

به نام خدا

دانشگاه صنعتی شریف - دانشکده مهندسی کامپیوتر

جبر خطی - دکتر سمیرا حسین قربان

نیم سال اول ۱۴۰۱-۱۴۰۰

پروژه همگامی در شبکه‌های پیچیده

طراحان:

امیر محمد محمدی (a.m.mohammadi266@gmail.com) ، سجاد فغفور

مغربی، حمید دهباشی، امیر حسین باقری، محمد جعفری، نازنین آذریان.

موعد تحویل: ۱۲ بهمن

از همفکری و مشورت استقبال می‌کنیم ولی ضروری است که پیاده‌سازی‌ها حتماً به تنهایی و بدون مشاهده کد دیگران صورت پذیرد.

لطفاً در صورت همفکری با سایر دانشجویان، نام و شماره دانشجویی آنها را وارد کنید.

"In a world seemingly filled with chaos, physicists have discovered new forms of synchronization and are learning how to predict and control them."

Road Map

In this project, the following steps are performed:

1. Introducing the concepts of complex networks and synchronization.
2. Familiarity with Runge-Kutta tool and graph data structure.
3. Measuring the degree of synchronization of graph using kuramoto order parameter.
4. Rewiring algorithm implementation to improve synchronization.

Complex Network (100 points)

Complex networks describe a wide range of natural and/or artificial systems, e.g. harmonic clapping of the crowd, simultaneous flashing of fireflies, the Internet, the WWW, networks of food webs, social acquaintances, paper citations, as well as many others.

[More information about CN](#)

https://www.researchgate.net/publication/324312765_What_is_a_Complex_Network_CDT-2

[Interesting video about complex networks and its samples \(https://www.youtube.com/watch?v=t_VPRCtiUg\)](https://www.youtube.com/watch?v=t_VPRCtiUg)

Synchronization

Synchronization is a fundamental physical phenomenon which can be observed in many different systems, ranging from biology to physics and even social phenomenon. "It was first described by Huygens in the 18th century, when he observed that pendulum clocks hanging on a wooden beam started to swing in unison."[\[link\] \(http://www.incite-itn.eu/wp-content/uploads/VL/Kuramoto/index.html\)](http://www.incite-itn.eu/wp-content/uploads/VL/Kuramoto/index.html)

Every synchronization appears in a system with these properties:

- "Individual parts, which have their own, individual behavior".
- "A coupling mechanism, which connects the parts and aligns their motion".

Synchronization appears in a complex system if the coupling mechanism is stronger than the individual evolvment of different parts of the system.

We could say that synchronization happens if the two following conditions are confirmed:

- "All oscillators rotate with a common frequency. This means that their state are not changing proportional to any other oscillator state."[\[link\] \(http://www.incite-itn.eu/wp-content/uploads/VL/Kuramoto/index.html\)](http://www.incite-itn.eu/wp-content/uploads/VL/Kuramoto/index.html)
- "The angular differences between the oscillators are bounded."[\[link\] \(http://www.incite-itn.eu/wp-content/uploads/VL/Kuramoto/index.html\)](http://www.incite-itn.eu/wp-content/uploads/VL/Kuramoto/index.html)

[Improve your intuition of Synchronization \(http://www.ams.org/publicoutreach/feature-column/gll-test\)](http://www.ams.org/publicoutreach/feature-column/gll-test)

```
In [3]: import copy
import math
import json
import numpy as np
import networkx as nx
import scipy as sp
from scipy.integrate import RK45
from matplotlib import pyplot as plt
```

Runge-Kutta brief explanation

If you want to find an approximate solution to the following ODE:

$$\frac{dx}{dt} = f(x)$$

You can use RK. A method of numerically integrating ordinary differential equations by using a trial step at the midpoint of an interval to cancel out lower-order error terms. We are discussing RK45 for more accuracy it provide in this project, but you can see how RK4 works as a simpler method relative to RK45.

$$k_1 = hf(x(t), t)$$

$$k_2 = hf(x(t) + \frac{k_1}{2}, t + \frac{h}{2})$$

$$k_3 = hf(x(t) + \frac{k_2}{2}, t + \frac{h}{2})$$

$$k_4 = hf(x(t) + k_3, t + h)$$

$$x(t + h) = x(t) + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)$$

[Very useful video about RK and its implementation \(https://www.youtube.com/watch?v=1FYrnwqWQNY\)](https://www.youtube.com/watch?v=1FYrnwqWQNY)

