```
import numpy as np
import matplotlib.pyplot as plt
from scipy.linalg import expm

# Constants
h = 6.62607015e-34
Q = 21
hbar = h/(2*np.pi)

# Define Time Interval (Four Floquet Steps)
tao = 1.4e-6
t1 = 2e-9
t2 = 2e-9

# Some constants define by random sampling
# Bz(# of spin)
Bz_sigma = 18e6
Bz_mu = np.array([0,20e6,0,50e6]) + 3.075e9

# Function for generating coupling strength J (randomly sampled + include
 fractional error)
# Range for J
J2min = 0.05e6
J2max = 5e6
epsilonmin = -0.6
epsilonmax = 0.6
J2s = np.linspace(J2min,J2max,200)
epsilons = np.linspace(epsilonmin,epsilonmax,200)

def J(epsilon):
    J1 = (np.random.normal(250e6,2679512.846))*(1+epsilon)
    J3 = (np.random.normal(250e6,2679512.846))*(1+epsilon)
    return J1,J3

# Constants for ST1
fg1 = 0.9
tm1 = 4e-6
T1_1 = 60e-6
fm1 = 0.99

# Constants for ST2
fg2 = 0.95
tm2 = 6e-6
T1_2 = 50e-6
fm2 = 0.95

# Basic States for composite ST qubits
up = np.array([1, 0])
down = np.array([0, 1])
g_state = np.kron(up, down)
e_state = np.kron(down, up)
```

```
Tplus = np.kron(up, up)
Tminus = np.kron(down, down)

# Define Matrices needed for Hamiltonian of the system
# Identity matrix
I2 = np.identity(2)
I4 = np.identity(4)

# Pauli matrices
sigma_x = np.matrix([[0, 1], [1, 0]])
sigma_y = np.matrix([[0, -1j], [1j, 0]])
sigma_z = np.matrix([[1, 0], [0, -1]])
sigma = np.array([sigma_x, sigma_y, sigma_z])

# Define Pauli matrices for each spin
sigma_1 = np.kron(np.kron(np.kron(sigma, I2), I2), I2)
sigma_2 = np.kron(np.kron(np.kron(I2, sigma), I2), I2)
sigma_3 = np.kron(np.kron(np.kron(I2, I2), sigma), I2)
sigma_4 = np.kron(np.kron(np.kron(I2, I2), I2), sigma)
sigma = np.array([sigma_1, sigma_2, sigma_3, sigma_4])

# function prepare initial state for ST qubits
def initial_state_ST(fg,tm,T1,fm):
    s1 = np.sqrt(fg)
    s2 = np.sqrt((1-fg)/3)
    s3 = np.sqrt((1-fg)/3)
    s4 = np.sqrt((1-fg)/3)
    return s1*g_state + s2*e_state + s3*Tplus + s4*Tminus

# Define ST 1 & ST 2
ST1_ini = initial_state_ST(fg1,tm1,T1_1,fm1)
ST2_ini = initial_state_ST(fg2,tm2,T1_2,fm2)

# Define initial state of the system
int_state = np.kron(ST1_ini, ST2_ini)

# Measurement Operator Mg
Pg = np.outer(g_state,g_state)
Mg1 = np.kron(Pg,I4)
Mg2 = np.kron(I4,Pg)

def final_return_probability(Mg,final_state,tm,T1,fm):
    temp = np.vdot(final_state,Mg@final_state)
    Pg_true = (np.abs(temp))**2
    r = 1 - np.exp(-tm/T1)
    q = 1 - fm
    return (1-r-2*q)*Pg_true + r + q

# Define the Hamiltonian of the system
# Define function for Hamiltonian of spin in B field (non-interacting)
def spin_in_B_field():
```

```
    H = np.zeros((16, 16),dtype=complex)
    Bz = np.random.normal(Bz_mu,Bz_sigma)
    for i in range(4):
        H = H + Bz[i]*sigma[i,2]
    H = (h/2)*H
    return H

# Define interacting Hamiltonian
def interacting_Hamiltonian(J,sigma_1st,sigma_2nd):
    return (h/4)*J*(sigma_1st[0]@sigma_2nd[0] + sigma_1st[1]@sigma_2nd[1] +
     sigma_1st[2]@sigma_2nd[2])

# Define Sint Hamiltonian
def S_int_Hamiltonian(J2):
    H = spin_in_B_field()
    H = H + interacting_Hamiltonian(J2,sigma_2,sigma_3)
    H = (-1j*tao/hbar)*H
    S_int = expm(H)
    return S_int

# Define S2 Hamiltonian
def S2_Hamiltonian(J3):
    H = spin_in_B_field()
    H = H + interacting_Hamiltonian(J3,sigma_3,sigma_4)
    H = (-1j*t2/hbar)*H
    S2 = expm(H)
    return S2

# Define S1 Hamiltonian
def S1_Hamiltonian(J1):
    H = spin_in_B_field()
    H = H + interacting_Hamiltonian(J1,sigma_1,sigma_2)
    H = (-1j*t1/hbar)*H
    S1 = expm(H)
    return S1

def non_time_floquet_operator(J1,J2,J3):
    return S_int_Hamiltonian(J2)@S2_Hamiltonian(J3)@S1_Hamiltonian(J1)

# Loop over for calculating probability
ProbST1 = np.zeros((len(J2s),len(epsilons)))
ProbST2 = np.zeros((len(J2s),len(epsilons)))
Num = 128
for k in range(Num):
    ProbabilitiesST1 = np.zeros((len(J2s),len(epsilons)))
    ProbabilitiesST2 = np.zeros((len(J2s),len(epsilons)))
    for i , J2 in enumerate(J2s):
        for j , epsilon in enumerate(epsilons):
            state = int_state
            J1,J3 = J(epsilon)
            U = non_time_floquet_operator(J1,J2,J3)
```

```
            state = U@U@U@U@state
            final_state = state
            ProbabilitiesST1[i,j] =
             final_return_probability(Mg1,final_state,tm1,T1_1,fm1)
            ProbabilitiesST2[i,j] =
             final_return_probability(Mg2,final_state,tm2,T1_2,fm2)
    ProbST1 += ProbabilitiesST1.T
    ProbST2 += ProbabilitiesST2.T
ProbST1 = ProbST1/Num
ProbST2 = ProbST2/Num

# Plotting
plt.figure()
plt.imshow(ProbST1, extent=[J2min, J2max, epsilonmin, epsilonmax],
 aspect='auto', cmap='viridis')
cbar = plt.colorbar()
cbar.set_label(r'$P_g^1$',loc='top')
plt.xlabel(r'$J_2 (MHz)$')
plt.ylabel(r'$\epsilon$')
plt.title('Return Probability of ST1')
plt.show()
plt.savefig('fig2c.png')

# Plotting
plt.figure()
plt.imshow(ProbST2, extent=[J2min, J2max, epsilonmin, epsilonmax],
 aspect='auto', cmap='viridis')
cbar = plt.colorbar()
cbar.set_label(r'$P_g^2$',loc='top')
plt.xlabel(r'$J_2 (MHz)$')
plt.ylabel(r'$\epsilon$')
plt.title('Return Probability of ST2')
plt.show()
plt.savefig('fig2d.png')
```