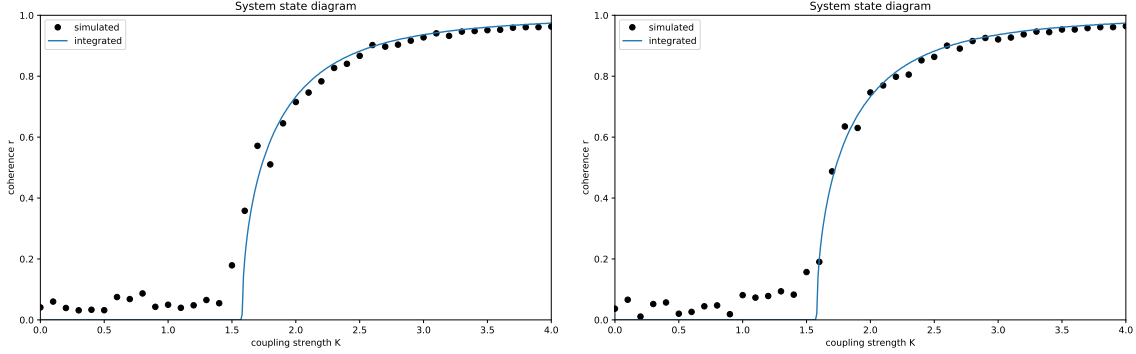


Figure 3: Alternative runs of $r_\infty(K)$

Instead, I hypothesise that the high-coherence observations in the range $1 < K < 1.5$ (which theory predicts should be very close to 0), are due to special sets of ICs. Out of all possible ICs of individual oscillators, a small subset will favour convergence of all oscillators to a (partially) synchronised state. If enough oscillators have correlated ICs, this could lead to higher-than expected coherence in the system. If it is true that the outliers in the range $1 < K < 1.5$ are caused by unlikely constellations of ICs, then increasing N should diminish the occurrence of unlikely starting conditions. The probability of any IC to be coherence-favouring does not increase with N; in fact it may decrease because the union of coherence-favouring ICs between all oscillators shrinks. As N gets larger, it thus becomes progressively less likely that a sufficiently large portion of the oscillator population (a 'critical mass') is randomly drawn with collectively-coherence-favouring ICs.

I tested this hunch by re-running the simulations a few times (ie. with different sets of ICs) and finding that the large deviations from the theoretical values are indeed uncommon. In the alternative runs in figure 3, the simulated values are much closer to where they should be, indicating that in my initial run I must have had some 'bad luck' in the random draws for a few values of $K$.

## 2.2   $r(t)$ for $K = [1, 2, 3]$

The simulation procedure for is the same as for `runK()` , with 2 exceptions:
1. there is still a $K$-loop (with limited values $K = [1, 2, 3]$), but there is no resetting of the population to new random values at the beginning of each such loop.
2. $r(t)$ is now calculated at every (simulation!) time step $t$.

The previous method of assigning new random values to the state variables of all oscillators did not work here for some reason. Instead, a bug occurred in which the final values of the last run were used as the starting values of a new run.[2] Therefore I had to go another way: the `OscPopulation` object is created anew in every loop. After the object is initiated with the right distribution of $\omega$, `runT()` is

---

[2]   .. despite many attempts to fix this bug. My hypothesis is that this is about memory management in python. Sometimes during other assignment of variables or objects, it merely creates a pointer instead of copying the value into a separate location in memory.

executed. It loops through all simulation time steps $s$. In one such loop, all oscillators are updated, then `calc_r()` is called to compute $r_s$.

`calc_r()` calculates $r_s$ in one of two different but equivalent ways. In one way, actual complex numbers are added up and divided by N. To get a value for $r$, the modulus of the resulting complex number is taken.

$$r_s = \frac{1}{N} \cdot \left| \sum_{n=0}^{N} e^{i\theta_s^n} \right|$$

The other way is to compute the double sum of real and imaginary component of the complex number (using the *sin* and *cos* relationships after DeMoivre and Euler). These two sums are also divided by N, yielding the average real and imaginary component of the `OscPopulation` at simulation time $s$.

$$realsum = \sum_{n=0}^{N} \frac{\cos(\theta_s^n)}{N}$$

$$imsum = \sum_{n=0}^{N} \frac{\sin(\theta_s^n)}{N}$$

From there, $r$ can be calculated through a square root:

$$r_s = \sqrt{realsum^2 \cdot imsum^2}$$

I tried both versions to double check the results and to see whether one would give a performance boost over the other, but both have approximately the same speed. In the end, `calc_r()` returns a list of $r(t)$ which will be appended to a list-of-lists with the $r$ values for all chosen $K$. These different trajectories are once again graphed: figure 4.

