**MATH3024 Project Interim Report**

Project: MATH3024 Interim report

Student: Fraser Paterson (22258324)

September 2021

University of Western Australia

**Preface:**

This All implementations have yet to be handled by their respective “odeint” functions, what follows is merely a discussion as to the general approach I have aimed to take in the course of the project, not al of the code will work as it is presently situated.

**1.0**

a.) The code in dynsys2021.py numerically integrates the system in question 3 of assignment 1. The provided code was filled with various bugs, most of which I have stripped away.

b.) I am attempting to accomplish the The synchronisation of the x components by creating an instance of “x\_init” called “x\_init\_prime” inside the new (coupled) dynamics function. This function is called “dynamics\_question\_b”. After computing the dynamics for the “xdot” vector, “x\_init\_prime” is then used to compute the dynamics of the “xdot\_prime” vector. “xdot\_prime” is merely the copied “xdot” vector where the appropriate substitutions have ben made for an x-x coupling. Once “x\_dot\_prime” has been computed, it is possible to calculate the component-wise errors and their associated dynamics. Finally, the vector of error dynamics is returned by the function.

c.) The y-y and z-z couplings can be made in precisely the same manner as in the above description for the x-x coupling. All that is required is to move the sigma\*(error) term to the desired component for the coupling – making sure to index the correct variable in the error term. See attached code below. The y-y coupling is performed in the “dynamics\_question\_c\_yy” function and the z-z coupling is performed in the “dynamics\_question\_c\_zz” function. Both the former and the latter remain to be properly implemented – I still need to calculate their respective error dynamics.

d.) The generation of the noise and the modification to the couppling term is simple enough to implement with the numpy library. I have yet to implement this.

e.) The Pecora-Carrol slave-master systems are implemented as “dynamics\_question\_e\_x\_driver”, “dynamics\_question\_e\_y\_driver” and “dynamics\_question\_e\_z\_driver”, respectively. I might change my present setup to calculate the eigenvalues of the Jacobian, in addition to returning the error system fr a graphical representation. There is not yet any means to run these functions, I’m working on this.

**2.0**

a.) I have not yet computed the required metrics for the specific Erdos-Renyi graph, however these graph properties are simple to compute with the use of the matplotlib, numpy and networkx python libraries. I shall wite a function for each required aspect of the Erdos-Renyi graph, to display the graph alongside the required properties.

b.) As with part a, I have yet to compute any such values. The method of execution is identical to that in part a; it will be simple enough to generate a graphical representation of the required properties for the given Small-World Graph.

c.) I plan on utilising networkx to generate and conduct the required manipulations upon the Barabasi-Albert Graph. I intend to use matplotlib as the visualisation library. For instance G\_0 = nx.barabasi\_albert\_graph(300, 6) will generate the graph.

d.) I have yet to begin the implementation of part d. However, the creation of a 4-regular graph with 400 vertices is trivial enough to implement in networkx; lattice = nx.random\_regular\_graph(4, 400) will create the initial lattice. I then intend on assigning a random integer to each vertex via a call to random.randrange(4, end) – where “end” is some integer greater than 4. The “whole” graph will then be stored as a hash table where each key is a unique graph node and each value is the number of sand grains assigned to the node. The dissipation for part I can then be conducted by randomly selecting 80 nodes from the graph. For each randomly selected node we then subtract 1 from its value in the hash table and assign neighbors = nx.Graph.neighbors(n), where n is the current node.

We then call random.choice(neighbors) and increment the number of sand grains for the node returned by random.choice(neighbors). The plots will be simple to construct via matplotlib.pyplot.

The implementation for part II will simply extend that of part I, all that needs revising is the conditions under which a sand grain is lost and/or gained.

**3.0.b**

I intend to implement the “I herd you!” model from the complexity zoo: <https://www.complexity-explorables.org/explorables/i-herd-you/>.

