0. README

This python notebook can be used for processing data and generating result figures in the main and supplementary manuscript. The python note book has 3 section.

The first section imports all relevant python packages and defines functions + key variables to be used in the subsequence sections. The second and the third section deal with synthetic data (for figure 2) and experimental data (for figure 3), respectively.

The second section can be used for generating synthetic data (from a user defined kinetic parameter set), use MCMC (Metropolis') for estimating posterior distributions of parameter sets, and finally use these posterior distributions to simulate timecourse dynamics of cell populations and analyzing certainty of each parameter values (which we can later compared to the actual parameter values used for generating synthetic data).

The third section can be used for processing experimental data to have the same format as synthetic data above, estimate posterior distribution of parameters with MCMC (Metropolis'), and finally use these posterior distributions to simulate timecourse dynamics of cell populations and analyzing certainty of each parameter values.

1. Define Functions, Key Variables, Packages

```
# Importing required packages
import numpy as np
import pandas as pd
from scipy.integrate import odeint
import matplotlib.pyplot as plt
import time
import seaborn as sns
from scipy.signal import find_peaks
```

ChatGPT

```
## define ODEs for the Model of conjugation system
def conjugationODEs(Y, t, eta, mu, mu13, mu61, mu1361, kappa13, kappa61, D, Nm):
    DH5a, DH5a13, DH5a61, DH5a1361 = Y
    dDH5adt = -eta * DH5a * DH5a61 - eta * DH5a * DH5a1361 + kappa13 * DH5a13 + kappa61 * DH5a61 - D * DH5a + mu * DH5a * (1
    dDH5a13dt = -eta * DH5a13 * DH5a61 - eta * DH5a13 * DH5a1361 + kappa61 * DH5a1361 - kappa13 * DH5a13 - D * DH5a13 + mu13
    dDH5a61dt = eta * DH5a * DH5a61 + eta * DH5a * DH5a1361 + kappa13 * DH5a1361 - kappa61 * DH5a61 - D * DH5a61 + mu61 * DH
    dDH5a1361dt = eta * DH5a13 * DH5a61 + eta * DH5a13 * DH5a1361 - kappa13 * DH5a1361 - kappa61 * DH5a1361 - D * DH5a1361 +

    return [dDH5adt, dDH5a13dt, dDH5a61dt, dDH5a1361dt]

### This four dimensional ODEs model the dynamics of four bacteria subpopulation in our systems: DH5a, DH5a-X13, DH5a-X61 ar
##  eta = conjugation efficiency of X61
##  mu, mu13, mu61, mu1361 = growth rate of DH5a, DH5a-X13, DH5a-X61, and DH5a-X13-X61, respectively
##  kappa13, kappa61 = loss rate of X13 and X61, respectively
##  D = death or dilution rate of cell
##  Nm = carrying capacity of the population (all subpopulations combined)
```