### 2.2.1   Results[3]

Of note in figure 4 is firstly that populations with $K < K_{crit} \approx 1.5985$ do not converge to coherence, while those with $K > K_{crit}$ do. In this first sense, it is in agreement with theory. And when convergence happens, it happens quickly, before $t = 10$.

Next, figure 4 confirms theory in the sense that the simulated points in figure 2 are a good representation of the system and not partial to large random fluctuations (which would make the simulated points in the diagram more arbitrary). The results of this section thus re-affirm the results in section 2.1.4. The argument is as follows.

The consistency equation predicts essentially three ranges of $K$ with distinguishable features, as we can see in figure 2. In the first region $0 < K < 1.5$, $r_\infty$ stays glued to zero. Afterwards, it predicts a sudden, almost discontinuous jump between $K = 1.5$ and $K = 2$. From $K = 2$ onwards, the functional form is logarithmic (or, more approximately, linear). The aforementioned jump does not carry $r$ all the way to $r = 1$ but only to $r \approx 0.7$ – there is some way for it to still increase. In this region, $r$ continues to increase but at a slowing pace.

In this sense, the $r(t)$ graph confirms the theory as well, as all runs with $K > K_{crit}$ stabilise at their own, distinct level of $r$ – they do not jump between totally synchronised and totally unsynchronised. We can therefore say that with some confidence that the values in the $r(K)$ diagram are indeed indicative of the *limiting* $r_\infty(K)$. Had there been a lot of variation around the 'idealised' $r_\infty$ the values we see in the $r(K)$ graph could have consisted merely of random points from within that range of variation. Instead we see that once converged, the coherence of a system does not vary greatly and the points in $r(K)$ are accurate representations of the system. The alternative

---

[3]   Out of curiosity, I added the cases for $K = 1.6$ and $K = 3$, those being very close to and far above the critical value, respectively.

Figure 4: $r(t)$ over time for 3 different values of $K$

run in figure 5 somewhat confirms this: The system with $K = 1.6$ did not manage to synchronise this time, but instead of oscillating between very high $r$ and low $r$, it stays in the lower region.

This is good for the regions in which the simulated $r_\infty$ track the theoretical closely, and bad for regions where they do not. However, the argument about unlikely coincidences of ICs being the culprit resolves this problem.

Lastly, the theoretical results from section 1.2 on stability are thereby also confirmed: once the system has reached a stationary point, it will tend to return to this point and not deviate from it greatly. The lines of systems that converged are horizontal and approximately straight, with only small variations – the stationary points are stable. The higher the coupling coefficient $K$, the stronger is that stability and the lower is the variation.
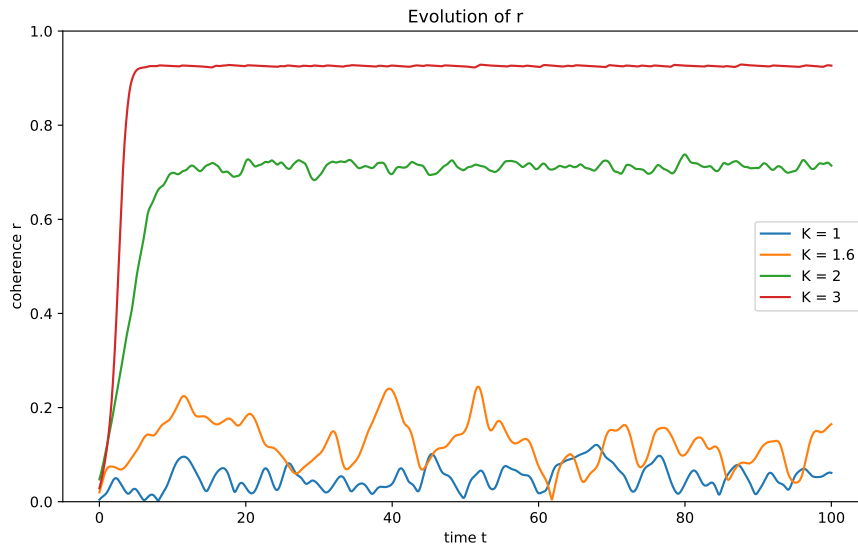