[Runge-Kutta method \(https://en.wikipedia.org/wiki/Runge%E2%80%93Kutta_methods\)](https://en.wikipedia.org/wiki/Runge%E2%80%93Kutta_methods)

Runge-Kutta training (10 points)

assume:

$$\frac{dy}{dt} = \sqrt{y}$$

$$y(2) = 1$$

plot y graph by 0.1 distance in interval (0, 10)

using [scipy.integrate.RK45](https://docs.scipy.org/doc/scipy/reference/generated/scipy.integrate.RK45.html)

(<https://docs.scipy.org/doc/scipy/reference/generated/scipy.integrate.RK45.html>) and [matplotlib.pyplot](https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.html) (https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.html)

```
In [165]: # collect data
t_values = []
y_values = []

### Begin your code

def f(t, x):
    return math.sqrt(x)

solution = RK45(f, 2, [1], 0.1, 0.1)

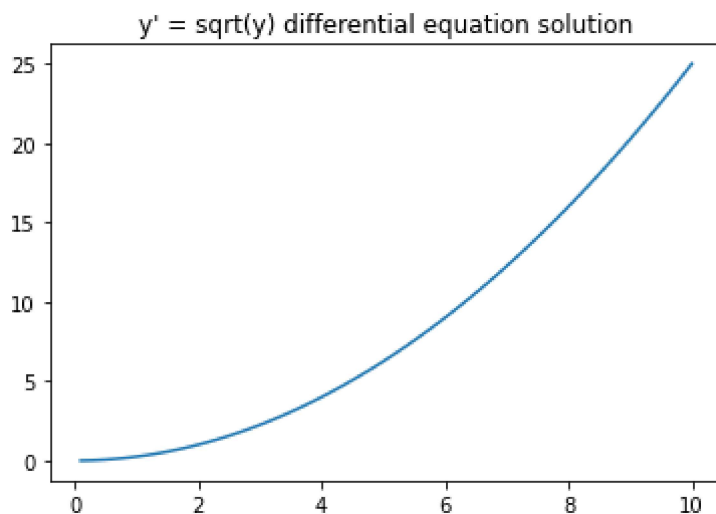
for i in range(100):
    solution.step()
    t_values.append(solution.t)
    y_values.append(solution.y[0])
    if solution.status == 'finished':
        break

t_values.reverse()
y_values.reverse()
solution = RK45(f, 2, [1], 10, 0.1)

for i in range(100):
    solution.step()
    t_values.append(solution.t)
    y_values.append(solution.y[0])
    if solution.status == 'finished':
        break

### End your code

# plot graph
plt.plot(t_values, y_values)
plt.title("y' = sqrt(y) differential equation solution")
plt.show()
```



Graph (5 points)

The cell below is the graph class that you use for the rest of the project.

Fill `get_mean_frequency` and `centroid_frequencies` methods of class and don't change other methods.

```

In [133]: # import graph and graph training
class Graph:
    # list of frequencies
    nodes_frequency = []
    # list of edges (as list like [1,2])
    adjacency_matrix = []
    naturalFreq = []
    vertices = []

    def __init__(self, nodes_frequency, adjacency_matrix, natural_frequency):
        self.nodes_frequency = nodes_frequency
        self.adjacency_matrix = adjacency_matrix
        self.naturalFreq = natural_frequency
        self.vertices = list(range(len(adjacency_matrix)))

    def number_of_nodes(self):
        return len(self.nodes_frequency)

    def get_mean_frequency(self):
        """
        Compute mean frequency of graph nodes.

        Args:
            self (Graph): graph instance.

        Returns:
            mean (float): mean frequency of graph nodes
        """
        ### Begin your code
        return sum(self.nodes_frequency) / len(self.nodes_frequency)
        ### End your code
        pass

    def centroid_frequencies(self):
        """
        Centroid graph nodes frequencies on zero.

        Args:
            self (Graph): graph instance.
        """
        ### Begin your code
        avg = self.get_mean_frequency()
        for i in range(len(self.nodes_frequency)):
            self.nodes_frequency[i] = self.nodes_frequency[i] - avg
        ### End your code
        pass

    def r(self):
        pass

    def is_edge(self, x, y):
        return self.adjacency_matrix[x][y] == 1

    def add_edge(self, x, y):
        if not self.is_edge(x, y):
            self.adjacency_matrix[x][y] = 1
            self.adjacency_matrix[y][x] = 1