ChatGPT

```
# define a function for generating synthetic data from give
## initial condition (initCond), time points (timeSim), number of expermetn repeats (repeatNum), measurement noise (sdNoise)
## we can also specify which subpopulation to be shown (outputShown) and the name of output csv file (saveName)

def makeSynthData(initCond, timeSim, repeatNum, sdNoise, outputShown, saveName):

  # Simulating the system
  solution = odeint(
    conjugationODEs,
    initCond,
    timeSim,
    args=(eta_value, mu_value, mu13_value, mu61_value, mu1361_value, kappa13_value, kappa61_value, D_value, Nm_value)
  )

  # Extract the species values at the specified dataTime points
  selected_solution = solution

  # Add noise to synthetic data
  dataTime_rep = np.tile(timeSim,(1,repeatNum))[0]
  dataSol_rep = np.tile(selected_solution,(repeatNum,1))
  randExp_rep = np.exp(np.random.normal(0, sdNoise, (dataSol_rep.shape)))
  simData_rep = dataSol_rep*randExp_rep

  # minimum CFU should not be lower than 1
  simData_rep = np.maximum(np.ones(simData_rep.shape), simData_rep)
```

```
  # Creating the DataFrame for simulated data
  df_simulated_data = pd.DataFrame({
      'Time (Hr)': dataTime_rep,
      'DH5a': simData_rep[:, 0],
      'DH5a-X13': simData_rep[:, 1],
      'DH5a-X61': simData_rep[:, 2],
      'DH5a-X13-X61': simData_rep[:, 3]
  })

  # save synthetic data to csv
  df_simulated_data.to_csv(saveName, encoding='utf-8', index=False)

  labels = ['DH5a', 'DH5a-X13', 'DH5a-X61', 'DH5a-X13', 'DH5a-X13-X61']
  colors = ['black', 'red', 'green', 'goldenrod']
  sS = [50, 200, 200, 200]
  alphas = [1, 0.5, 0.5, 0.5]

  # Plotting synthetic data using scatter plots
  plt.figure(figsize=(14, 8))
  for k in outputShown:
    plt.scatter(dataTime_rep, simData_rep[:, k], label=labels[k], color=colors[k], s = sS[k], alpha = alphas[k])

  plt.xlabel('Time (Hr)', fontsize = 15)
  plt.ylabel('CFU', fontsize = 15)
  plt.xticks(fontsize=12)
  plt.yticks(fontsize=12)
  plt.yscale('log')
  plt.grid(True)
  plt.legend()
  plt.show()


# These functions can collectively be used for applying MCMC (Metropolis) to
# estimate posterior distributions of parameters. Note that posterior values here
# are all shown in log scale. Thus,
## log(posterior) = log(prior) + log(likelihood)
## log(posterior) is always negative


# T variable used for adjusting noise level in Metropolis MCMC
T = 10


# define prior distribution of parameters.
# here we simply state that no kinetic parameter can have negative value
# (otherwise, we will set prior -inf which will also make posterior -inf)
# when all parameters are zero or positive, prior is set to 0 so it has no
# contribution of posterios, only likelihood doese

def prior(params):
    # Uniform prior
    for param in params:
        if param <= 0:
            return -np.inf
    return 0


# this likelihood function diffence between model_solution (generated from
# a given parameter set) and  actual data. The larger the difference, the
# less likely this paramter set is. 'sel' allow user to choose which subpopultion
# to be used in calculation (for example when we do not have CFU for certain subpopulation)

def likelihood(model_solution, data, sel):
    # Assuming a Gaussian likelihood for simplicity
    dataSel = data[:, sel]
    model_solutionSel = model_solution[:, sel]
    ssrlog = -0.5 * np.sum((np.log(dataSel) - np.log(model_solutionSel))**2) ## use log difference of species density, inclu
    return ssrlog

# Define a function to simulate timecourse dynamics using a set of parameter values ... with initial condition
def simulate_timecourse(params, model, init, times):
    unique_times = np.unique(times)
    full_params = np.concatenate((params, [Nm_value]))
    model_solution = odeint(
        model,
        init,
        unique_times,
        args=tuple(full_params)
    )

    model_solution_rep = np.array([model_solution[np.where(unique_times == time)[0][0]] for time in times]) + 1 # add 1 to a
    return model_solution_rep
```

```
# Calculate posterior probability of a give parameter set and setup including model function, initial condition, data set ar
def posterior(params, modelInitDataTimes):
    posterior = prior(params)
    for model, init, data, times, sel in modelInitDataTimes:
      model_solution = simulate_timecourse(params, model, init, times)
      # Handle cases where the simulation failed to produce valid results (e.g., due to extreme parameter values)
      if np.any(np.isnan(model_solution)):
        print("**** NaN Posterior!!!")
        return -np.inf
      posterior = posterior + likelihood(model_solution, data, sel)
    return posterior

# Metropolis MCMC function with up to four input arguments
# args[0] initial parameter values, i.e., initial location of Metropolis random walk landscape
# args[1] proposal_fraction, i.e., step size of Metropolis random walk
# args[2] iterations, i.e., number of steps to take for Metropolis random walk
# args[3:] is a list of information bout model and data.
##  This list contains any number of lists; each list has form: (conjugationODEs, initial_conditions_i, data, times, noDH5a)
## conjugationODEs is an ODEs used for the model
## initial_conditions_i is an initial condition used for the model
## data is an experimental data
## noDH5a specifies the index of bacteria species to be used for Metropolis random walk.

def metropolis(*args):
    starting_params = args[0]
    proposal_fraction = args[1]
    iterations = args[2]
    modelInitDataTimes = args[3:]

    current_params = starting_params
    current_posterior = posterior(current_params, modelInitDataTimes)
    accepted = [] # a list to store the values of parameter set for each step of Metropolis random walk
    print(range(iterations))
    for i in range(iterations):
        # Scaled proposal step
        ## proposed_params = current_params + proposal_fraction * np.array(current_params) * np.random.randn(len(current_par
        proposed_params =  np.array(current_params)*np.exp(proposal_fraction *np.random.randn(len(current_params)))  # log s

        proposed_posterior = posterior(proposed_params, modelInitDataTimes)
        alpha = proposed_posterior - current_posterior # compare posterior probability of the current and proposed parameter
        print(100*i/iterations)

        # since posterior probability is measured is log scale. alpha is also in a log scale
        # if proposed_posterior is higher than current posteror alpha is +
        # if proposed_posterior is lower than current posteror alpha is -
        # np.random.rand() give a number between 0, 1 drawn from uniform distribution
        # thus, np.log(np.random.rand()) is always -
        # if alpha is + --> always accepted proposed_params
        # if alpha is - --> accept proposed_params probabilistically; the closer alpha is to 0, the more likely it proposed_
        # T tune the likeliness to accept; higher T, more likely to accept (pushing the right of > toward more negative)

        if alpha > (np.log(np.random.rand())*T):
            current_params = proposed_params
            current_posterior = proposed_posterior
        accepted.append([i, current_posterior] + list(current_params)) # record iteation index, posterior, param
        print("current posterior: " + str(current_posterior))
    return accepted



# Generate random starting parameters from a log-uniform distribution
def generate_random_parameters(min_values, max_values):
    log_min_values = np.log(min_values)
    log_max_values = np.log(max_values)
    random_parameters = np.exp(np.random.uniform(log_min_values, log_max_values))
    return random_parameters




# this function can be used for plotting the value of each parameter
# from each step of Metropolis random walk (data stored in .csv file)
def plot_multiple_files_updated(filenames, correct_values):
    fig, axes = plt.subplots(5, 1, figsize=(8, 14))

    for filename in filenames:
        df_ensemble = pd.read_csv(filename)

        # Plot Eta on the top subplot
        axes[0].plot(df_ensemble['Iteration'], df_ensemble['Eta'], color='blue', alpha = 0.5)
        # Removed x-axis label for top subplot
```

```
        axes[0].set_ylabel('Eta', fontsize = 18)
        axes[0].set_yscale('log')
        # Removed title for top subplot
        axes[0].grid(True)
        axes[0].text(df_ensemble['Iteration'].max() * 1.1, correct_values[0], f'{correct_values[0]:.2e}', va='center', color
        axes[0].text(df_ensemble['Iteration'].max() * 1.1, correct_values[0]*10000, 'Actual Values', va='center', color = 'b
        # Remove x-axis ticks for the top subplot
        axes[0].tick_params(axis='x', which='both', bottom=False, top=False, labelbottom=False)
        axes[0].tick_params(axis='y', labelsize=14)

        # Plot Mu, Mu13, Mu61, Mu1361 on the second subplot
        for idx, (param, color) in enumerate(zip(['Mu', 'Mu13', 'Mu61', 'Mu1361'], ['red', 'green', 'black', 'purple'])):
            axes[1].plot(df_ensemble['Iteration'], df_ensemble[param], label=f'{param} ({filename})', color=color, alpha = 0
            axes[1].text(df_ensemble['Iteration'].max() * 1.1, correct_values[idx+1], f'{correct_values[idx+1]:.2e}', va='ce

        # Removed x-axis label for second subplot
        axes[1].set_ylabel('Mu*', fontsize = 18)
        axes[1].set_yscale('log')
        # Removed title for second subplot
        axes[1].grid(True)
        # Remove x-axis ticks for the second subplot
        axes[1].tick_params(axis='x', which='both', bottom=False, top=False, labelbottom=False)
        axes[1].tick_params(axis='y', labelsize=14)

        # Plot Kappa13, Kappa61, on the third subplot
        for idx, (param, color) in enumerate(zip(['Kappa13', 'Kappa61'], ['red', 'green'])):
            axes[2].plot(df_ensemble['Iteration'], df_ensemble[param], label=f'{param} ({filename})', color=color, alpha = 0
            axes[2].text(df_ensemble['Iteration'].max() * 1.1, correct_values[idx+5], f'{correct_values[idx+5]:.2e}', va='ce

        # Removed x-axis label for third subplot
        axes[2].set_ylabel('Kappa*', fontsize = 18)
        axes[2].set_yscale('log')
        # Removed title for third subplot
        axes[2].grid(True)
        # Remove x-axis ticks for the third subplot
        axes[2].tick_params(axis='x', which='both', bottom=False, top=False, labelbottom=False)
        axes[2].tick_params(axis='y', labelsize=14)

        # Plot D, on the forth subplot
        for param, color in zip(['D'], ['blue']):
            axes[3].plot(df_ensemble['Iteration'], df_ensemble[param], color=color, alpha = 0.3)
        # Removed x-axis label for forth subplot
        axes[3].set_ylabel('D', fontsize = 18)
        axes[3].set_yscale('log')
        # Removed title for forth subplot
        axes[3].grid(True)
        axes[3].text(df_ensemble['Iteration'].max() * 1.1, correct_values[-1], f'{correct_values[-1]:.2e}', va='center', col
        # Remove x-axis ticks for the forth subplot
        axes[3].tick_params(axis='x', which='both', bottom=False, top=False, labelbottom=False)
        axes[3].tick_params(axis='y', labelsize=14)


        # plot posterior on the bottom subplot
        axes[4].plot(df_ensemble['Iteration'], df_ensemble['Posterior'], color='blue', alpha = 0.3)
        axes[4].set_ylabel('posterior', fontsize = 18)
        axes[4].tick_params(axis='x', labelsize=14)
        axes[4].tick_params(axis='y', labelsize=14)



def trim_and_concat_csv(file_list):
    """
    This function takes a list of CSV file names, trims the first 25% of rows from each,
    concatenates them into a single dataframe, and returns the resulting dataframe.

    :param file_list: List of strings, where each string is a CSV file name.
    :return: A single pandas dataframe containing the concatenated data.
    """
    trimmed_dataframes = []  # to store the trimmed dataframes

    for file_name in file_list:
        # Read the CSV file into a pandas dataframe
        df = pd.read_csv(file_name)

        # Calculate the number of rows to trim
        rows_to_trim = len(df) // 4

        # Trim the first 25% of rows
        trimmed_df = df.iloc[rows_to_trim:]

        # Append the trimmed dataframe to the list
        trimmed_dataframes.append(trimmed_df)
```

```
    # Concatenate all the trimmed dataframes
    concatenated_df = pd.concat(trimmed_dataframes, ignore_index=True)

    return concatenated_df

# Example usage:
# Assuming you have CSV files named 'file1.csv', 'file2.csv', etc. in the current directory
# concatenated_df = trim_and_concat_csv(['file1.csv', 'file2.csv'])
# print(concatenated_df)


# from a given set of parameter ensemble (after trim and concatenation),
# randomly select a certain number of parameter sets.

def selParam(ensembleSet, selNum):
  params_subset = ensembleSet.iloc[:, 2:].values  # post burnt ensemble

  # get the best parameter (max posterior)
  params_Best = ensembleSet.iloc[ensembleSet['Posterior'].idxmax(), 2:].values
  ##formatted_array = ["{:.1e}".format(x) for x in params_Best]

  # Randomly selecting a certain number (selNum) parameter sets from params_subset
  np.random.seed(0)  # Setting a seed for reproducibility
  random_indices = np.random.choice(len(params_subset), selNum, replace=False)
  selected_params = params_subset[random_indices]

  return params_Best, selected_params


# from a given set of parameter ensembles (after using selParam) and initital conidition (initCond),
# run a simulation and show the plowt together with dataset
# dataShown and colorShown specificy which cell type data to be shown and what colors to show them

def showDataFit(selected_params, initCond, Tmax, dataName, dataShown, colorShown):
  finer_dataTime = np.arange(0, Tmax, 1)

  finer_simulations = [simulate_timecourse(params, conjugationODEs, initCond, finer_dataTime) for params in selected_params]
  finer_simulations_array = np.array(finer_simulations)
  finer_mean_simulation =  np.exp(np.mean(np.log(finer_simulations_array), axis =0))+1 # geometric mean
  #finer_best_simulation = simulate_timecourse(params_Best, conjugationODEs, initCond, finer_dataTime)
  #finer_best_simulation_array = np.array(finer_best_simulation)

  finer_lower_bound = np.percentile(finer_simulations_array, 2.5, axis=0)+1
  finer_upper_bound = np.percentile(finer_simulations_array, 97.5, axis=0)+1

  df_simulated_data = pd.read_csv(dataName)

  speciesInd = {
      "DH5a":0,
      "DH5a-X13":1,
      "DH5a-X61":2,
      "DH5a-X13-X61":3
  }

  plt.figure(figsize=(7, 5))
  for i, (species, color) in enumerate(zip(dataShown, colorShown)):
    plt.plot(finer_dataTime, finer_mean_simulation[:, speciesInd[species]], color=color)
    plt.fill_between(finer_dataTime, finer_lower_bound[:, speciesInd[species]], finer_upper_bound[:, speciesInd[species]], c
    plt.scatter(df_simulated_data['Time (Hr)'], df_simulated_data[species], label=f'{species} Data', color=color, s=50)
  plt.xlabel('Time (Hr)', fontsize = 20)
  plt.ylabel('CFU', fontsize = 20)
  plt.yscale('log')
  plt.grid(True)
  plt.ylim([1E-1, 1E+10])
  plt.xticks(fontsize=18)
  plt.yticks(fontsize=18)


# similar to showDataFit but without showing shaded area

def showDataFit_NS(selected_params, initCond, Tmax, dataName, dataShown, colorShown):
  finer_dataTime = np.arange(0, Tmax, 1)

  finer_simulations = [simulate_timecourse(params, conjugationODEs, initCond, finer_dataTime) for params in selected_params]
  finer_simulations_array = np.array(finer_simulations)
  finer_mean_simulation =  np.exp(np.mean(np.log(finer_simulations_array), axis =0))+1 # geometric mean
  #finer_best_simulation = simulate_timecourse(params_Best, conjugationODEs, initCond, finer_dataTime)
  #finer_best_simulation_array = np.array(finer_best_simulation)

  #finer_lower_bound = np.percentile(finer_simulations_array, 2.5, axis=0)+1
  #finer_upper_bound = np.percentile(finer_simulations_array, 97.5, axis=0)+1
```

```
  df_simulated_data = pd.read_csv(dataName)

  speciesInd = {
      "DH5a":0,
      "DH5a-X13":1,
      "DH5a-X61":2,
      "DH5a-X13-X61":3
  }

  plt.figure(figsize=(7, 5))
  for i, (species, color) in enumerate(zip(dataShown, colorShown)):
    plt.plot(finer_dataTime, finer_mean_simulation[:, speciesInd[species]], color=color)
    #plt.fill_between(finer_dataTime, finer_lower_bound[:, speciesInd[species]], finer_upper_bound[:, speciesInd[species]],
    plt.scatter(df_simulated_data['Time (Hr)'], df_simulated_data[species], label=f'{species} Data', color=color, s=50)
  plt.xlabel('Time (Hr)', fontsize = 20)
  plt.ylabel('CFU', fontsize = 20)
  plt.yscale('log')
  plt.grid(True)
  plt.ylim([1E-1, 1E+10])
  plt.xticks(fontsize=18)
  plt.yticks(fontsize=18)


# similart to showDataFit but for growth
def showDataFitGrowth(selected_params, AinitCond, Tmax, AdataName, AdataShown, AcolorShown):
  finer_dataTime = np.arange(0, Tmax, 1)

  speciesInd = {
      "DH5a":0,
      "DH5a-X13":1,
      "DH5a-X61":2,
      "DH5a-X13-X61":3
  }

  plt.figure(figsize=(7, 5))

  for k in range(0, len(AinitCond)):
    initCond = AinitCond[k]
    dataName = AdataName[k]
    species = AdataShown[k]
    color = AcolorShown[k]

    finer_simulations = [simulate_timecourse(params, conjugationODEs, initCond, finer_dataTime) for params in selected_param
    finer_simulations_array = np.array(finer_simulations)
    finer_mean_simulation =  np.exp(np.mean(np.log(finer_simulations_array), axis =0))+1 # geometric mean
    #finer_best_simulation = simulate_timecourse(params_Best, conjugationODEs, initCond, finer_dataTime)
    #finer_best_simulation_array = np.array(finer_best_simulation)

    finer_lower_bound = np.percentile(finer_simulations_array, 2.5, axis=0)+1
    finer_upper_bound = np.percentile(finer_simulations_array, 97.5, axis=0)+1

    df_simulated_data = pd.read_csv(dataName)

    plt.plot(finer_dataTime, finer_mean_simulation[:, speciesInd[species]], color=color)
    plt.fill_between(finer_dataTime, finer_lower_bound[:, speciesInd[species]], finer_upper_bound[:, speciesInd[species]], c
    plt.scatter(df_simulated_data['Time (Hr)'], df_simulated_data[species], label=f'{species} Data', color=color, s=50)

  plt.xlabel('Time (Hr)', fontsize = 20)
  plt.ylabel('CFU', fontsize = 20)
  plt.yscale('log')
  plt.grid(True)
  plt.ylim([1E+5, 1E+10])
  plt.xticks(fontsize=18)
  plt.yticks(fontsize=18)

                                                                ChatGPT
# similart to showDataFitGrowth but without showing shaded area
def showDataFitGrowth_NS(selected_params, AinitCond, Tmax, AdataName, AdataShown, AcolorShown):
  finer_dataTime = np.arange(0, Tmax, 1)

  speciesInd = {
      "DH5a":0,
      "DH5a-X13":1,
      "DH5a-X61":2,
      "DH5a-X13-X61":3
  }

  plt.figure(figsize=(7, 5))

  for k in range(0, len(AinitCond)):
    initCond = AinitCond[k]
    dataName = AdataName[k]
    species = AdataShown[k]
```

```
    color = AcolorShown[k]

    finer_simulations = [simulate_timecourse(params, conjugationODEs, initCond, finer_dataTime) for params in selected_param
    finer_simulations_array = np.array(finer_simulations)
    finer_mean_simulation =  np.exp(np.mean(np.log(finer_simulations_array), axis =0))+1 # geometric mean
    #finer_best_simulation = simulate_timecourse(params_Best, conjugationODEs, initCond, finer_dataTime)
    #finer_best_simulation_array = np.array(finer_best_simulation)

    #finer_lower_bound = np.percentile(finer_simulations_array, 2.5, axis=0)+1
    #finer_upper_bound = np.percentile(finer_simulations_array, 97.5, axis=0)+1

    df_simulated_data = pd.read_csv(dataName)

    plt.plot(finer_dataTime, finer_mean_simulation[:, speciesInd[species]], color=color)
    #plt.fill_between(finer_dataTime, finer_lower_bound[:, speciesInd[species]], finer_upper_bound[:, speciesInd[species]],
    plt.scatter(df_simulated_data['Time (Hr)'], df_simulated_data[species], label=f'{species} Data', color=color, s=50)

  plt.xlabel('Time (Hr)', fontsize = 20)
  plt.ylabel('CFU', fontsize = 20)
  plt.yscale('log')
  plt.grid(True)
  plt.ylim([1E+5, 1E+10])
  plt.xticks(fontsize=18)
  plt.yticks(fontsize=18)


# show correlation and distribution of each parameters from ensemble we got from Metropolis random walk

#correctParams = [eta_value, mu_value, mu13_value, mu61_value, mu1361_value, kappa13_value, kappa61_value, D_value]

def process_and_plot_csv(CSVlist, burnN, fracSample, correctParams, xrange=None):
    all_dfs = []

    # Read each CSV file, remove the first burnN rows and append to the list
    for csv_file in CSVlist:
        try:
            df = pd.read_csv(csv_file)
            trimmed_df = df.iloc[burnN:]
            all_dfs.append(trimmed_df)
        except Exception as e:
            print(f"An error occurred with file {csv_file}: {e}")


    greek_letters = [r'$\eta$', r'$\mu$', r'$\mu_{13}$', r'$\mu_{61}$',
                 r'$\mu_{1361}$', r'$\kappa_{13}$', r'$\kappa_{61}$', r'$D$']


    # Concatenate all dataframes into one
    allParamSets = pd.concat(all_dfs, ignore_index=True)
    print(allParamSets.shape)

    # Create logAllParamSets from the log10 transformed values of specified columns
    columns_to_transform = ['Eta', 'Mu', 'Mu13', 'Mu61', 'Mu1361', 'Kappa13', 'Kappa61', 'D']
    logAllParamSets = allParamSets[columns_to_transform].apply(np.log10)

    # Randomly select fracSample percent of the rows
    logAllParamSets_sampled = logAllParamSets.sample(frac=fracSample)
    print(logAllParamSets_sampled.shape)

    # Plot 1: Correlation matrix using Spearman method
    correlation_matrix = logAllParamSets_sampled.corr(method='spearman')
    plt.figure(figsize=(10, 8))
    sns.heatmap(correlation_matrix, annot=True, fmt=".2f", cmap='coolwarm', square=True, cbar_kws={"shrink": .5})
    plt.xticks(ticks = np.arange(0.5, 8.5, 1), labels = greek_letters, fontsize = 22)
    plt.yticks(ticks = np.arange(0.5, 8.5, 1), labels = greek_letters, fontsize = 22)
    plt.title('Spearman Correlation Matrix', fontsize=16)
    plt.show()

    # Plot 2: Histograms of the columns
    fig, axes = plt.subplots(nrows=2, ncols=4, figsize=(20, 10))  # Adjust the grid size depending on the number of columns
    axes = axes.flatten()  # Flatten the array of axes for easy iteration
    ymax = [600,1400, 1400, 1400, 1400, 1400, 1400, 1000]
    ymax = [0.7, 0.7, 0.7, 0.7, 0.7, 0.7, 0.7, 0.7]
    for i, col in enumerate(columns_to_transform):
        #ax = sns.histplot(logAllParamSets_sampled[col], ax=axes[i], kde=True, bins = np.arange(xrange[i][0], xrange[i][1],
        ax = sns.histplot(logAllParamSets_sampled[col], ax=axes[i],
                          kde=True, bins = np.arange(xrange[i][0], xrange[i][1], (xrange[i][1]-xrange[i][0])/20),
                          stat = 'probability')


        if len(correctParams) > 1:
          ax.axvline(np.log10(correctParams[i]), color = 'red', linewidth = 2)
```

```
        ax.set_title(f'log10 {greek_letters[i]}', fontsize=24)
        ax.tick_params(axis='x', labelsize=21)
        ax.tick_params(axis='y', labelsize=21)
        ax.set_xlabel(None)
        ax.set_ylabel(None)
        ax.set_ylim([0,ymax[i]])

        # Set x-axis range if specified
        if xrange and len(xrange) > i:
            ax.set_xlim(xrange[i])

    plt.tight_layout()
    plt.show()
    return logAllParamSets_sampled
```