Figure 5: $r(t)$ over time for 3 different values of $K$

## *Uniformly distributed $\omega$*

**Program profile**
The profile of the code for tasks 3, 4 and 5 can be seen in figure 6.
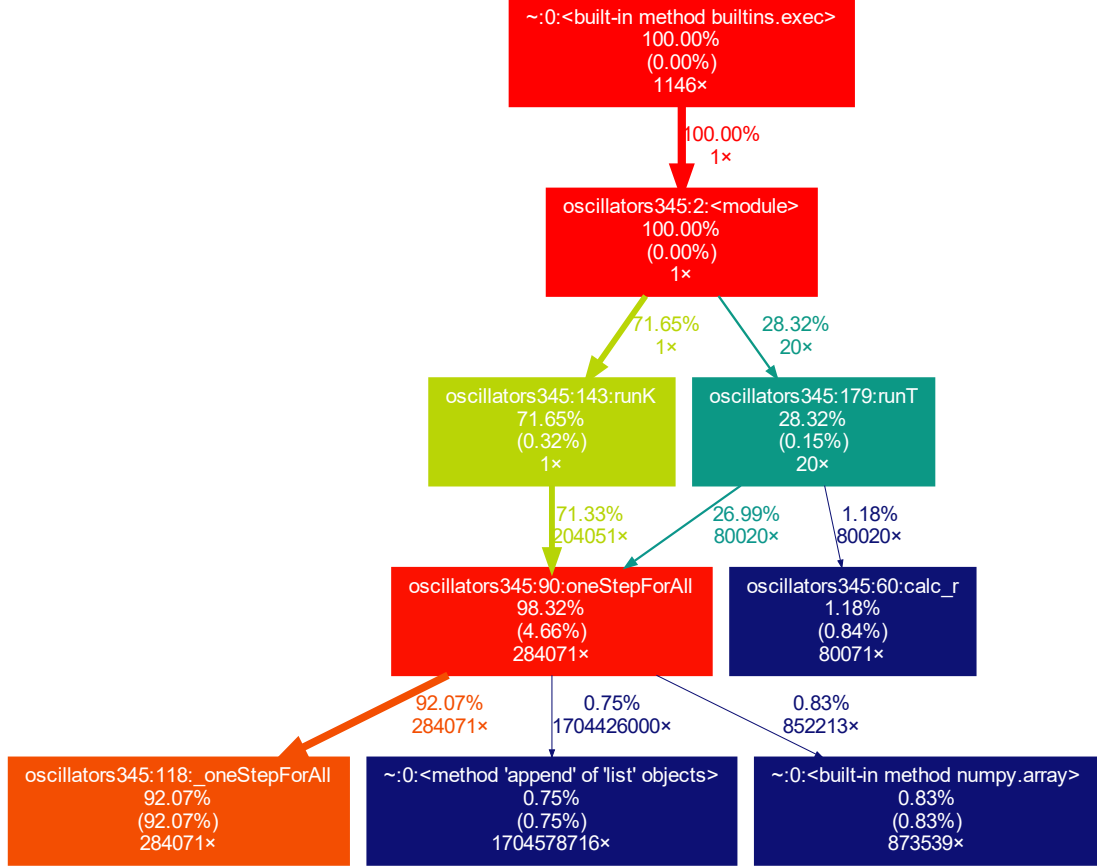


Figure 6: Profile of code for task 3, 4 and 5

## 2.3 $r_\infty(K)$

It can firstly be observed that also the systems with a uniform distribution of $\omega$ exhibit criticality. Different from the case of normal distribution of $\omega$, the criticality seems to be stronger. The jump from mostly unsynchronised to mostly synchronised happens over a smaller range of $K$ and it happens more abruptly, going from $r(K = 0.60) \approx 0.1$ to $r(K = 0.65) \approx 0.8$. This represents a $\Delta r = 0.7$ within $\Delta K = 0.05$ against a $\Delta r = 0.6$ within $\Delta K = 0.5$ in the case of normally distributed $\omega$.