```

```

def remove_edge(self, x, y):
    if self.is_edge(x, y):
        self.adjacency_matrix[x][y] = 0
        self.adjacency_matrix[y][x] = 0

def normalize_frequencies(self):
    for i in range(len(self.nodes_frequency)):
        while self.nodes_frequency[i] > math.pi:
            self.nodes_frequency[i] -= math.pi

```

```

In [134]: # fix
# Download graph
# go to this url and download graph.json https://drive.google.com/file/d/1vSjfd62
graph_json = json.load(open('graphs.json', 'r'))
graph = Graph(graph_json[0], graph_json[1], graph_json[2])

```

```

In [135]: # test Graph functions
# fix
print(graph.get_mean_frequency())
graph.centroid_frequencies()
print(graph.get_mean_frequency())

```

```

1.9224288087187333
-8.881784197001253e-17

```

Measuring the degree of synchronization (25 points)

We use the Kuramoto order parameter to measure the degree synchronization in the network:

$$re^{i\psi} = \frac{1}{N} \sum_{j=1}^N e^{i\theta_j}$$

In this equation, r is the magnitude and θ is frequency of each node. We know that $0 \leq r \leq 1$, and in order to have perfect synchronization, we must have $r = 1$. In that case, we have $\forall i, j : \theta_i = \theta_j$. We call $r = 0$, incoherence, since it shows that there is no synchronization among oscillators.

In the next cell, you must define the r function. It gets the graph as input and calculates r based on current frequencies.

[More information about Kuramoto order parameter and how it works](https://mathinsight.org/applet/kuramoto_order_parameters)
(https://mathinsight.org/applet/kuramoto_order_parameters)

Hint: There is no need to use a complex number by transferring complex numbers into two-dimensional space.

```
In [136]: # calculating r
def r(self):
    """
    calculate r parameter.

    Args:
        self (Graph): graph instance.

    Returns:
        r (float): r parameter.
    """
    ### Begin your code
    sin = 0
    cos = 0
    for i in range(len(self.nodes_frequency)):
        sin = sin + math.sin(self.nodes_frequency[i])
        cos = cos + math.cos(self.nodes_frequency[i])
    return 1 / len(graph.nodes_frequency) * math.sqrt(cos * cos + sin * sin)
    ### End your code
    pass
```

```
In [137]: # fix
Graph.r = r
```

The Kuramoto Model (30 points)

We have N oscillators which are coupled in a network. The Kuramoto model suggests an equation to calculate the phase of each oscillator based on their natural frequency and the effect of them on each other. In this equation, we get the derivative of oscillator's phase as a function of time.

$$\frac{d\theta_i(t)}{dt} = \omega_i + \kappa \sum_{j=1}^N A_{ij} H(\theta_j(t) - \theta_i(t))$$

In this model, ω_i is the intrinsic frequency of the i 'th oscillator, and A is the adjacency matrix. $H(\theta)$ is a coupling function. It shows the coupling strength of 2 oscillators with θ phase difference. So it needs to be a 2π -periodic function. We use $H(\theta) = \sin(\theta)$ in this model. κ is the coupling strength.

Since we have the derivatives, we can use RK45 to find the solution. First, define the *theta_calculator* function which gets graph, current frequencies and κ as input and calculates $\frac{d\theta}{dt}$. Then, use this function in RK45 method to solve the problem.

[More about Kuramoto model \(https://en.wikipedia.org/wiki/Kuramoto_model\)](https://en.wikipedia.org/wiki/Kuramoto_model)


```

In [138]: # calculating next step of graph
#  $H(a) = \sin(a)$ 
def theta_calculator(graph, k):
    """
    function calculate  $d\theta/dt$  for using in RK45

    Args:
        graph (Graph): graph instance.
        k (float): coupling strength.

    Returns:
        nodes_frequencies (List[float]):  $d\theta/dt$ 
    """
    ### Begin your code
    frequencies = []
    for i in range(len(graph.nodes_frequency)):
        freq = graph.naturalFreq[i]
        for j in range(len(graph.nodes_frequency)):
            freq = freq + k * graph.adjacency_matrix[i][j] * math.sin(graph.nodes_frequencies[i])
        frequencies.append(freq)
    return frequencies
    ### End your code
    pass

def next_step(graph, k, t0=0.1, t_bound=10, rtol=0.001, atol=1e-06):
    """
    function calculates graph nodes frequencies using the above function and RK45

    Args:
        graph (Graph): graph instance.
        k (float): coupling strength.

    Returns:
        graph (Graph): graph instance in epsilon = 1e-06s ahead
    """
    ### Begin your code
    def func(t, y):
        return theta_calculator(graph, k)
    solution = RK45(func, t0, graph.nodes_frequency, t0 + rtol * 2, rtol, atol)
    graph_copy = copy.deepcopy(graph)
    solution.step()
    graph_copy.nodes_frequency = solution.y
    return graph_copy
    ### End your code
    pass