ChatGPT

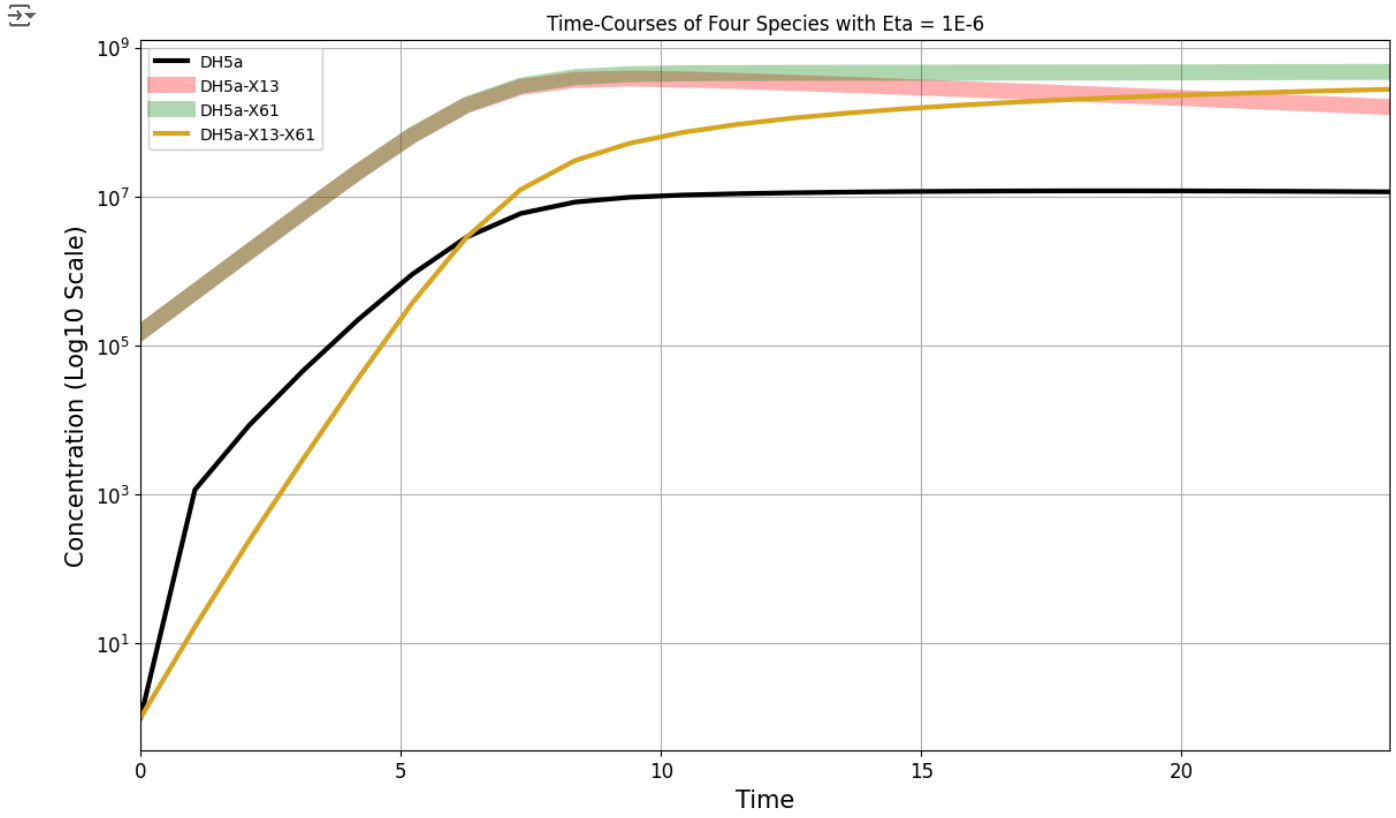
```
def findPeak(data):
  # Create the histogram
  counts, bin_edges = np.histogram(data, bins='auto')

  # Find peaks in the histogram (counts are the histogram heights)
  peaks, _ = find_peaks(counts)

  # Get the values (bin centers) corresponding to the peaks
  bin_centers = (bin_edges[:-1] + bin_edges[1:]) / 2
  peak_values = bin_centers[peaks]

  # Plot the histogram
  plt.hist(data, bins='auto', alpha=0.7, color='blue')
  plt.plot(bin_centers[peaks], counts[peaks], "x", color='red')  # Mark the peaks
  plt.xlabel('Value')
  plt.ylabel('Frequency')
  plt.title('Histogram with Peaks Marked')
  plt.show()

  # Print the values at the peaks and their corresponding bin centers
  print("Bin centers at the peaks:", bin_centers[peaks])

  # Find the indices of the original array values closest to each peak
  # Use .iloc to access elements by position in the Series
  closest_indices = [np.abs(data.to_numpy() - peak).argmin() for peak in peak_values]
  closest_values = data.iloc[closest_indices] # Access using iloc

  # Print the indices and the corresponding values in the original array
  print("Indices of closest values in the original array:", closest_indices)
  print("Values in the original array closest to the peaks:", closest_values)
  return closest_indices


### Define Parameters and Initial Conditions for Synthetic Data

# Parameters for synthetic data

# From Sheppard et al 2020
## eta is between 1E-20 and 1E-6
## mu, mu13, mu61, mu1361 values were estimated from doubling time
#### doubling time 30-35 min correspond to mu's values of 1.2-1.4 ml/cell/hr
#### Nm are estimated from value we measured in lab
## every 24 hr we dilute sample from 1E+9 down to 1.5E+5 --> correspond to D = 0.37
### dilute 10 fold in 24 hr correspond to D ~ 0.1

eta_value = 1E-10
kappa13_value = 1E-3#1E-8
kappa61_value = 1E-3#1E-9
mu_value = 1.4
mu13_value = 1.3
mu61_value = 1.3
mu1361_value = 1.2
D_value = 0.1
Nm_value = 1E9

correctParams = [eta_value, mu_value, mu13_value, mu61_value, mu1361_value, kappa13_value, kappa61_value, D_value]

# Initial conditions for synthetic data
initial_conditions_i = [0, 1.5E5, 1.5E5, 0]
initial_conditions_ii = [0, 1.5E5, 1.5E5, 0]
initial_conditions_iii = [0, 1.5E5, 1.5E2, 0]
initial_conditions_iv = [1.5E5, 0, 1.5E2, 0]
initial_condition_vDH5a   = [1.5E5, 0, 0, 0]
initial_condition_vDH5a13 = [0, 1.5E5, 0, 0]
initial_condition_vDH5a61 = [0, 0, 1.5E5, 0]
initial_condition_vDH5a1361 = [0, 0, 0, 1.5E5]
```

```
# Time array prelim
times_simulation_prelim = np.linspace(0, 24, 24)
times_simulation_i   =  [0,1,2,4,8,16,24]; repeatNum_i = 3; sdNoise_i = 1
times_simulation_ii  =  [0,24,72,120,168,264,336]; repeatNum_ii  = 3; sdNoise_ii  = 1
times_simulation_iii =  [0,24,72,120,168,264,336]; repeatNum_iii = 3; sdNoise_iii = 1
times_simulation_iv  =  [0,1,2,4,8,16,24,72,120,168,264,336]; repeatNum_iv = 3; sdNoise_iv = 1

time_simulation_v = [0,1, 2, 4, 8, 16, 24,72,120]; repeatNum_v = 3; sdNoise_v = 1
```

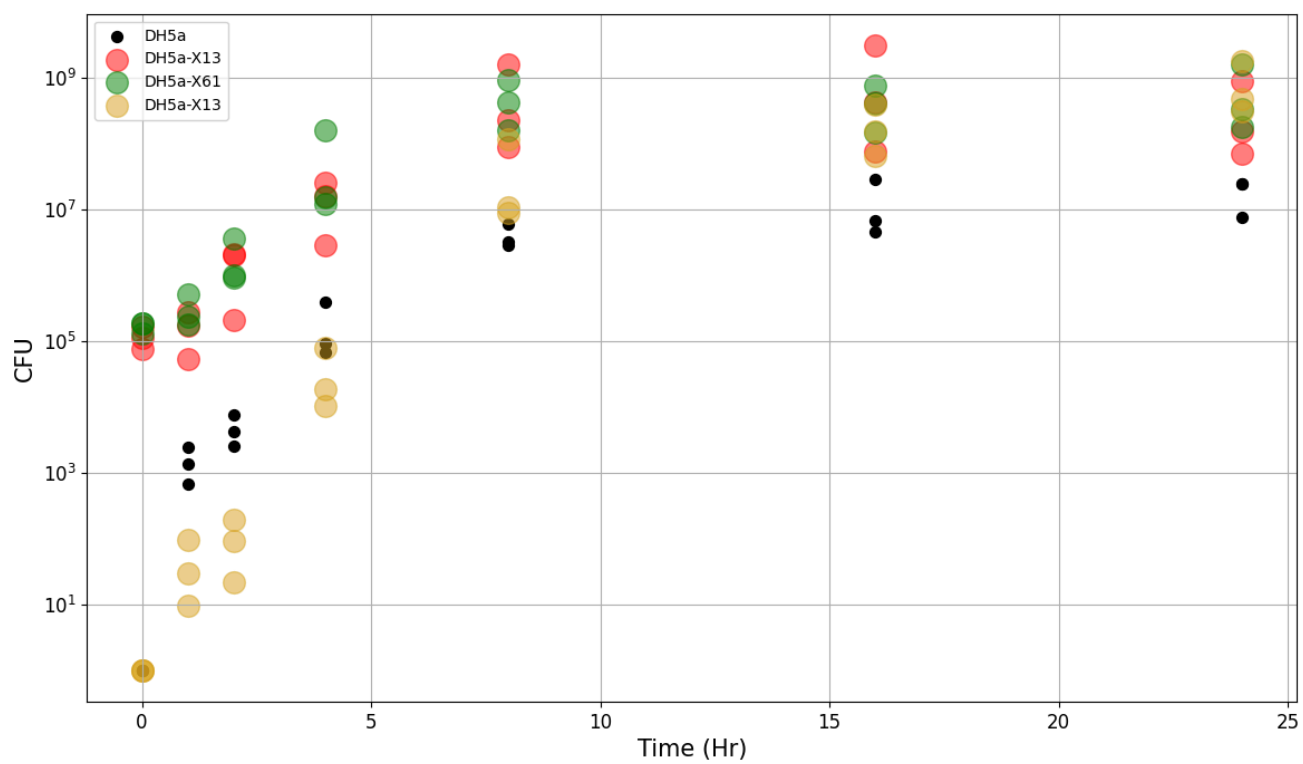
2. Simulated Data, Parameter Estimation & Parameter Analysis

