**Superconductivity python code**

import numpy as np

import matplotlib.pyplot as plt

# Parameters

M = 50 # Number of spatial sites

gamma = 1.0 # Damping coefficient

alpha = 5.0 # Feedback strength

beta = 2.0 # Magnetic field suppression coefficient

D = 0.1 # Diffusion coefficient (spatial coupling)

k\_B = 1.0 # Boltzmann constant

T\_init = 2.0 # Initial temperature

T\_final = 0.5 # Final temperature

B\_init = 0.0 # Initial magnetic field

B\_final = 1.0 # Final magnetic field

dt = 0.01 # Time step

steps = 200 # Time steps per temperature/magnetic field

temp\_steps = 50 # Number of temperature steps

noise\_std = 0.05 # Noise standard deviation

# Electron energies xi\_j (random)

np.random.seed(42)

xi = np.random.uniform(-1, 1, M)

# Pairing potential V\_j,k' (simplified as diagonal)

V0 = -0.5

V = V0 \* np.eye(M)

# Initialize gaps Delta\_j (complex)

Delta = 0.01 \* (np.random.randn(M) + 1j \* np.random.randn(M))

# Initialize emergent order parameter

E = 0.0

# Temperature and magnetic field schedules (linear)

temperatures = np.linspace(T\_init, T\_final, temp\_steps)

mag\_fields = np.linspace(B\_init, B\_final, temp\_steps)

# Storage for plotting

E\_history = []

Delta\_avg\_history = []

for T, B in zip(temperatures, mag\_fields):

for \_ in range(steps):

E\_j = np.sqrt(xi\*\*2 + np.abs(Delta)\*\*2)

tanh\_terms = np.tanh(E\_j / (2 \* k\_B \* T))

# Interaction term (simplified diagonal)

interaction = - V.diagonal() \* Delta / (2 \* E\_j) \* tanh\_terms

# Spatial diffusion (nearest neighbors with periodic BC)

diffusion = np.roll(Delta, -1) + np.roll(Delta, 1) - 2 \* Delta

# Noise term

noise = noise\_std \* (np.random.randn(M) + 1j \* np.random.randn(M))

# Downward causation feedback

feedback = alpha \* E \* Delta

# Magnetic field suppression

mag\_suppression = - beta \* (B\*\*2) \* Delta

# Update gaps

dDelta\_dt = -gamma \* Delta + D \* diffusion + interaction + feedback + mag\_suppression + noise

Delta += dDelta\_dt \* dt

# Update emergent order parameter

E += dt \* np.abs(np.mean(Delta))

# Store for plotting

E\_history.append(E)

Delta\_avg\_history.append(np.abs(np.mean(Delta)))

# Plotting results

time\_axis = np.arange(len(E\_history)) \* dt

plt.figure(figsize=(12,6))

plt.plot(time\_axis, E\_history, label='Emergent order parameter E(t)')

plt.plot(time\_axis, Delta\_avg\_history, label='Average gap magnitude |<Δ>|')

plt.xlabel('Time')

plt.ylabel('Magnitude')

plt.title('Emergence of Superconductivity with Spatial, Noise & Magnetic Field Effects')

plt.legend()

plt.grid(True)

plt.show()

**ANN Python code**

import numpy as np

import matplotlib.pyplot as plt

# Parameters

input\_size = 5

hidden\_size = 10

output\_size = 3

learning\_rate = 0.1

time\_steps = 200

batch\_size = 50

def sigmoid(x):

return 1 / (1 + np.exp(-x))

def sigmoid\_derivative(x):

s = sigmoid(x)

return s \* (1 - s)

# Initialize weights and biases

np.random.seed(0)

W1 = np.random.randn(hidden\_size, input\_size) \* 0.1

b1 = np.zeros((hidden\_size, 1))

W2 = np.random.randn(output\_size, hidden\_size) \* 0.1

b2 = np.zeros((output\_size, 1))

# Random input data and one-hot targets

X = np.random.rand(input\_size, batch\_size)

Y = np.zeros((output\_size, batch\_size))

for i in range(batch\_size):

Y[np.random.randint(0, output\_size), i] = 1

def activation\_entropy(activations):

entropies = []

for neuron\_acts in activations:

hist, \_ = np.histogram(neuron\_acts, bins=10, range=(0,1), density=True)

hist += 1e-12 # avoid log(0)

prob = hist / hist.sum()

entropy = -np.sum(prob \* np.log(prob))

entropies.append(entropy)

return np.mean(entropies)