```

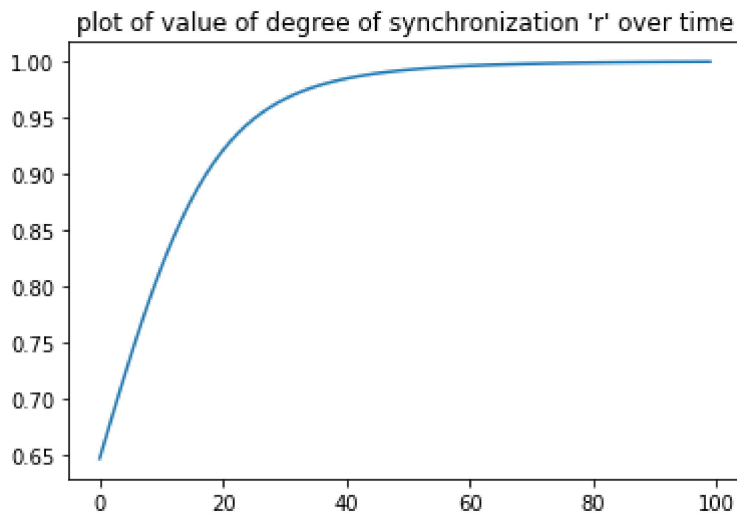
```

In [166]: # calculate n step and plot graph r
def plot_rn(graph_main, k, n):
    """
    calculate next n step and plot r-n graph using pyplot

    Args:
        graph (Graph): graph instance.
        k (float): coupling strength.
        n (int): number of steps
    """
    graph = copy.deepcopy(graph_main)
    # do not use graph_main
    ### Begin your code
    r_values = []
    t_values = []
    r_values.append(graph.r())
    t_values.append(0)
    for i in range(n - 1):
        graph = next_step(graph, k)
        r_values.append(graph.r())
        t_values.append(i + 1)
    plt.plot(t_values, r_values)
    plt.title("plot of value of degree of synchronization 'r' over time")
    plt.show()
    ### End your code

plot_rn(graph, 10, 100)

```



Accept-Reject and Rewiring (30 points)

Imagine there are a fixed set of frequencies and our goal is to define a structure of network which optimizes the synchronization represented by the measure r which has been already explained. In fact, checking all possible states requires $O(N!)$ while N is a number of nodes. Obviously, it is not efficient enough. Thus, we try to reach an approximate solution. The method used here is Accept-Reject. In each stage, We rewire the current network in order to obtain better synchronization(which is measured by r). If the rewiring result is better than the current network, it is accepted, else, it is rejected (in [reference 1](https://www.researchgate.net/publication/260439709_Optimal_Synchronization_of_Complex_Networks) (https://www.researchgate.net/publication/260439709_Optimal_Synchronization_of_Complex_Networks you can find a theoretical criteria to accept or reject rewired graph with linear formulation of the Kuramoto model based on the eigenvalues of the Laplacian matrix).

In the below cells, you should implement the rewire method which deletes 2 edges and adds 2 new edges to the network. Plus Accept-Reject checks the r measure between the new graph that has been rewired m times and the current graph and accepts the new network if it is more synchronized. And also n is a maximum number of iterations to find the optimal solution.