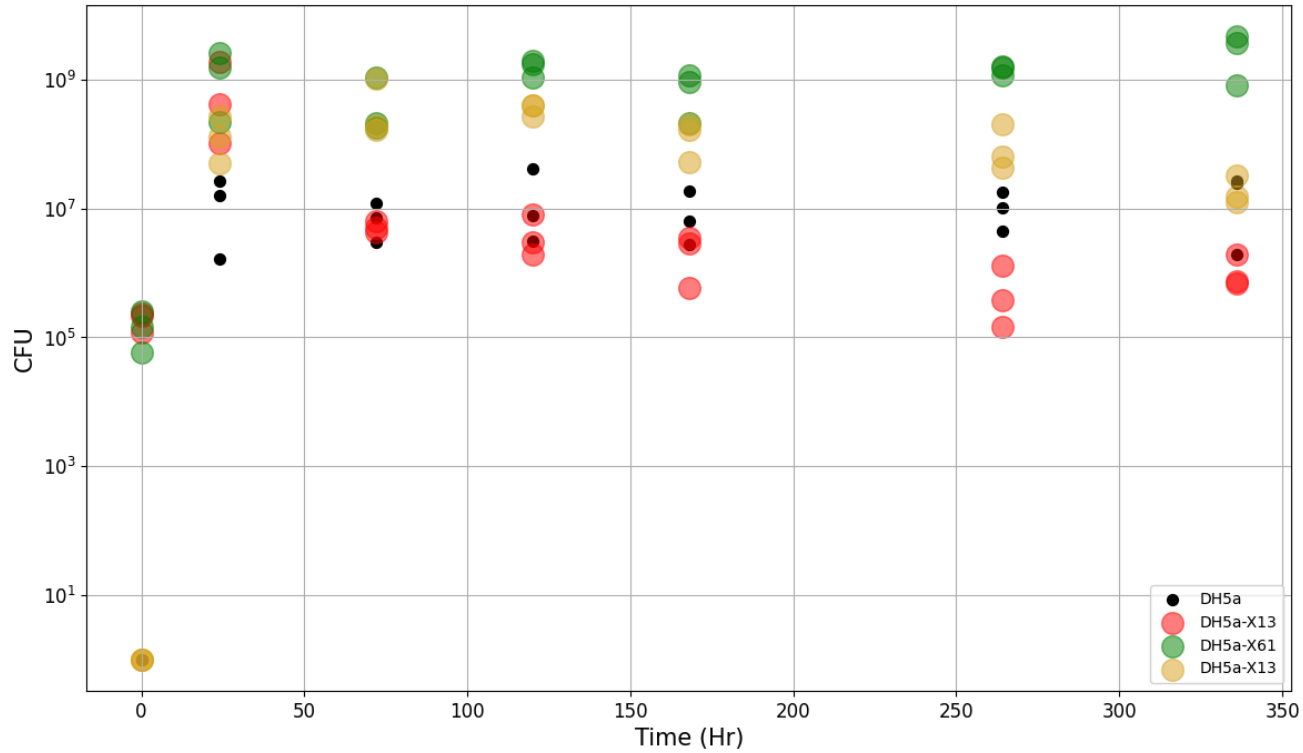
Generate Synthetic Data

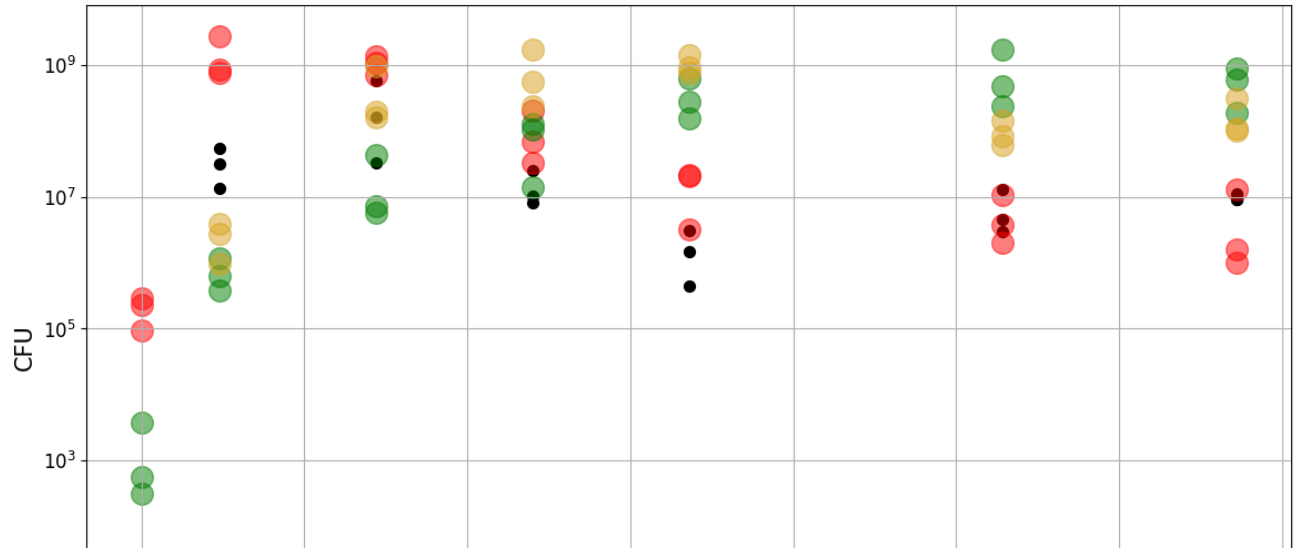
```
#### Prelim Simulation Timecourse

# Simulating the system
solution = odeint(
    conjugationODEs,
    initial_conditions_i,
    times_simulation_prelim,
    args=(eta_value, mu_value, mu13_value, mu61_value, mu1361_value, kappa13_value, kappa61_value, D_value, Nm_value)
)

# Plotting the results
plt.figure(figsize=(14, 8))
plt.semilogy(times_simulation_prelim, solution[:, 0]+1, label='DH5a', color='black', alpha=1, linewidth=3)
plt.semilogy(times_simulation_prelim, solution[:, 1]+1, label='DH5a-X13', color='red', alpha=0.3, linewidth=10)
plt.semilogy(times_simulation_prelim, solution[:, 2]+1, label='DH5a-X61', color='green', alpha=0.3, linewidth=10)
plt.semilogy(times_simulation_prelim, solution[:, 3]+1, label='DH5a-X13-X61', color='goldenrod', alpha=1, linewidth=3)
plt.xlabel('Time', fontsize = 15)
plt.ylabel('Concentration (Log10 Scale)', fontsize = 15)
plt.title('Time-Courses of Four Species with Eta = 1E-6')
plt.xlim(0, max(times_simulation_prelim))
plt.xticks(fontsize=12)
plt.yticks(fontsize=12)
plt.grid(True)
plt.legend()
plt.show()
```

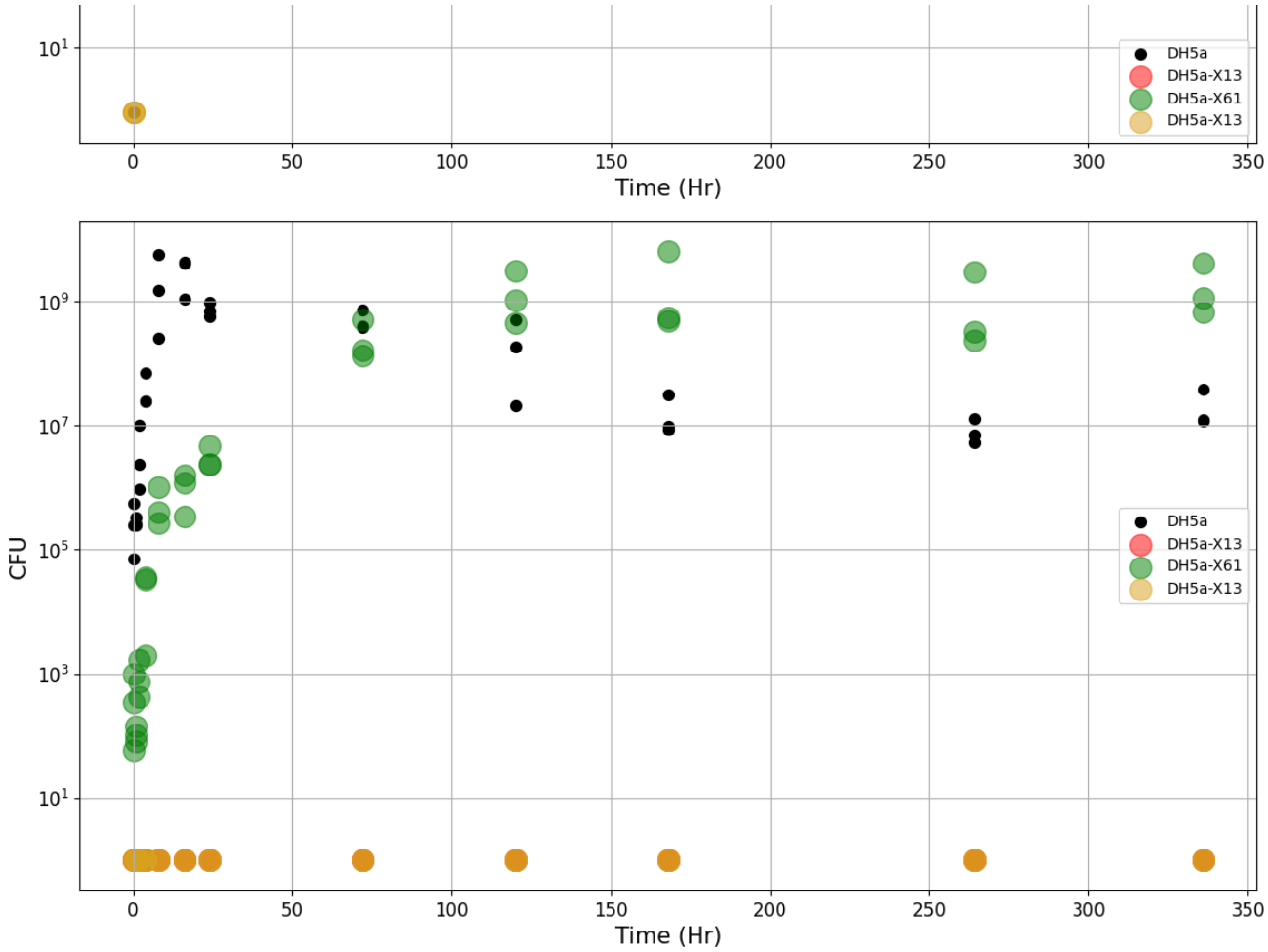



```
outputShownDf = [0,1,2,3]

#### (i) 24 hr with DH5a-X13:DH5a-X61 = 1:1
makeSynthData(initial_conditions_i, times_simulation_i, repeatNum_i, sdNoise_i, outputShownDf, 'synthData_i.csv')

#### (ii) 336 hr with DH5a-X13:DH5a-X61 = 1:1
makeSynthData(initial_conditions_ii, times_simulation_ii, repeatNum_ii, sdNoise_ii, outputShownDf, 'synthData_ii.csv')

#### (iii) 336 hr with DH5a-X13:DH5a-X61 = 1000:1
makeSynthData(initial_conditions_iii, times_simulation_iii, repeatNum_iii, sdNoise_iii, outputShownDf, 'synthData_iii.csv')

#### (iv) 336 hr with DH5a:DH5a-X61 = 1000:1
makeSynthData(initial_conditions_iv, times_simulation_iv, repeatNum_iv, sdNoise_iv, outputShownDf, 'synthData_iv.csv')
```

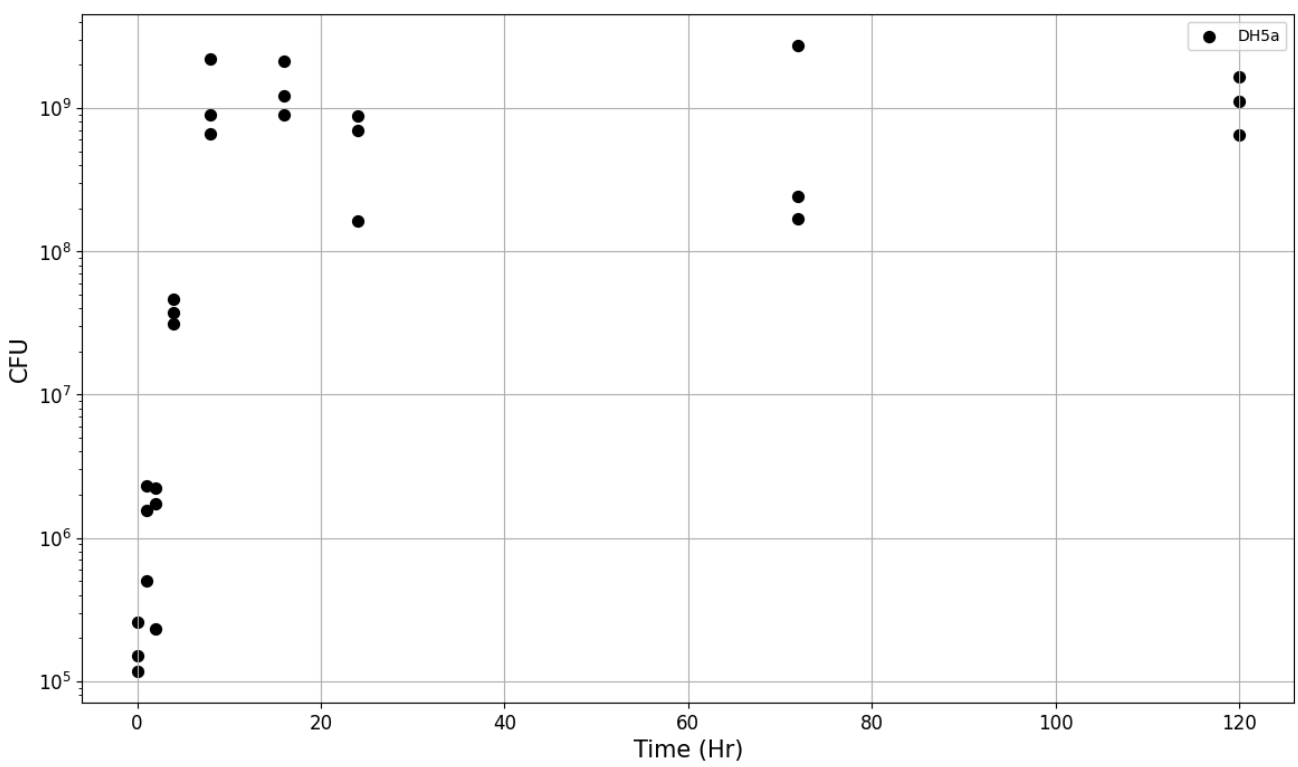

DH5a
DH5a-X13
DH5a-X61
DH5a-X13
CFU
Time (Hr)

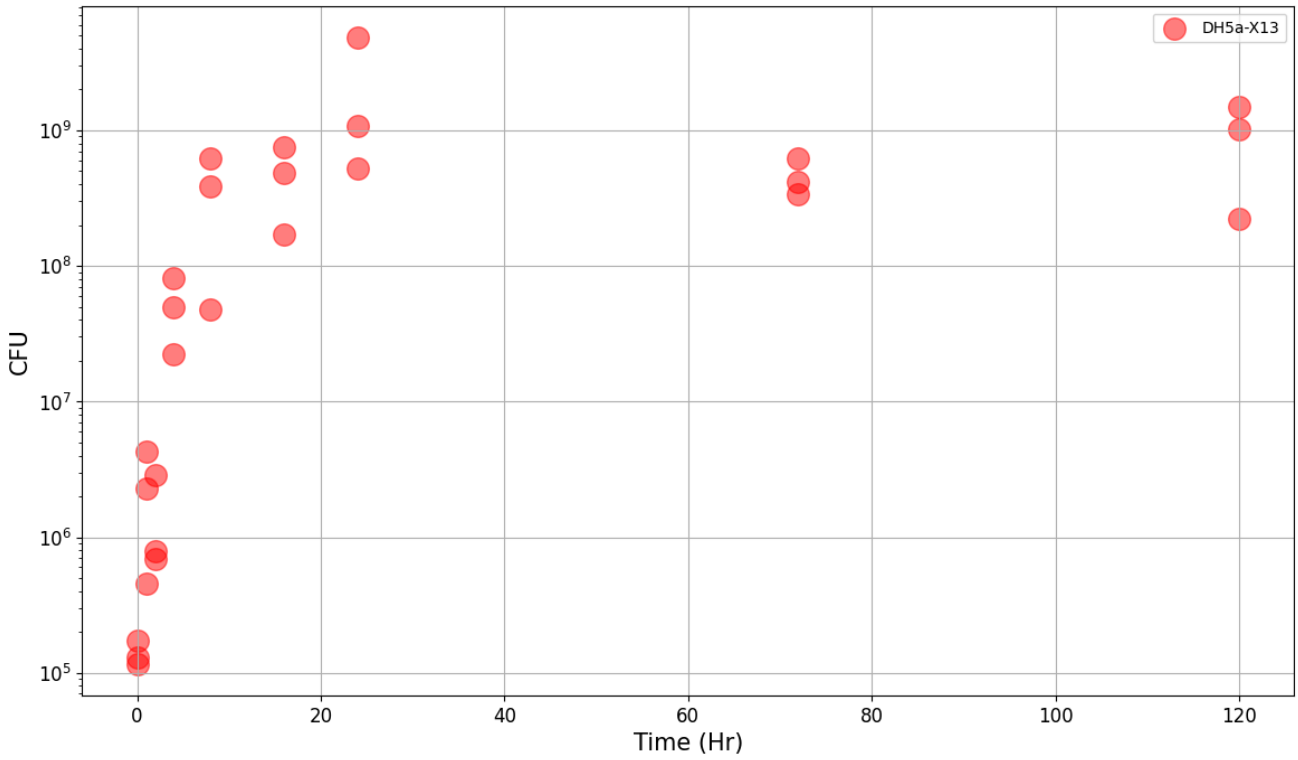

DH5a
DH5a-X13
DH5a-X61
DH5a-X13
CFU
Time (Hr)