E\_history = []

loss\_history = []

E\_integral = 0

for t in range(time\_steps):

# Forward pass

Z1 = np.dot(W1, X) + b1

A1 = sigmoid(Z1)

Z2 = np.dot(W2, A1) + b2

A2 = sigmoid(Z2)

# Loss (MSE)

loss = np.mean((A2 - Y) \*\* 2)

loss\_history.append(loss)

# Backpropagation

dZ2 = (A2 - Y) \* sigmoid\_derivative(Z2)

dW2 = np.dot(dZ2, A1.T) / batch\_size

db2 = np.sum(dZ2, axis=1, keepdims=True) / batch\_size

dA1 = np.dot(W2.T, dZ2)

dZ1 = dA1 \* sigmoid\_derivative(Z1)

dW1 = np.dot(dZ1, X.T) / batch\_size

db1 = np.sum(dZ1, axis=1, keepdims=True) / batch\_size

# Emergent property: average activation entropy

entropy = activation\_entropy(A1)

E\_integral += entropy

E\_history.append(E\_integral)

# Adaptive learning rate modulated by emergent property

adaptive\_lr = learning\_rate \* (1 + 0.5 \* np.tanh(E\_integral - 2))

# Update weights and biases

W2 -= adaptive\_lr \* dW2

b2 -= adaptive\_lr \* db2

W1 -= adaptive\_lr \* dW1

b1 -= adaptive\_lr \* db1

# Plotting

import matplotlib.pyplot as plt

plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)

plt.plot(loss\_history, label='Loss')

plt.xlabel('Time step')

plt.ylabel('Loss')

plt.title('Training Loss over Time')

plt.grid(True)

plt.legend()

plt.subplot(1, 2, 2)

plt.plot(E\_history, label='Integrated Activation Entropy (Emergence)')

plt.xlabel('Time step')

plt.ylabel('Emergent Property E(t)')

plt.title('Emergent Property over Time')

plt.grid(True)

plt.legend()

plt.tight\_layout()

plt.show()

**ECM Python code**

**import numpy as np**

**import matplotlib.pyplot as plt**

**# Parameters**

**r\_c = 0.5 # cell proliferation rate**

**K\_c = 100 # carrying capacity**

**alpha\_e = 0.01 # ECM enhancement on cell growth**

**delta\_c = 0.1 # cell death rate**

**r\_e = 0.3 # ECM production rate by cells**

**delta\_e = 0.05 # ECM degradation rate**

**beta = 0.02 # feedback from accumulated ECM**

**dt = 0.1**

**T = 100**

**steps = int(T / dt)**

**# Initialize variables**

**x\_c = np.zeros(steps)**

**x\_e = np.zeros(steps)**

**E = np.zeros(steps)**

**# Initial conditions**

**x\_c[0] = 10**

**x\_e[0] = 5**

**E[0] = x\_e[0] \* dt**

**for t in range(1, steps):**

**dx\_c = r\_c \* x\_c[t-1] \* (1 - x\_c[t-1]/K\_c) + alpha\_e \* x\_e[t-1] \* x\_c[t-1] - delta\_c \* x\_c[t-1]**

**dx\_e = r\_e \* x\_c[t-1] - delta\_e \* x\_e[t-1] + beta \* E[t-1]**

**x\_c[t] = x\_c[t-1] + dx\_c \* dt**

**x\_e[t] = x\_e[t-1] + dx\_e \* dt**

**E[t] = E[t-1] + x\_e[t] \* dt**

**# Plotting**

**time = np.linspace(0, T, steps)**

**plt.figure(figsize=(12, 6))**

**plt.plot(time, x\_c, label='Cell Density $x\_c(t)$')**

**plt.plot(time, x\_e, label='ECM Concentration/Stiffness $x\_e(t)$')**

**plt.plot(time, E, label='Accumulated ECM $E(t)$', linestyle='--')**

**plt.xlabel('Time')**

**plt.ylabel('Values')**

**plt.title('ECM and Cell Dynamics in Tissue Formation')**

**plt.legend()**

**plt.grid(True)**

**plt.show()**