In [160]:

```
# rewire graph
def rewire(graph_main):
    """
    remove 2 edge and add 2 edge randomly.
    feel free to define new functions in case you want.

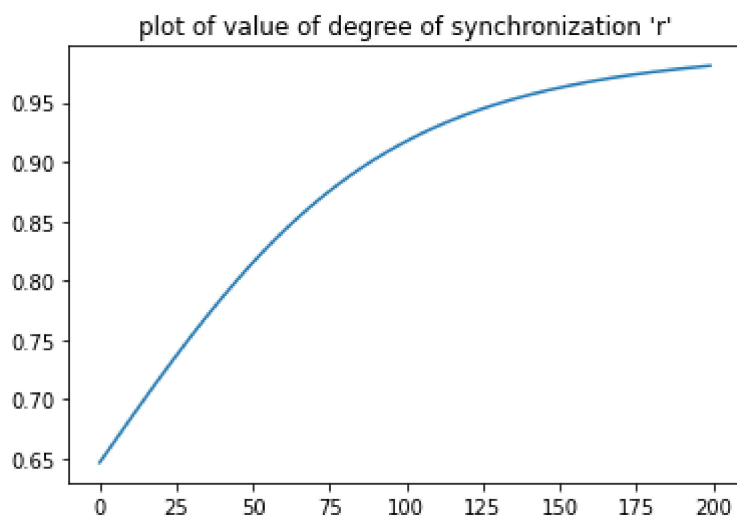
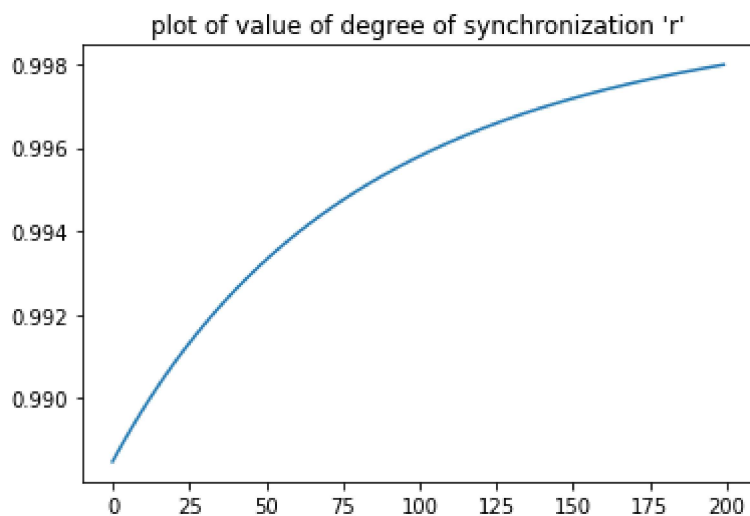
    Args:
        graph (Graph): graph instance.

    Returns:
        graph (Graph): rewired graph instance.
    """
    graph = copy.deepcopy(graph_main)
    # do not use graph_main
    ### Begin your code
    deleted = 2
    added = 2
    size = len(graph.nodes_frequency)
    while deleted > 0:
        i = math.floor(np.random.random() * size)
        j = math.floor(np.random.random() * size)
        if graph.adjacency_matrix[i][j] == 1:
            deleted = deleted - 1
            graph.adjacency_matrix[i][j] = 0
    while added > 0:
        i = math.floor(np.random.random() * size)
        j = math.floor(np.random.random() * size)
        if graph.adjacency_matrix[i][j] == 0:
            added = added - 1
            graph.adjacency_matrix[i][j] = 1
    return graph
    ### End your code
```

```
In [161]: # accept-reject algorithm (based on r in next m step)
def accept_reject(graph, n, m, k):
    """
    rewired graph, then compare rewired graph r in next m step with old graph r; i
    repeat this process n times.

    Args:
        graph (Graph): graph instance.
        n (int): number of steps.
        m (int): number of steps for checking r.
        k (float): coupling strength.
    """
    ### Begin your code
    old_graph = copy.deepcopy(graph)
    new_graph = copy.deepcopy(graph)
    for i in range(n):
        new_graph = rewire(old_graph)
        for j in range(m):
            new_graph = next_step(new_graph, k)
            if new_graph.r() > old_graph.r():
                old_graph = new_graph
    return old_graph
### End your code
```

```
In [164]: # fix
# test rewiring
graph_rewired = accept_reject(graph, 100, 10, 0.5)
plot_rn(graph_rewired, 2, 200)
plot_rn(graph, 2, 200)
```



References

1. [Skardal, P. S., Taylor, D. & Sun, J. Optimal synchronization of complex networks. Phys. Rev.Lett. 113, 144101 \(2014\).](#)
(https://www.researchgate.net/publication/260439709_Optimal_Synchronization_of_Complex_N

2. [Yamir Moreno & Amalio F Pacheco, Synchronization of Kuramoto oscillators in scale-free networks, EPL \(Europhysics Letters\) 68, 603 \(2004\).](#)
(https://www.researchgate.net/publication/1941138_Synchronization_of_Kuramoto_Oscillators_in_Free_Networks/citations)
3. [da F. Costa, Luciano. \(2018\). What is a Complex Network? \(CDT-2\).](#)
[10.13140/RG.2.2.10450.04804/2.](https://www.researchgate.net/publication/324312765_What_is_a_Complex_Network_CDT-2)
(https://www.researchgate.net/publication/324312765_What_is_a_Complex_Network_CDT-2)