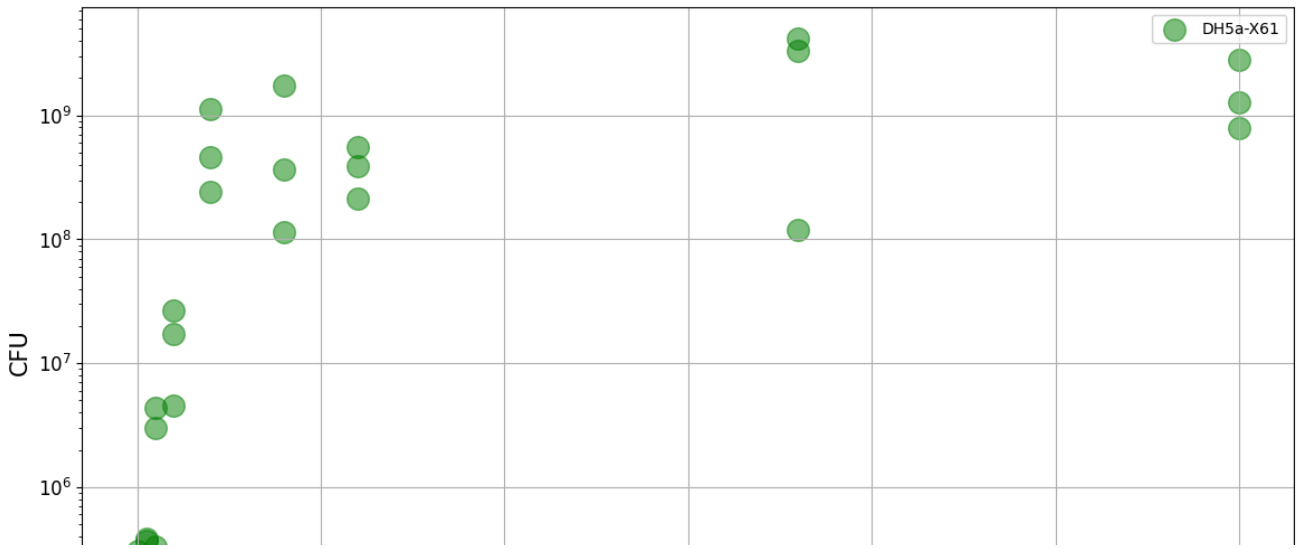

CFU

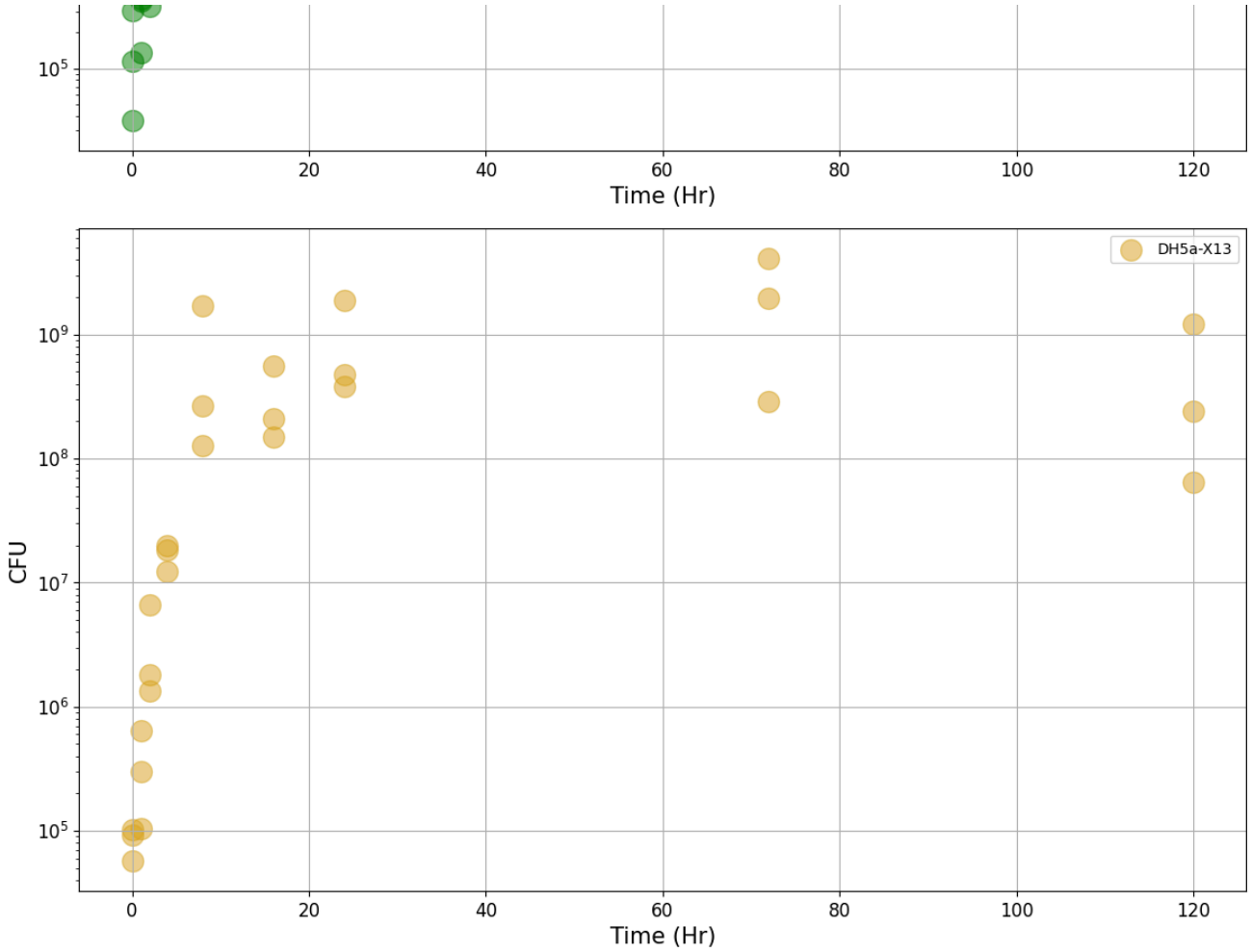

10^1
0
50
100
150
200
250
300
350
Time (Hr)
DH5a
DH5a-X13
DH5a-X61
DH5a-X13
10^9
10^7
10^5
10^3
10^1
CFU
0
50
100
150
200
250
300
350
Time (Hr)
DH5a
DH5a-X13
DH5a-X61
DH5a-X13

```
#### (v) 48 hr Growth Curve
makeSynthData(initial_condition_vDH5a, time_simulation_v, repeatNum_v, sdNoise_v, [0], 'synthData_vDH5a.csv')
makeSynthData(initial_condition_vDH5a13, time_simulation_v, repeatNum_v, sdNoise_v, [1], 'synthData_vDH5a13.csv')
makeSynthData(initial_condition_vDH5a61, time_simulation_v, repeatNum_v, sdNoise_v, [2], 'synthData_vDH5a61.csv')
makeSynthData(initial_condition_vDH5a1361, time_simulation_v, repeatNum_v, sdNoise_v, [3], 'synthData_vDH5a1361.csv')
```


DH5a
CFU
Time (Hr)


DH5a-X13
CFU
Time (Hr)


DH5a-X61
CFU


10^5
0
20
40
60
80
100
120
Time (Hr)
DH5a-X13
10^9
10^8
10^7
10^6
10^5
CFU
0
20
40
60
80
100
120
Time (Hr)

Parameter Estimation

Define parameters for Metropolist Monte Carlo

```
# Define the minimal and maximal values for starting parameters
# Note that these are just starting parameters, not limited ranges of possible parameter values.
# Thus, the actual Metropolis random walk could go beyond these min and max values

min_values = [1E-13, 0.1, 0.1, 0.1, 0.1, 1E-7, 1E-7, 0.01]
max_values = [1E-7, 10, 10, 10, 10, 10, 10, 10]

proposal_fraction = [0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05] # step size in log scale for Metropolis random walk
iterNum = 20000
burnNum = 5000
```

Preliminary Parameter Estimation and Analysis

```
# Starting parameters, proposal fractions, interation number
starting_parameters = [1E-5, 0.7, 2.5, 0.05, 0.01, 0.01, 0.01, 0.01]  # manualy chosen random
#starting_parameters = [eta_value, mu_value, mu13_value, mu61_value, mu1361_value, kappa13_value, kappa61_value, D_value] #
starting_parameters = generate_random_parameters(min_values, max_values)
```

ChatGPT

```
# Read synthetic data from csv
df_simulated_data = pd.read_csv('synthData_i.csv')
data = df_simulated_data[['DH5a', 'DH5a-X13', 'DH5a-X61', 'DH5a-X13-X61']].values
times = df_simulated_data['Time (Hr)'].values

df_simulated_dataLONG = pd.read_csv('synthData_ii.csv')
dataLONG = df_simulated_dataLONG[['DH5a', 'DH5a-X13', 'DH5a-X61', 'DH5a-X13-X61']].values
timesLONG = df_simulated_dataLONG['Time (Hr)'].values

# Run Metropolis
start_time = time.time()

allSpecies = [0,1,2,3]
#noDH5a = [1,2,3]

## short time
ensemble_iterations = metropolis(starting_parameters, proposal_fraction, iterNum,
 (conjugationODEs, initial_conditions_i, data, times, allSpecies))

## long time
#ensemble_iterations = metropolis(starting_parameters, proposal_fraction, iterNum,
# (conjugationODEs, initial_conditions_i, dataLONG, timesLONG))

## mix
#ensemble_iterations = metropolis(starting_parameters, proposal_fraction, iterNum,
# (conjugationODEs, initial_conditions_i, data, times),
# (conjugationODEs, initial_conditions_ii, dataLONG, timesLONG))

elapsed_time = time.time() - start_time


# save emsemble parameters to csv
ensemble_iterations_pd  = pd.DataFrame(ensemble_iterations, columns=['Iteration', 'Posterior', 'Eta', 'Mu', 'Mu13', 'Mu61',
ensemble_iterations_pd.to_csv('ensembleParamsPrelim_i.csv', encoding='utf-8', index=False)

print(f"Simulation completed in {elapsed_time:.2f} seconds.")



# Show line plots of parameter values vs iterations

#df_ensemble = pd.DataFrame(ensemble_iterations, columns=['Iteration', 'Posterior', 'Eta', 'Mu', 'Mu13', 'Mu61', 'Mu1361'])
df_ensemble = pd.read_csv('ensembleParamsPrelim_i.csv')

fig, axes = plt.subplots(5, 1, figsize=(12, 12))

# Plot Eta on the top subplot
axes[0].plot(df_ensemble['Iteration'], df_ensemble['Eta'], label='Eta', color='blue')
axes[0].set_xlabel('Iteration')
axes[0].set_ylabel('Eta Value')
axes[0].set_yscale('log')  # Set y-axis to logarithmic scale for the top subplot
```

```
axes[0].set_title('Trace of Eta Value')
axes[0].legend()
axes[0].grid(True)


# Plot Mu, Mu13, Mu61, Mu1361 on the second subplot
for param, color in zip(['Mu', 'Mu13', 'Mu61', 'Mu1361'], ['red', 'green', 'black', 'purple']):
    axes[1].plot(df_ensemble['Iteration'], df_ensemble[param], label=param, color=color)
axes[1].set_xlabel('Iteration')
axes[1].set_ylabel('Parameter Value')
axes[1].set_yscale('log')  # Set y-axis to logarithmic scale for the top subplot
axes[1].set_title('Trace of Mu, Mu13, Mu61, Mu1361 Values')
axes[1].legend()
axes[1].grid(True)

# Plot Kappa13, Kappa61, on the third subplot
for param, color in zip(['Kappa13', 'Kappa61'], ['red', 'green']):
    axes[2].plot(df_ensemble['Iteration'], df_ensemble[param], label=param, color=color)
axes[2].set_xlabel('Iteration')
axes[2].set_ylabel('Parameter Value')
axes[2].set_yscale('log')  # Set y-axis to logarithmic scale for the top subplot
axes[2].set_title('Trace of Kappa13, Kappa61 Values')
axes[2].legend()
axes[2].grid(True)

# Plot D, on the forth subplot
for param, color in zip(['D'], ['blue']):
    axes[3].plot(df_ensemble['Iteration'], df_ensemble[param], label=param, color=color)
axes[3].set_xlabel('Iteration')
axes[3].set_ylabel('Parameter Value')
axes[3].set_yscale('log')  # Set y-axis to logarithmic scale for the top subplot
axes[3].set_title('Trace of D Values')
axes[3].legend()
axes[3].grid(True)

# plot posterior
axes[4].plot(df_ensemble['Iteration'], df_ensemble['Posterior'], label='Posterior', color='blue')
axes[4].set_xlabel('Iteration')
axes[4].set_ylabel('Posterior Value')
axes[4].set_title('Trace of Posterior')
axes[4].legend()
axes[4].grid(True)

plt.tight_layout()
plt.show()
```