Secondly, figures 7b and 7c show that the critical point is lower for the uniform distribution ($K_{crit} \approx 0.6$) than for the normal distribution ($K_{crit} \approx 1.6$). This indicates that a uniform distribution of natural frequencies favours convergence to a synchronised state even when oscillators are coupled much weaker. This is mostly likely due to the fact that the given uniform distribution greatly restricts the range that any $\omega$ can take. Now, $-\frac{1}{2} \leq \omega \leq \frac{1}{2}$ when before, it could routinely take values as high or low as $\pm 1$ (68% of $\omega$ will lie in this range). A larger range of natural frequencies makes synchronisation harder. Thus a stronger coupling force is required to make our normally-distributed system synchronise.[4]

---

[4] This is not true generally, however. It just so happens that the uniform distribution given to us is much narrower

(a)



(b) $r_\infty(K)$ for *uniform* $\omega$-distribution



(c) $r_\infty(K)$ for *normal* $\omega$-distribution (figure 2)
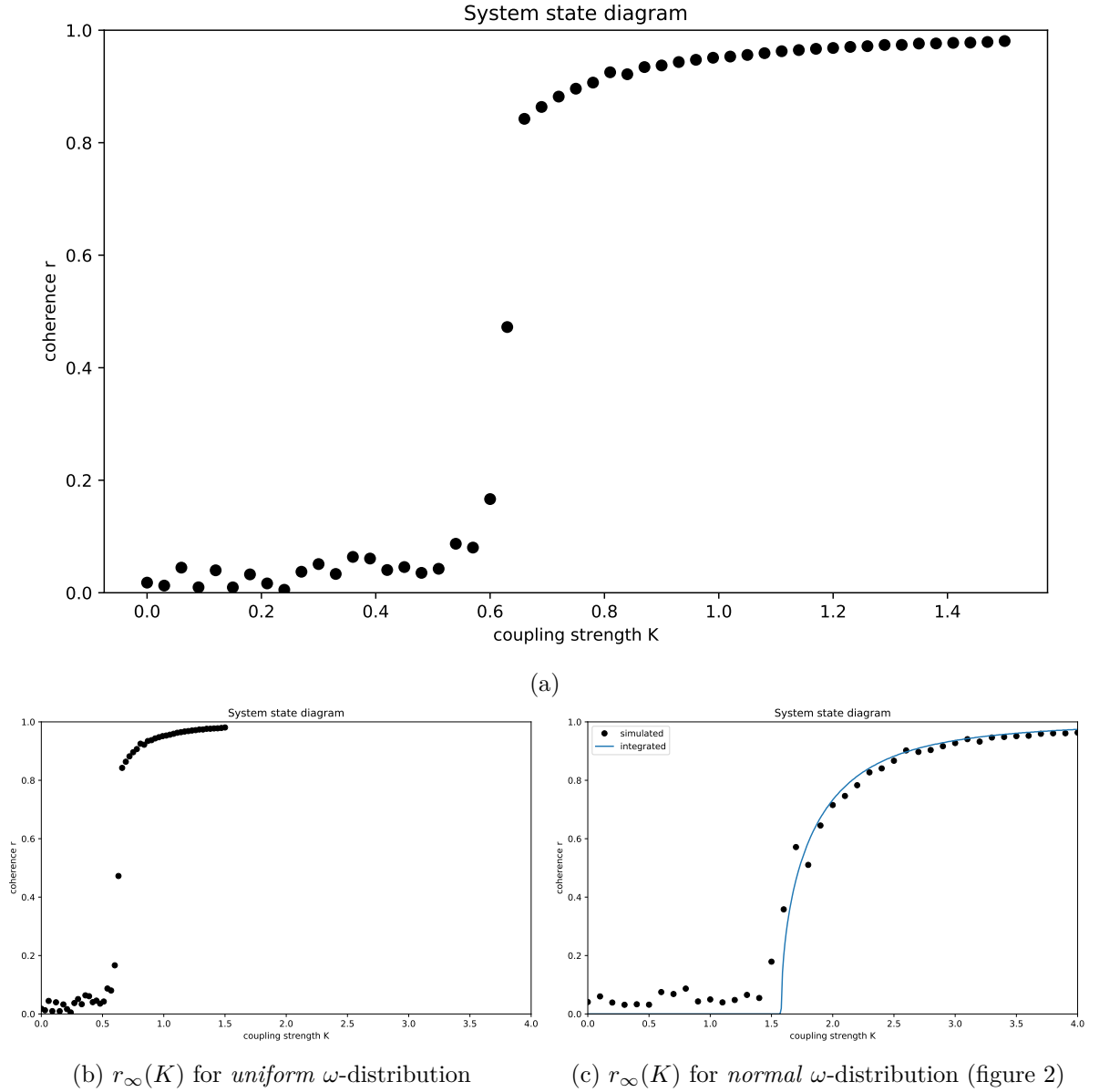
Figure 7: $r_\infty(K)$ for different values of K