**4.0**

The Code as it currently stands is as follows. Please note the various comments placed to explain my choice of implementation-related detail.

import numpy as np

import matplotlib.pyplot as plt

from scipy.integrate import odeint

params = { 'figure.figsize': (8,8),

'axes.labelsize': 40,

'lines.linewidth': 2,

'font.size': 12, # this needed to be changed

'lines.color': 'r',

'xtick.labelsize': 20,

'ytick.labelsize': 20,

'legend.fontsize': 10,

'legend.title\_fontsize': 12, # this needed to be changed

'text.usetex': False,

'font.sans-serif': ['DejaVu Sans', # this needed to be changed

'Bitstream Vera Sans',

'Computer Modern Sans Serif',

'Lucida Grande',

'Verdana',

'Geneva',

'Lucid',

'Arial',

'Helvetica',

'Avant Garde',

'sans-serif'],

'mathtext.bf': 'helvetica:bold',

'xtick.major.pad': 6,

'ytick.major.pad': 6,

'xtick.major.size': 5,

'ytick.major.size': 5,

'xtick.minor.size': 3, # minor tick size in points

'xtick.major.width': 1., # major tick width in points

'xtick.minor.width': 1., # minor tick width in points

'xtick.labelsize': 'small', # this needed to be added

'ytick.minor.size': 3, # minor tick size in points

'ytick.major.width': 1., # major tick width in points

'ytick.minor.width': 1., # minor tick width in points

'ytick.labelsize':'small' # this needed to be added

}

plt.rcParams.update(params)

def dynamics(x,t,alpha\_c=10.0, beta\_c=14.87, a\_c=-1.27, b\_c=-0.68):

'''

The system from question 3 of Assignment 1.

'''

xdot = np.zeros(3)

# we can't index a floating point value, so I created a list to take its place.

# this has been repeated for all the functions below.

x\_list = [int(e) for e in str(x) if e.isdigit()]

fx = b\_c\*x\_list[0] + 0.5\*(a\_c-b\_c)\*(np.abs(x\_list[0]+1.0)-np.abs(x\_list[0]-1.0))

# the construction of the state-dynamics vector

# again this is repeated for all the subsequent functions

xdot[0] = alpha\_c\*(x\_list[1]-x\_list[0]-fx)

xdot[1] = x\_list[0]-x\_list[1]+x\_list[2]

xdot[2] = -beta\_c\*x\_list[1]

return xdot

def dynamics\_question\_b(x,t,alpha\_c=10.0, beta\_c=14.87, a\_c=-1.27, b\_c=-0.68, sigma=0.03):

xdot = np.zeros(3)

xdot\_prime = np.zeros(3)

x\_list = [int(e) for e in str(x) if e.isdigit()]

x\_init\_prime = np.random.uniform(0.1,0.5,3)

x\_list\_prime = [int(e) for e in str(x\_init\_prime) if e.isdigit()]

fx = b\_c\*x\_list[0] + 0.5\*(a\_c-b\_c)\*(np.abs(x\_list[0]+1.0)-np.abs(x\_list[0]-1.0))

xdot[0] = alpha\_c\*(x\_list[1]-x\_list[0]-fx)

xdot[1] = x\_list[0] - x\_list[1] + x\_list[2]

xdot[2] = -beta\_c\*x\_list[1]

fx\_prime = b\_c\*x\_list\_prime[0] + 0.5\*(a\_c-b\_c)\*(np.abs(x\_list\_prime[0]+1.0)-np.abs(x\_list\_prime[0]-1.0))

xdot\_prime[0] = alpha\_c\*(x\_list\_prime[1]-x\_list\_prime[0]-fx\_prime) + sigma\*(x\_list[0] - x\_list\_prime[0]) # coupple to x

xdot\_prime[1] = x\_list\_prime[0] - x\_list\_prime[1] + x\_list\_prime[2]

xdot\_prime[2] = -beta\_c\*x\_list\_prime[1]

# define the following errors:

ex = xdot[0] - xdot\_prime[0]

ey = xdot[1] - xdot\_prime[1]

ez = xdot[2] - xdot\_prime[2]

# error dynamics:

ex\_dot = (-alpha\_c - alpha\_c\*(fx-fx\_prime) - 2\*sigma)\*ex + alpha\_c\*ey

ey\_dot = ex - ey + ez

ez\_dot = -beta\_c\*ey

e\_dot = [ex\_dot, ey\_dot, ez\_dot]