Double-click (or enter) to edit

```
df_ensemble = pd.read_csv('ensembleParamsPrelim_i.csv')

params_subset = df_ensemble.iloc[(burnNum+1):, 2:].values  # post burnt ensemble

#params_subset = df_ensemble.iloc[0:10, 2:].values  ## initial parameter ensemble


# get the best parameter (max posterior)
params_Best = df_ensemble.iloc[df_ensemble['Posterior'].idxmax(), 2:].values
formatted_array = ["{:.1e}".format(x) for x in params_Best]

# Randomly selecting 20 parameter sets from params_subset
np.random.seed(0)  # Setting a seed for reproducibility
random_indices = np.random.choice(len(params_subset), 200, replace=False)
selected_params = params_subset[random_indices]



finer_dataTime = np.arange(0, 25, 1)

finer_simulations = [simulate_timecourse(params, conjugationODEs, initial_conditions_i, finer_dataTime) for params in select
finer_simulations_array = np.array(finer_simulations)
#finer_mean_simulation = np.mean(finer_simulations_array, axis=0)+1 # regular mean
finer_mean_simulation =  np.exp(np.mean(np.log(finer_simulations_array), axis =0))+1 # geometric mean
```

```
finer_lower_bound = np.percentile(finer_simulations_array, 2.5, axis=0)+1
finer_upper_bound = np.percentile(finer_simulations_array, 97.5, axis=0)+1

# 5. Plot the average and 95% interval of simulated timecourse dynamics; overlay synthetic data

species_column_names = ['DH5a', 'DH5a-X13', 'DH5a-X61', 'DH5a-X13-X61']
colors = ['black', 'red', 'green', 'goldenrod']
plt.figure(figsize=(14, 10))
for i, (species, color) in enumerate(zip(species_column_names, colors)):
    #plt.plot(finer_dataTime, finer_mean_simulation[:, i], label=f'{species} Mean', color=color)
    plt.plot(finer_dataTime, finer_mean_simulation[:, i], color=color)
    plt.fill_between(finer_dataTime, finer_lower_bound[:, i], finer_upper_bound[:, i], color=color, alpha=0.2)
    plt.scatter(df_simulated_data['Time (Hr)'], df_simulated_data[species], label=f'{species} Data', color=color, s=50)
plt.xlabel('Time (Hr)', fontsize = 24)
plt.ylabel('CFU', fontsize = 24)
plt.yscale('log')
plt.title('Mean, 95% Confidence Intervals (Finer Scale), and Synthetic Data for Species Concentrations')
plt.grid(True)
plt.legend(loc = 'upper left')
plt.xticks(fontsize=18)
plt.yticks(fontsize=18)


param_names = ['Eta', 'Mu', 'Mu13', 'Mu61', 'Mu1361', 'Kappa13', 'Kappa61', 'D']

# Calculating standard deviation for each parameter in the params_subset
std_devs = params_subset.std(axis=0)
std_dev_text = [f"Std Dev {name}: {value:.1e}" for name, value in zip(param_names, std_devs)]

# True parameter values
true_values = [eta_value, mu_value, mu13_value, mu61_value, mu1361_value]
true_param_text = [f"True {name}: {value:.1e}" for name, value in zip(param_names, true_values)]

# Displaying comparison of starting parameters, best parameters, and true parameters on the plot
starting_values = [f"Start {name}: {value:.1e}" for name, value in zip(param_names, starting_parameters)]
best_values = [f"Best {name}: {value:.1e}" for name, value in zip(param_names, params_Best)]
param_comparison = "\n".join(starting_values + [""] + best_values + [""] + std_dev_text + [""] + true_param_text)
plt.text(26, 1e1, param_comparison, bbox=dict(facecolor='white', edgecolor='black'))
```

```
params_subset = df_ensemble.iloc[(burnNum+1):, 2:].values  # post burnt ensemble
```

```
#params_subset = df_ensemble.iloc[0:10, 2:].values  ## initial parameter ensemble


# get the best parameter (max posterior)
params_Best = df_ensemble.iloc[df_ensemble['Posterior'].idxmax(), 2:].values
formatted_array = ["{:.1e}".format(x) for x in params_Best]

# Randomly selecting 20 parameter sets from params_subset
np.random.seed(0)  # Setting a seed for reproducibility
random_indices = np.random.choice(len(params_subset), 200, replace=False)
selected_params = params_subset[random_indices]



finer_dataTime = np.arange(0, 350, 1)

finer_simulations = [simulate_timecourse(params, conjugationODEs, initial_conditions_ii, finer_dataTime) for params in selec
finer_simulations_array = np.array(finer_simulations)
finer_mean_simulation =  np.exp(np.mean(np.log(finer_simulations_array), axis =0))+1 # geometric mean


finer_lower_bound = np.percentile(finer_simulations_array, 2.5, axis=0)+1
finer_upper_bound = np.percentile(finer_simulations_array, 97.5, axis=0)+1

# 5. Plot the average and 95% interval of simulated timecourse dynamics; overlay synthetic data

species_column_names = ['DH5a', 'DH5a-X13', 'DH5a-X61', 'DH5a-X13-X61']
colors = ['black', 'red', 'green', 'goldenrod']
plt.figure(figsize=(14, 10))
for i, (species, color) in enumerate(zip(species_column_names, colors)):
    #plt.plot(finer_dataTime, finer_mean_simulation[:, i], label=f'{species} Mean', color=color)
    plt.plot(finer_dataTime, finer_mean_simulation[:, i], color=color)
    plt.fill_between(finer_dataTime, finer_lower_bound[:, i], finer_upper_bound[:, i], color=color, alpha=0.2)
    plt.scatter(df_simulated_dataLONG['Time (Hr)'], df_simulated_dataLONG[species], label=f'{species} Data', color=color, s=
plt.xlabel('Time (Hr)', fontsize = 24)
plt.ylabel('CFU', fontsize = 24)
plt.yscale('log')
#plt.title('Mean, 95% Confidence Intervals (Finer Scale), and Synthetic Data for Species Concentrations')
plt.grid(True)
#plt.legend(loc = 'upper left')
plt.xticks(fontsize=18)
plt.yticks(fontsize=18)


param_names = ['Eta', 'Mu', 'Mu13', 'Mu61', 'Mu1361', 'Kappa13', 'Kappa61', 'D']

# Calculating standard deviation for each parameter in the params_subset
#std_devs = params_subset.std(axis=0)
#std_dev_text = [f"Std Dev {name}: {value:.1e}" for name, value in zip(param_names, std_devs)]

# True parameter values
#true_values = [eta_value, mu_value, mu13_value, mu61_value, mu1361_value]
#true_param_text = [f"True {name}: {value:.1e}" for name, value in zip(param_names, true_values)]

# Displaying comparison of starting parameters, best parameters, and true parameters on the plot
#starting_values = [f"Start {name}: {value:.1e}" for name, value in zip(param_names, starting_parameters)]
#best_values = [f"Best {name}: {value:.1e}" for name, value in zip(param_names, params_Best)]
#param_comparison = "\n".join(starting_values + [""] + best_values + [""] + std_dev_text + [""] + true_param_text)
#plt.text(380, 1e1, param_comparison, bbox=dict(facecolor='white', edgecolor='black'))
```

Full Parameter Estimation (from different initial parameter values)

```
import pandas as pd
import numpy as np


# Read synthetic data from csv
# Assuming you have already imported pandas as pd
df_simulated_data = pd.read_csv('synthData_i.csv')
data = df_simulated_data[['DH5a', 'DH5a-X13', 'DH5a-X61', 'DH5a-X13-X61']].values
times = df_simulated_data['Time (Hr)'].values

df_simulated_dataLONG = pd.read_csv('synthData_ii.csv')
dataLONG = df_simulated_dataLONG[['DH5a', 'DH5a-X13', 'DH5a-X61', 'DH5a-X13-X61']].values
timesLONG = df_simulated_dataLONG['Time (Hr)'].values

df_simulated_DH5aGrowth = pd.read_csv('synthData_vDH5a.csv')
dataDH5a = df_simulated_DH5aGrowth[['DH5a', 'DH5a-X13', 'DH5a-X61', 'DH5a-X13-X61']].values
timesDH5a = df_simulated_DH5aGrowth['Time (Hr)'].values

df_simulated_DH5a13Growth = pd.read_csv('synthData_vDH5a13.csv')
dataDH5a13 = df_simulated_DH5a13Growth[['DH5a', 'DH5a-X13', 'DH5a-X61', 'DH5a-X13-X61']].values
timesDH5a13 = df_simulated_DH5a13Growth['Time (Hr)'].values

df_simulated_DH5a61Growth = pd.read_csv('synthData_vDH5a61.csv')
dataDH5a61 = df_simulated_DH5a61Growth[['DH5a', 'DH5a-X13', 'DH5a-X61', 'DH5a-X13-X61']].values
timesDH5a61 = df_simulated_DH5a61Growth['Time (Hr)'].values

df_simulated_DH5a1361Growth = pd.read_csv('synthData_vDH5a1361.csv')
dataDH5a1361 = df_simulated_DH5a1361Growth[['DH5a', 'DH5a-X13', 'DH5a-X61', 'DH5a-X13-X61']].values
timesDH5a1361 = df_simulated_DH5a1361Growth['Time (Hr)'].values

noDH5a = [1,2,3]
sDH5a = [0]
sDH5a13 = [1]
sDH5a61 = [2]
sDH5a1361 = [3]


# Run Metropolis 10 times
```

```
for i in range(10):
    # Generate random starting parameters for each iteration
    starting_parameters = generate_random_parameters(min_values, max_values)

    # Run Metropolis
    start_time = time.time()

    #ensemble_iterations = metropolis(starting_parameters, proposal_fraction, iterNum,
    #              (conjugationODEs, initial_conditions_i, data, times, noDH5a))

    #ensemble_iterations = metropolis(starting_parameters, proposal_fraction, iterNum,
    #             (conjugationODEs, initial_conditions_i, data, times, noDH5a),
    #             (conjugationODEs, initial_conditions_ii, dataLONG, timesLONG, noDH5a))

    #ensemble_iterations = metropolis(starting_parameters, proposal_fraction, iterNum,
    #              (conjugationODEs, initial_conditions_i, data, times, noDH5a),
    #              (conjugationODEs, initial_condition_vDH5a, dataDH5a, timesDH5a, sDH5a),
    #              (conjugationODEs, initial_condition_vDH5a13, dataDH5a13, timesDH5a13, sDH5a13),
    #              (conjugationODEs, initial_condition_vDH5a61, dataDH5a61, timesDH5a61, sDH5a61),
    #              (conjugationODEs, initial_condition_vDH5a1361, dataDH5a1361, timesDH5a1361, sDH5a1361))

    ensemble_iterations = metropolis(starting_parameters, proposal_fraction, iterNum,
                  (conjugationODEs, initial_conditions_i, data, times, noDH5a),
                  (conjugationODEs, initial_conditions_ii, dataLONG, timesLONG, noDH5a),
                  (conjugationODEs, initial_condition_vDH5a, dataDH5a, timesDH5a, sDH5a),
                  (conjugationODEs, initial_condition_vDH5a13, dataDH5a13, timesDH5a13, sDH5a13),
                  (conjugationODEs, initial_condition_vDH5a61, dataDH5a61, timesDH5a61, sDH5a61),
                  (conjugationODEs, initial_condition_vDH5a1361, dataDH5a1361, timesDH5a1361, sDH5a1361))



    elapsed_time = time.time() - start_time

    # Save ensemble parameters to csv
    ensemble_iterations_pd = pd.DataFrame(
        ensemble_iterations,
        columns=['Iteration', 'Posterior', 'Eta', 'Mu', 'Mu13', 'Mu61', 'Mu1361', 'Kappa13', 'Kappa61', 'D']
    )
    filename = f'ensembleParamsMXnDgr_{i+1}.csv'
    ensemble_iterations_pd.to_csv(filename, encoding='utf-8', index=False)

    print(f"Round {i+1} completed in {elapsed_time:.2f} seconds.")
```

Analyze Parameter Ensembles

Visualize Parameter Ensembles over Metropolis Iterations

```
filenames = ['ensembleParamsSHnDgr_1.csv','ensembleParamsSHnDgr_2.csv','ensembleParamsSHnDgr_3.csv',
             'ensembleParamsSHnDgr_4.csv','ensembleParamsSHnDgr_5.csv','ensembleParamsSHnDgr_6.csv',
             'ensembleParamsSHnDgr_7.csv','ensembleParamsSHnDgr_8.csv','ensembleParamsSHnDgr_9.csv',
             'ensembleParamsSHnDgr_10.csv',]

correct_values = [eta_value, mu_value, mu13_value, mu61_value, mu1361_value, kappa13_value, kappa61_value, D_value]
plot_multiple_files_updated(filenames, correct_values)
```

```
filenames = ['ensembleParamsMXnDgr_1.csv','ensembleParamsMXnDgr_2.csv','ensembleParamsMXnDgr_3.csv',
             'ensembleParamsMXnDgr_4.csv','ensembleParamsMXnDgr_5.csv','ensembleParamsMXnDgr_6.csv',
             'ensembleParamsMXnDgr_8.csv','ensembleParamsMXnDgr_9.csv', 'ensembleParamsMXnDgr_10.csv',
             'ensembleParamsMXnDgr_7.csv',
             ]

correct_values = [eta_value, mu_value, mu13_value, mu61_value, mu1361_value, kappa13_value, kappa61_value, D_value]
plot_multiple_files_updated(filenames, correct_values)
```