## 2.4  $r(t)$ **for different initial conditions** $\theta_0$

To run simulations repeatedly while keeping selected initial conditions the same, I implemented the following. First, a 'reference' `OscPopulation` object is initiated. Then a loop begins in which a 'temporary' `OscPopulation` objects is initialised with all new random values. The desired initial conditions from the reference object are then written into the temporary object: here, the $\omega$ are kept, in section 2.5 the $\theta$. `runT()` is executed on the temporary object and the returned trajectory saved into another list-of-lists.

---

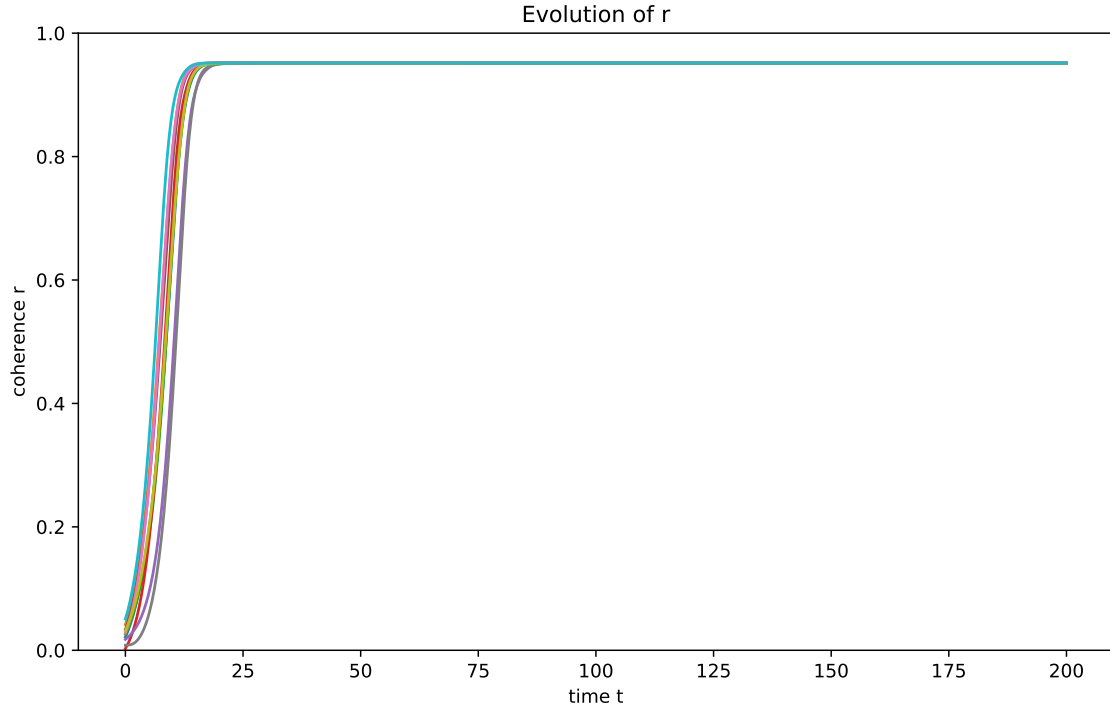than the (Standard) normal distribution given to us.