# return the error dynamics

return e\_dot

def dynamics\_question\_c\_yy(x,t,alpha\_c=10.0, beta\_c=14.87, a\_c=-1.27, b\_c=-0.68, sigma=0.03):

xdot = np.zeros(3)

xdot\_prime = np.zeros(3)

x\_list = [int(e) for e in str(x) if e.isdigit()]

x\_init\_prime = np.random.uniform(0.1,0.5,3)

x\_list\_prime = [int(e) for e in str(x\_init\_prime) if e.isdigit()]

fx = b\_c\*x\_list[0] + 0.5\*(a\_c-b\_c)\*(np.abs(x\_list[0]+1.0)-np.abs(x\_list[0]-1.0))

xdot[0] = alpha\_c\*(x\_list[1]-x\_list[0]-fx)

xdot[1] = x\_list[0] - x\_list[1] + x\_list[2]

xdot[2] = -beta\_c\*x\_list[1]

fx\_prime = b\_c\*x\_list\_prime[0] + 0.5\*(a\_c-b\_c)\*(np.abs(x\_list\_prime[0]+1.0)-np.abs(x\_list\_prime[0]-1.0))

xdot\_prime[0] = alpha\_c\*(x\_list\_prime[1]-x\_list\_prime[0]-fx\_prime)

xdot\_prime[1] = x\_list\_prime[0] - x\_list\_prime[1] + x\_list\_prime[2] + sigma\*(x\_list[1] - x\_list\_prime[1]) # couple to y

xdot\_prime[2] = -beta\_c\*x\_list\_prime[1]

# define the following errors:

ex = xdot[0] - xdot\_prime[0]

ey = xdot[1] - xdot\_prime[1]

ez = xdot[2] - xdot\_prime[2]

# error dynamics:

# TO DO

'''

ex\_dot = (-alpha\_c - alpha\_c\*(fx-fx\_prime) - 2\*sigma)\*ex + alpha\_c\*ey

ey\_dot = ex - ey + ez

ez\_dot = -beta\_c\*ey

e\_dot = [ex\_dot, ey\_dot, ez\_dot]

'''

# return the error dynamics

return e\_dot

def dynamics\_question\_c\_zz(x,t,alpha\_c=10.0, beta\_c=14.87, a\_c=-1.27, b\_c=-0.68, sigma=0.03):

xdot = np.zeros(3)

xdot\_prime = np.zeros(3)

x\_list = [int(e) for e in str(x) if e.isdigit()]

x\_init\_prime = np.random.uniform(0.1,0.5,3)

x\_list\_prime = [int(e) for e in str(x\_init\_prime) if e.isdigit()]

fx = b\_c\*x\_list[0] + 0.5\*(a\_c-b\_c)\*(np.abs(x\_list[0]+1.0)-np.abs(x\_list[0]-1.0))

xdot[0] = alpha\_c\*(x\_list[1]-x\_list[0]-fx)

xdot[1] = x\_list[0] - x\_list[1] + x\_list[2]

xdot[2] = -beta\_c\*x\_list[1]

fx\_prime = b\_c\*x\_list\_prime[0] + 0.5\*(a\_c-b\_c)\*(np.abs(x\_list\_prime[0]+1.0)-np.abs(x\_list\_prime[0]-1.0))

xdot\_prime[0] = alpha\_c\*(x\_list\_prime[1]-x\_list\_prime[0]-fx\_prime)

xdot\_prime[1] = x\_list\_prime[0] - x\_list\_prime[1] + x\_list\_prime[2]

xdot\_prime[2] = -beta\_c\*x\_list\_prime[1] + sigma\*(x\_list[2] - x\_list\_prime[2]) # couple to z

# define the following errors:

ex = xdot[0] - xdot\_prime[0]

ey = xdot[1] - xdot\_prime[1]

ez = xdot[2] - xdot\_prime[2]

# error dynamics:

# TO DO

'''

ex\_dot = (-alpha\_c - alpha\_c\*(fx-fx\_prime) - 2\*sigma)\*ex + alpha\_c\*ey

ey\_dot = ex - ey + ez

ez\_dot = -beta\_c\*ey

e\_dot = [ex\_dot, ey\_dot, ez\_dot]

'''

# return the error dynamics

return e\_dot

def dynamics\_question\_d\_xx(x,t,alpha\_c=10.0, beta\_c=14.87, a\_c=-1.27, b\_c=-0.68):

xdot = np.zeros(3)

x\_list = [int(e) for e in str(x) if e.isdigit()]

fx = b\_c\*x\_list[0] + 0.5\*(a\_c-b\_c)\*(np.abs(x\_list[0]+1.0)-np.abs(x\_list[0]-1.0))

xdot[0] = alpha\_c\*(x\_list[1]-x\_list[0]-fx)

xdot[1] = x\_list[0]-x\_list[1]+x\_list[2]

xdot[2] = -beta\_c\*x\_list[1]

return xdot

def dynamics\_question\_e\_x\_driver(x,t,alpha\_c=10.0, beta\_c=14.87, a\_c=-1.27, b\_c=-0.68, sigma=0.03):

'''

Pecora-Carrol Couppling for an x-x coupled system.