Simulate Timecourses from Parameter Ensembles

Parameter Ensembles from (i) + (v) training set

```
# get file names from good Metropolis iterations
filenamesSHORT = ['ensembleParamsSHnDgr_1.csv','ensembleParamsSHnDgr_2.csv','ensembleParamsSHnDgr_3.csv',
             'ensembleParamsSHnDgr_4.csv','ensembleParamsSHnDgr_5.csv','ensembleParamsSHnDgr_6.csv',
             'ensembleParamsSHnDgr_7.csv','ensembleParamsSHnDgr_8.csv','ensembleParamsSHnDgr_9.csv',
             'ensembleParamsSHnDgr_10.csv',]

# trim, concatenate and select parameter sets from ensembles
ensembleParamsSHORTset = trim_and_concat_csv(filenamesSHORT)
param_Best, selected_params = selParam(ensembleParamsSHORTset, 200)



# display synthetic data, mean and 95% interval of simulated timecourses from ensembles

showDataFit(selected_params, initial_conditions_i, 25, 'synthData_i.csv',
            ['DH5a-X13', 'DH5a-X61', 'DH5a-X13-X61'], ['red', 'green', 'goldenrod'])

showDataFit(selected_params, initial_conditions_ii, 336, 'synthData_ii.csv',
            ['DH5a-X13', 'DH5a-X61', 'DH5a-X13-X61'], ['red', 'green', 'goldenrod'])

showDataFit(selected_params, initial_conditions_iii, 336, 'synthData_iii.csv',
            ['DH5a-X13', 'DH5a-X61', 'DH5a-X13-X61'], ['red', 'green', 'goldenrod'])
```

```
showDataFit(selected_params, initial_conditions_iv, 336, 'synthData_iv.csv',
            ['DH5a', 'DH5a-X61'], ['black', 'green'])
```

```
AinitCond = [initial_condition_vDH5a, initial_condition_vDH5a13,
             initial_condition_vDH5a61, initial_condition_vDH5a1361]
AdataName = ['synthData_vDH5a.csv', 'synthData_vDH5a13.csv',
             'synthData_vDH5a61.csv', 'synthData_vDH5a1361.csv']

showDataFitGrowth(selected_params, AinitCond, 125, AdataName,
                  ['DH5a', 'DH5a-X13', 'DH5a-X61', 'DH5a-X13-X61'],
                  ['black', 'red', 'green', 'goldenrod'])
```

Parameter Ensembles from (i) +(ii) + (v) training set

```
# get file names from good Metropolis iterations
filenamesMIX = ['ensembleParamsMXnDgr_1.csv','ensembleParamsMXnDgr_2.csv','ensembleParamsMXnDgr_3.csv',
             'ensembleParamsMXnDgr_4.csv','ensembleParamsMXnDgr_5.csv','ensembleParamsMXnDgr_6.csv',
             'ensembleParamsMXnDgr_8.csv','ensembleParamsMXnDgr_9.csv', 'ensembleParamsMXnDgr_10.csv',
             ]

# trim, concatenate and select parameter sets from ensembles
ensembleParamsMIXset = trim_and_concat_csv(filenamesMIX)
param_Best, selected_params = selParam(ensembleParamsMIXset, 200)


# display synthetic data, mean and 95% interval of simulated timecourses from ensembles

showDataFit(selected_params, initial_conditions_i, 25, 'synthData_i.csv',
            ['DH5a-X13', 'DH5a-X61', 'DH5a-X13-X61'], ['red', 'green', 'goldenrod'])

showDataFit(selected_params, initial_conditions_ii, 336, 'synthData_ii.csv',
            ['DH5a-X13', 'DH5a-X61', 'DH5a-X13-X61'], ['red', 'green', 'goldenrod'])

showDataFit(selected_params, initial_conditions_iii, 336, 'synthData_iii.csv',
            ['DH5a-X13', 'DH5a-X61', 'DH5a-X13-X61'], ['red', 'green', 'goldenrod'])

showDataFit(selected_params, initial_conditions_iv, 336, 'synthData_iv.csv',
            ['DH5a', 'DH5a-X61'], ['black', 'green'])
```

```
AinitCond = [initial_condition_vDH5a, initial_condition_vDH5a13,
             initial_condition_vDH5a61, initial_condition_vDH5a1361]
AdataName = ['synthData_vDH5a.csv', 'synthData_vDH5a13.csv',
             'synthData_vDH5a61.csv', 'synthData_vDH5a1361.csv']

showDataFitGrowth(selected_params, AinitCond, 125, AdataName,
                  ['DH5a', 'DH5a-X13', 'DH5a-X61', 'DH5a-X13-X61'],
                  ['black', 'red', 'green', 'goldenrod'])
```

Parameter Distribution and Correlation Analysis

```
xrange = [[-11.5,-8.5], [-0.5, 1], [-0.5,1], [-0.5,1],[-0.5,1], [-18,1], [-18,1], [-6,1]]
np.arange(xrange[1][0], xrange[1][1], (xrange[1][1]-xrange[1][0])/20)
```

```
array([-0.5  , -0.425, -0.35 , -0.275, -0.2  , -0.125, -0.05 ,  0.025,
        0.1  ,  0.175,  0.25 ,  0.325,  0.4  ,  0.475,  0.55 ,  0.625,
        0.7  ,  0.775,  0.85 ,  0.925])
```

Parameter Ensembles from (i) + (v) training set

```
process_and_plot_csv(filenamesSHORT, 5000, 0.020, correctParams, xrange)
```

Parameter Ensembles from (i) + (ii) + (v) training set

```
process_and_plot_csv(filenamesMIX, 5000, 0.028, correctParams, xrange)
```

2. Experimental Data, Parameter Estimation & Parameter Analysis

Import and process experimental data

```
import pandas as pd

def conjDataProcess(csv_file, ExpList, outputNameList):
    # Import the csv file into a data frame 'df'
    df = pd.read_csv(csv_file)

    # Process each element in ExpList
    for i, ExpList_i in enumerate(ExpList):
        # Find rows in df whose ExperimentID value is ExpList_i
        df_i = df[df['ExperimentID'].isin(ExpList_i)]

        # Remove ExperimentID column
        df_i = df_i.drop(columns=['ExperimentID'])

        df_i.iloc[:, [1, 2, 3, 4]] = df_i.iloc[:, [1, 2, 3, 4]] + 1

        # Create a csv file for each subset of data
        df_i.to_csv(f'{outputNameList[i]}.csv', index=False)

# Example usage
```

```
# conjDataProcess_v2('path_to_csv_file.csv', [[1,2,3], [4,5,6], [7,8,9], [10,11,12]])



#ExpList = [['CJ002'], ['CJ003'], ['CJ029', 'CJ030'], ['CJ026', 'CJ027']] ##Old
ExpList = [['CJ002'], ['CJ003'], ['CJ029', 'CJ030'], ['CJ034']] # update 24/04/24
outputNameList = ['expData_i', 'expData_ii', 'expData_iii', 'expData_iv']
conjDataProcess('Nms020_conjData.csv', ExpList, outputNameList)

ExpList = [['CT004', 'CT006'], ['CT012', 'CT013'], ['CT012x', 'CT013x'], ['CT012y', 'CT013y']]
outputNameList = ['expData_vDH5a', 'expData_vDH5a13', 'expData_vDH5a61', 'expData_vDH5a1361']
conjDataProcess('Nms020_growthData.csv', ExpList, outputNameList)


def showDataFit2(selected_params, initCond, Tmax, dataName, dataShown, colorShown):
  ## finer_dataTime = np.arange(0, Tmax, 1)

  ##finer_simulations = [simulate_timecourse(params, conjugationODEs, initCond, finer_dataTime) for params in selected_param
  ##finer_simulations_array = np.array(finer_simulations)
  ##finer_mean_simulation =  np.exp(np.mean(np.log(finer_simulations_array), axis =0))+1 # geometric mean

  #finer_best_simulation = simulate_timecourse(params_Best, conjugationODEs, initCond, finer_dataTime)
  #finer_best_simulation_array = np.array(finer_best_simulation)

  ##finer_lower_bound = np.percentile(finer_simulations_array, 2.5, axis=0)+1
  ##finer_upper_bound = np.percentile(finer_simulations_array, 97.5, axis=0)+1

  df_simulated_data = pd.read_csv(dataName)

  speciesInd = {
      "DH5a":0,
      "DH5a-X13":1,
      "DH5a-X61":2,
      "DH5a-X13-X61":3
  }

  plt.figure(figsize=(7, 5))
  for i, (species, color) in enumerate(zip(dataShown, colorShown)):
    ##plt.plot(finer_dataTime, finer_mean_simulation[:, speciesInd[species]], color=color)
    #plt.fill_between(finer_dataTime, finer_lower_bound[:, speciesInd[species]], finer_upper_bound[:, speciesInd[species]],
    plt.scatter(df_simulated_data['Time (Hr)'], df_simulated_data[species], label=f'{species} Data', color=color, s=50)
  plt.xlabel('Time (Hr)', fontsize = 20)
  plt.ylabel('CFU', fontsize = 20)
  plt.yscale('log')
  plt.ylim([1E-1, 1E+10])
  plt.grid(True)
  plt.xticks(fontsize=18)
  plt.yticks(fontsize=18)


def showDataFitGrowth2(selected_params, AinitCond, Tmax, AdataName, AdataShown, AcolorShown):
  finer_dataTime = np.arange(0, Tmax, 1)

  speciesInd = {
      "DH5a":0,
      "DH5a-X13":1,
      "DH5a-X61":2,
      "DH5a-X13-X61":3
  }

  plt.figure(figsize=(7, 5))

  for k in range(0, len(AinitCond)):
    initCond = AinitCond[k]
    dataName = AdataName[k]
    species = AdataShown[k]
    color = AcolorShown[k]

    ###finer_simulations = [simulate_timecourse(params, conjugationODEs, initCond, finer_dataTime) for params in selected_pa
    ###finer_simulations_array = np.array(finer_simulations)
    ###finer_mean_simulation =  np.exp(np.mean(np.log(finer_simulations_array), axis =0))+1 # geometric mean
    #finer_best_simulation = simulate_timecourse(params_Best, conjugationODEs, initCond, finer_dataTime)
    #finer_best_simulation_array = np.array(finer_best_simulation)

    ###finer_lower_bound = np.percentile(finer_simulations_array, 2.5, axis=0)+1
    ###finer_upper_bound = np.percentile(finer_simulations_array, 97.5, axis=0)+1

    df_simulated_data = pd.read_csv(dataName)

    ###plt.plot(finer_dataTime, finer_mean_simulation[:, speciesInd[species]], color=color)
    ###plt.fill_between(finer_dataTime, finer_lower_bound[:, speciesInd[species]], finer_upper_bound[:, speciesInd[species]]
    plt.scatter(df_simulated_data['Time (Hr)'], df_simulated_data[species], label=f'{species} Data', color=color, s=50)
```

```
  plt.xlabel('Time (Hr)', fontsize = 20)
  plt.ylabel('CFU', fontsize = 20)
  plt.yscale('log')
  plt.grid(True)
  plt.xticks(fontsize=18)
  plt.yticks(fontsize=18)
  plt.ylim([1E-1, 1E+10])
```

ChatGPT

```
showDataFit2(0, 0, 25, 'expData_i.csv',
            ['DH5a-X13', 'DH5a-X61', 'DH5a-X13-X61'], ['red', 'green', 'goldenrod'])

showDataFit2(0, 0, 336, 'expData_ii.csv',
            ['DH5a-X13', 'DH5a-X61', 'DH5a-X13-X61'], ['red', 'green', 'goldenrod'])

showDataFit2(0, 0, 336, 'expData_iii.csv',
            ['DH5a-X13', 'DH5a-X61', 'DH5a-X13-X61'], ['red', 'green', 'goldenrod'])

showDataFit2(0, 0, 336, 'expData_iv.csv',
            ['DH5a', 'DH5a-X61'], ['black', 'green'])
```

```
AinitCond = [initial_condition_vDH5a, initial_condition_vDH5a13,
             initial_condition_vDH5a61, initial_condition_vDH5a1361]
AdataName = ['expData_vDH5a.csv', 'expData_vDH5a13.csv',
             'expData_vDH5a61.csv', 'expData_vDH5a1361.csv']

showDataFitGrowth2(0, AinitCond, 125, AdataName,
                  ['DH5a', 'DH5a-X13', 'DH5a-X61', 'DH5a-X13-X61'],
                  ['black', 'red', 'green', 'goldenrod'])
```

Parameter Estimation

Define parameter for Metropolis Monte Carlo

```
# Define the minimal and maximal values for starting parameters
min_values = [1E-13, 0.1, 0.1, 0.1, 0.1, 1E-7, 1E-7, 0.01]
max_values = [1E-7, 10, 10, 10, 10, 10, 10, 10]

proposal_fraction = [0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05] # step size in log scale for Metropolis random walk
iterNum = 20000
burnNum = 5000
```