Figure 8: $r(t)$ over time for 10 different initial conditions of $\theta$

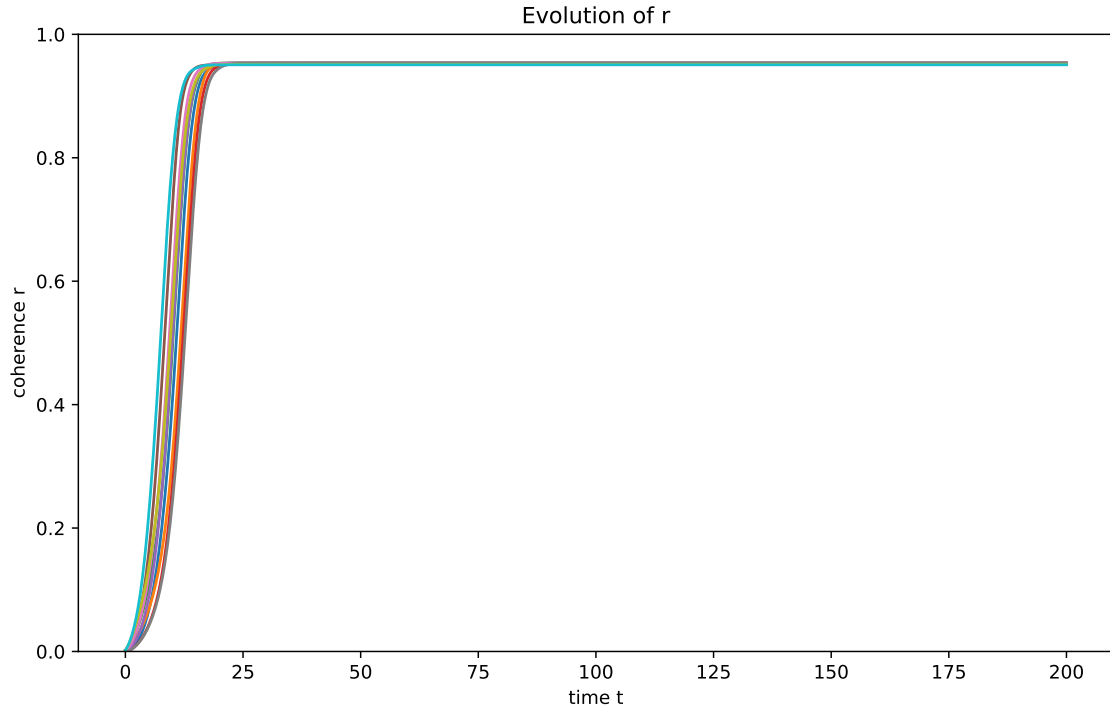## 2.5  $r(t)$ for different initial conditions $\omega$



Figure 9: $r(t)$ over time for 10 different initial conditions of $\omega$

## 2.6  Interpretation

To my eye, the graphs in figures 8 and 9 look very similar. They both converge very quickly –in less than 20 time periods– since the given $K = 1$ is much larger than the required $K_{crit} \approx 0.6$. They also converge to the same level, $r_\infty \approx 0.95$.

If I squint, I can potentially see a greater variance around the stationary point in the case of changing the $\omega$ (figure 9) as opposed to changing the $\theta_0$. This could be interpreted as the initial conditions in $\omega$ having a greater impact on the future of the system than the $\theta_0$. But this visible effect is very small and most likely random noise.

The phenomenon to be explained is then why the two cases do *not* differ significantly. As we saw in the general juxtaposition of normally and uniformly distributed $\omega$, it seems that $\omega$ as the defining/characteristic property of an oscillator *can* have a profound impact on the behaviour of the system (namely, changing the criticality properties). Thus it could be expected that when all $\omega$ are changed, this should have a larger effect, ie. cause larger deviations between the different runs, than changing merely the initial positions of the oscillators but leaving their 'characteristic property' $\omega$ the same.

Ostensibly, this is not true. My conclusion is therefore that *qualitative* changes to the oscillators' properties (such as changing the *type* of distribution) can cause the system to behaviour differently, but merely *quantitative* changes in the same regime cannot.