'''

xdot = np.zeros(3)

xdot\_prime = np.zeros(3)

x\_list = [int(e) for e in str(x) if e.isdigit()] # change

x\_init\_prime = np.random.uniform(0.1,0.5,3)

x\_list\_prime = [int(e) for e in str(x\_init\_prime) if e.isdigit()] # change

fx = b\_c\*x\_list[0] + 0.5\*(a\_c-b\_c)\*(np.abs(x\_list[0]+1.0)-np.abs(x\_list[0]-1.0))

xdot[0] = alpha\_c\*(x\_list[1]-x\_list[0]-fx)

xdot[1] = x\_list[0] - x\_list[1] + x\_list[2]

xdot[2] = -beta\_c\*x\_list[1]

# fx\_prime = b\_c\*x\_list\_prime[0] + 0.5\*(a\_c-b\_c)\*(np.abs(x\_list\_prime[0]+1.0)-np.abs(x\_list\_prime[0]-1.0))

# xdot\_prime[0] = alpha\_c\*(x\_list\_prime[1]-x\_list\_prime[0]-fx\_prime)

xdot\_prime[1] = x\_list\_prime[0] - x\_list\_prime[1] + x\_list\_prime[2]

xdot\_prime[2] = -beta\_c\*x\_list\_prime[1]

# define the following errors:

# ex = xdot[0] - xdot\_prime[0]

ey = xdot[1] - xdot\_prime[1]

ez = xdot[2] - xdot\_prime[2]

# error dynamics:

# ex\_dot = (-alpha\_c - alpha\_c\*(fx-fx\_prime) - 2\*sigma)\*ex + alpha\_c\*ey

ey\_dot = ex - ey + ez

ez\_dot = -beta\_c\*ey

# e\_dot = [ex\_dot, ey\_dot, ez\_dot]

e\_dot = [xdot[0], ey\_dot, ez\_dot]

# return the error dynamics

return e\_dot

def dynamics\_question\_e\_y\_driver(x,t,alpha\_c=10.0, beta\_c=14.87, a\_c=-1.27, b\_c=-0.68, sigma=0.03):

'''

Pecora-Carrol Coupling for an x-x coupled system.

'''

xdot = np.zeros(3)

xdot\_prime = np.zeros(3)

x\_list = [int(e) for e in str(x) if e.isdigit()] # change

x\_init\_prime = np.random.uniform(0.1,0.5,3)

x\_list\_prime = [int(e) for e in str(x\_init\_prime) if e.isdigit()] # change

fx = b\_c\*x\_list[0] + 0.5\*(a\_c-b\_c)\*(np.abs(x\_list[0]+1.0)-np.abs(x\_list[0]-1.0))

xdot[0] = alpha\_c\*(x\_list[1]-x\_list[0]-fx)

xdot[1] = x\_list[0] - x\_list[1] + x\_list[2]

xdot[2] = -beta\_c\*x\_list[1]

# fx\_prime = b\_c\*x\_list\_prime[0] + 0.5\*(a\_c-b\_c)\*(np.abs(x\_list\_prime[0]+1.0)-np.abs(x\_list\_prime[0]-1.0))

# xdot\_prime[0] = alpha\_c\*(x\_list\_prime[1]-x\_list\_prime[0]-fx\_prime)

xdot\_prime[1] = x\_list\_prime[0] - x\_list\_prime[1] + x\_list\_prime[2]

xdot\_prime[2] = -beta\_c\*x\_list\_prime[1]

# define the following errors:

ex = xdot[0] - xdot\_prime[0]

# ey = xdot[1] - xdot\_prime[1]

ez = xdot[2] - xdot\_prime[2]

# error dynamics:

ex\_dot = (-alpha\_c - alpha\_c\*(fx-fx\_prime) - 2\*sigma)\*ex + alpha\_c\*ey

# ey\_dot = ex - ey + ez

ez\_dot = -beta\_c\*ey

e\_dot = [ex\_dot, xdot[1], ez\_dot]

# return the error dynamics

return e\_dot

def dynamics\_question\_e\_z\_driver(x,t,alpha\_c=10.0, beta\_c=14.87, a\_c=-1.27, b\_c=-0.68, sigma=0.03):

'''

Pecora-Carrol Coupling for an x-x coupled system.

'''

xdot = np.zeros(3)

xdot\_prime = np.zeros(3)

x\_list = [int(e) for e in str(x) if e.isdigit()] # change

x\_init\_prime = np.random.uniform(0.1,0.5,3)

x\_list\_prime = [int(e) for e in str(x\_init\_prime) if e.isdigit()] # change

fx = b\_c\*x\_list[0] + 0.5\*(a\_c-b\_c)\*(np.abs(x\_list[0]+1.0)-np.abs(x\_list[0]-1.0))

xdot[0] = alpha\_c\*(x\_list[1]-x\_list[0]-fx)

xdot[1] = x\_list[0] - x\_list[1] + x\_list[2]

xdot[2] = -beta\_c\*x\_list[1]

# fx\_prime = b\_c\*x\_list\_prime[0] + 0.5\*(a\_c-b\_c)\*(np.abs(x\_list\_prime[0]+1.0)-np.abs(x\_list\_prime[0]-1.0))

# xdot\_prime[0] = alpha\_c\*(x\_list\_prime[1]-x\_list\_prime[0]-fx\_prime)

xdot\_prime[1] = x\_list\_prime[0] - x\_list\_prime[1] + x\_list\_prime[2]

xdot\_prime[2] = -beta\_c\*x\_list\_prime[1]

# define the following errors:

ex = xdot[0] - xdot\_prime[0]

ey = xdot[1] - xdot\_prime[1]

# ez = xdot[2] - xdot\_prime[2]

# error dynamics:

ex\_dot = (-alpha\_c - alpha\_c\*(fx-fx\_prime) - 2\*sigma)\*ex + alpha\_c\*ey

ey\_dot = ex - ey + ez

# ez\_dot = -beta\_c\*ey

e\_dot = [ex\_dot, ey\_dot, xdot[2]]

# return the error dynamics

return e\_dot

# x\_init = np.random.uniform(0.1,0.5,6)

x\_init = np.random.uniform(0.1,0.5,3)

# x\_init = np.random.uniform(0.1,0.5,3)

# x\_init\_prime = np.random.uniform(0.1,0.5,3)

# x\_inits = [x\_init, x\_init\_prime]

t\_init = 0; t\_final = 100; t\_step = 0.01

tpoints = np.arange(t\_init, t\_final, t\_step) # discrete time intervals at which to numerically integrate the systems

transient=int(0.8\*len(tpoints))

alpha\_c=10.0; beta\_c=14.87; a\_c=-1.27; b\_c=-0.68; sigma=0.03

# y = odeint(dynamics, x\_init, tpoints,args=(alpha\_c,beta\_c,a\_c,b\_c), hmax = 0.01)

# y = odeint(dynamics, x\_init, tpoints, args=(alpha\_c,beta\_c,a\_c,b\_c,sigma), full\_output = 1, hmax = 0.01)

# y = odeint(dynamics, x\_inits, tpoints, args=(alpha\_c,beta\_c,a\_c,b\_c,sigma), full\_output = 1, hmax = 0.01)

y = odeint(dynamics\_question\_b, x\_init, tpoints, args=(alpha\_c,beta\_c,a\_c,b\_c,sigma), full\_output = 1, hmax = 0.01) # the call to the code as is necessary for part b

'''

Returns y

array, shape (len(t), len(y0))

Array containing the value of y for each desired time in t, with the initial value y0 in the first row.

'''

'''

fig, axs = plt.subplots(3)

fig.suptitle('Vertically stacked subplots')

axs[0].plot(tpoints, np.array(y[0][0]),'k')

axs[1].plot(tpoints, np.array(y[0][1]),'k')

axs[2].plot(tpoints, np.array(y[0][2]),'k')

'''

plt.figure()

plt.plot(tpoints, np.array(y[0]),'k') # plot takes the time points

# plt.plot(y[:,0],y[:,1],'k')

# plt.plot(tpoints, y,'k')

plt.xlabel(r"$x$")

plt.ylabel(r"$y$")

plt.xlim([-4,4])

plt.ylim([-2,2])

plt.tight\_layout()

plt.show()