Full Parameter Estimation (from different initial parameter values)

```
import pandas as pd
import numpy as np


# Read synthetic data from csv
# Assuming you have already imported pandas as pd
df_exp_data = pd.read_csv('expData_i.csv')
data = df_exp_data[['DH5a', 'DH5a-X13', 'DH5a-X61', 'DH5a-X13-X61']].values
times = df_exp_data['Time (Hr)'].values

df_exp_dataLONG = pd.read_csv('expData_ii.csv')
dataLONG = df_exp_dataLONG[['DH5a', 'DH5a-X13', 'DH5a-X61', 'DH5a-X13-X61']].values
timesLONG = df_exp_dataLONG['Time (Hr)'].values

df_exp_DH5aGrowth = pd.read_csv('expData_vDH5a.csv')
dataDH5a = df_exp_DH5aGrowth[['DH5a', 'DH5a-X13', 'DH5a-X61', 'DH5a-X13-X61']].values
timesDH5a = df_exp_DH5aGrowth['Time (Hr)'].values

df_exp_DH5a13Growth = pd.read_csv('expData_vDH5a13.csv')
dataDH5a13 = df_exp_DH5a13Growth[['DH5a', 'DH5a-X13', 'DH5a-X61', 'DH5a-X13-X61']].values
timesDH5a13 = df_exp_DH5a13Growth['Time (Hr)'].values

df_exp_DH5a61Growth = pd.read_csv('expData_vDH5a61.csv')
dataDH5a61 = df_exp_DH5a61Growth[['DH5a', 'DH5a-X13', 'DH5a-X61', 'DH5a-X13-X61']].values
timesDH5a61 = df_exp_DH5a61Growth['Time (Hr)'].values

df_exp_DH5a1361Growth = pd.read_csv('expData_vDH5a1361.csv')
dataDH5a1361 = df_exp_DH5a1361Growth[['DH5a', 'DH5a-X13', 'DH5a-X61', 'DH5a-X13-X61']].values
```

```
timesDH5a1361 = df_exp_DH5a1361Growth['Time (Hr)'].values

noDH5a = [1,2,3]
sDH5a = [0]
sDH5a13 = [1]
sDH5a61 = [2]
sDH5a1361 = [3]


# Run Metropolis 10 times
for i in range(10):
    # Generate random starting parameters for each iteration
    starting_parameters = generate_random_parameters(min_values, max_values)

    # Run Metropolis
    start_time = time.time()

    #ensemble_iterations = metropolis(starting_parameters, proposal_fraction, iterNum,
    #              (conjugationODEs, initial_conditions_i, data, times, noDH5a))

    #ensemble_iterations = metropolis(starting_parameters, proposal_fraction, iterNum,
    #             (conjugationODEs, initial_conditions_i, data, times, noDH5a),
    #             (conjugationODEs, initial_conditions_ii, dataLONG, timesLONG, noDH5a))

    #ensemble_iterations = metropolis(starting_parameters, proposal_fraction, iterNum,
    #              (conjugationODEs, initial_conditions_i, data, times, noDH5a),
    #              (conjugationODEs, initial_condition_vDH5a, dataDH5a, timesDH5a, sDH5a),
    #              (conjugationODEs, initial_condition_vDH5a13, dataDH5a13, timesDH5a13, sDH5a13),
    #              (conjugationODEs, initial_condition_vDH5a61, dataDH5a61, timesDH5a61, sDH5a61),
    #              (conjugationODEs, initial_condition_vDH5a1361, dataDH5a1361, timesDH5a1361, sDH5a1361))

    ensemble_iterations = metropolis(starting_parameters, proposal_fraction, iterNum,
                  (conjugationODEs, initial_conditions_i, data, times, noDH5a),
                  (conjugationODEs, initial_conditions_ii, dataLONG, timesLONG, noDH5a),
                  (conjugationODEs, initial_condition_vDH5a, dataDH5a, timesDH5a, sDH5a),
                  (conjugationODEs, initial_condition_vDH5a13, dataDH5a13, timesDH5a13, sDH5a13),
                  (conjugationODEs, initial_condition_vDH5a61, dataDH5a61, timesDH5a61, sDH5a61),
                  (conjugationODEs, initial_condition_vDH5a1361, dataDH5a1361, timesDH5a1361, sDH5a1361))



    elapsed_time = time.time() - start_time

    # Save ensemble parameters to csv
    ensemble_iterations_pd = pd.DataFrame(
        ensemble_iterations,
        columns=['Iteration', 'Posterior', 'Eta', 'Mu', 'Mu13', 'Mu61', 'Mu1361', 'Kappa13', 'Kappa61', 'D']
    )
    filename = f'ensembleParamsExpMXnDgr_{i+1}.csv'
    ensemble_iterations_pd.to_csv(filename, encoding='utf-8', index=False)

    print(f"Round {i+1} completed in {elapsed_time:.2f} seconds.")
```

Analyze Parameter Ensemble

Visualize Parameter Ensembeles over Metropolis Iteration

```
filenames = ['ensembleParamsExpSHnDgr_1.csv', 'ensembleParamsExpSHnDgr_2.csv', 'ensembleParamsExpSHnDgr_3.csv',
             'ensembleParamsExpSHnDgr_4.csv', 'ensembleParamsExpSHnDgr_5.csv', 'ensembleParamsExpSHnDgr_6.csv',
             'ensembleParamsExpSHnDgr_7.csv', 'ensembleParamsExpSHnDgr_8.csv', 'ensembleParamsExpSHnDgr_9.csv',
             'ensembleParamsExpSHnDgr_10.csv',]
```

```
correct_values = [eta_value, mu_value, mu13_value, mu61_value, mu1361_value, kappa13_value, kappa61_value, D_value]
plot_multiple_files_updated(filenames, correct_values)
```

```
filenames = [  'ensembleParamsExpMXnDgr_6.csv',
             'ensembleParamsExpMXnDgr_7.csv', 'ensembleParamsExpMXnDgr_8.csv', ]
```

```
'ensembleParamsExpMXnDgr_1.csv',
'ensembleParamsExpMXnDgr_3.csv',
'ensembleParamsExpMXnDgr_2.csv',
```

```
'ensembleParamsExpMXnDgr_4.csv',
'ensembleParamsExpMXnDgr_5.csv',
'ensembleParamsExpMXnDgr_10.csv',

## selected
#filenames = ['ensembleParamsExpMXnDgr_1.csv', 'ensembleParamsExpMXnDgr_3.csv', 'ensembleParamsExpMXnDgr_4.csv',
#             'ensembleParamsExpMXnDgr_6.csv', 'ensembleParamsExpMXnDgr_7.csv', 'ensembleParamsExpMXnDgr_8.csv',
#             'ensembleParamsExpMXnDgr_10.csv',]




correct_values = [eta_value, mu_value, mu13_value, mu61_value, mu1361_value, kappa13_value, kappa61_value, D_value]
plot_multiple_files_updated(filenames, correct_values)
```

Simulate Timecourses from Parameter Ensembles

Parameter Ensembles from (i) + (v) training set

```
# get file names from good Metropolis iterations
filenamesSHORT = ['ensembleParamsExpSHnDgr_1.csv', 'ensembleParamsExpSHnDgr_2.csv', 'ensembleParamsExpSHnDgr_3.csv',
             'ensembleParamsExpSHnDgr_4.csv', 'ensembleParamsExpSHnDgr_5.csv', 'ensembleParamsExpSHnDgr_6.csv',
             'ensembleParamsExpSHnDgr_7.csv', 'ensembleParamsExpSHnDgr_8.csv', 'ensembleParamsExpSHnDgr_9.csv',
             'ensembleParamsExpSHnDgr_10.csv',]

# trim, concatenate and select parameter sets from ensembles
ensembleParamsSHORTset = trim_and_concat_csv(filenamesSHORT)
param_Best, selected_params = selParam(ensembleParamsSHORTset, 200)

selected_params_i_v = selected_params

## get best parameter set
params_Best_i_v = ensembleParamsSHORTset.iloc[ensembleParamsSHORTset['Posterior'].idxmax(), 2:].values



# display synthetic data, mean and 95% interval of simulated timecourses from ensembles

showDataFit(selected_params, initial_conditions_i, 25, 'expData_i.csv',
            ['DH5a-X13', 'DH5a-X61', 'DH5a-X13-X61'], ['red', 'green', 'goldenrod'])

showDataFit(selected_params, initial_conditions_ii, 336, 'expData_ii.csv',
            ['DH5a-X13', 'DH5a-X61', 'DH5a-X13-X61'], ['red', 'green', 'goldenrod'])

showDataFit(selected_params, initial_conditions_iii, 336, 'expData_iii.csv',
            ['DH5a-X13', 'DH5a-X61', 'DH5a-X13-X61'], ['red', 'green', 'goldenrod'])

showDataFit(selected_params, initial_conditions_iv, 336, 'expData_iv.csv',
            ['DH5a', 'DH5a-X61'], ['black', 'green'])
```

```
AinitCond = [initial_condition_vDH5a, initial_condition_vDH5a13,
             initial_condition_vDH5a61, initial_condition_vDH5a1361]
AdataName = ['expData_vDH5a.csv', 'expData_vDH5a13.csv',
             'expData_vDH5a61.csv', 'expData_vDH5a1361.csv']

showDataFitGrowth(selected_params, AinitCond, 130, AdataName,
                  ['DH5a', 'DH5a-X13', 'DH5a-X61', 'DH5a-X13-X61'],
                  ['black', 'red', 'green', 'goldenrod'])
```

Parameter Ensembles from (i) +(ii) + (v) training set


```
# get file names from good Metropolis iterations
filenamesMIX = [  'ensembleParamsExpMXnDgr_6.csv',
             'ensembleParamsExpMXnDgr_7.csv', 'ensembleParamsExpMXnDgr_8.csv', ]

# trim, concatenate and select parameter sets from ensembles
ensembleParamsMIXset = trim_and_concat_csv(filenamesMIX)
param_Best, selected_params = selParam(ensembleParamsMIXset, 200)

selected_params_i_ii_v = selected_params
```

```
## get best parameter set
params_Best_i_ii_v = ensembleParamsMIXset.iloc[ensembleParamsMIXset['Posterior'].idxmax(), 2:].values



# display synthetic data, mean and 95% interval of simulated timecourses from ensembles

showDataFit(selected_params, initial_conditions_i, 25, 'expData_i.csv',
            ['DH5a-X13', 'DH5a-X61', 'DH5a-X13-X61'], ['red', 'green', 'goldenrod'])

showDataFit(selected_params, initial_conditions_ii, 336, 'expData_ii.csv',
            ['DH5a-X13', 'DH5a-X61', 'DH5a-X13-X61'], ['red', 'green', 'goldenrod'])

showDataFit(selected_params, initial_conditions_iii, 336, 'expData_iii.csv',
            ['DH5a-X13', 'DH5a-X61', 'DH5a-X13-X61'], ['red', 'green', 'goldenrod'])

showDataFit(selected_params, initial_conditions_iv, 336, 'expData_iv.csv',
            ['DH5a', 'DH5a-X61'], ['black', 'green'])
```

```
AinitCond = [initial_condition_vDH5a, initial_condition_vDH5a13,
             initial_condition_vDH5a61, initial_condition_vDH5a1361]
AdataName = ['expData_vDH5a.csv', 'expData_vDH5a13.csv',
             'expData_vDH5a61.csv', 'expData_vDH5a1361.csv']

showDataFitGrowth(selected_params, AinitCond, 130, AdataName,
                  ['DH5a', 'DH5a-X13', 'DH5a-X61', 'DH5a-X13-X61'],
                  ['black', 'red', 'green', 'goldenrod'])
```

Parameter Distribution and Correlation Analysis

```
xrange = [[-11.5,-8.5], [-1.5, 1], [-0.5,1], [-0.5,1],[-1.5,1], [-18,1], [-18,1], [-4,1]]
np.arange(xrange[1][0], xrange[1][1], (xrange[1][1]-xrange[1][0])/20)
```

```
array([-1.5  , -1.375, -1.25 , -1.125, -1.   , -0.875, -0.75 , -0.625,
       -0.5  , -0.375, -0.25 , -0.125,  0.   ,  0.125,  0.25 ,  0.375,
        0.5  ,  0.625,  0.75 ,  0.875])
```

Double-click (or enter) to edit

Parameter Ensembles from (i) + (v) training set

```
process_and_plot_csv(filenamesSHORT, 5000, 0.020, [], xrange)
```

Parameter Ensembles from (i) + (ii) + (v) training set

```
logAllParamSetSample = process_and_plot_csv(filenamesMIX, 5000, 0.028, [], xrange)
```

```
plt.scatter(logAllParamSetSample['Kappa13'], logAllParamSetSample['Kappa61'])
```

```
plt.scatter(logAllParamSetSample['Mu61'], logAllParamSetSample['D'])
```

```
plt.scatter(logAllParamSetSample['Mu61'], logAllParamSetSample['Mu1361'])
```

Compare Ensemble Mean VS Best Fit

From (i)+(v) training set

```
# display synthetic data, mean simulated timecourses from ensembles

showDataFit_NS(selected_params_i_v, initial_conditions_i, 25, 'expData_i.csv',
            ['DH5a-X13', 'DH5a-X61', 'DH5a-X13-X61'], ['red', 'green', 'goldenrod'])

showDataFit_NS(selected_params_i_v, initial_conditions_ii, 336, 'expData_ii.csv',
            ['DH5a-X13', 'DH5a-X61', 'DH5a-X13-X61'], ['red', 'green', 'goldenrod'])

showDataFit_NS(selected_params_i_v, initial_conditions_iii, 336, 'expData_iii.csv',
            ['DH5a-X13', 'DH5a-X61', 'DH5a-X13-X61'], ['red', 'green', 'goldenrod'])

showDataFit_NS(selected_params_i_v, initial_conditions_iv, 336, 'expData_iv.csv',
            ['DH5a', 'DH5a-X61'], ['black', 'green'])

showDataFitGrowth_NS(selected_params_i_v, AinitCond, 130, AdataName,
                  ['DH5a', 'DH5a-X13', 'DH5a-X61', 'DH5a-X13-X61'],
                  ['black', 'red', 'green', 'goldenrod'])
```

```
aParam_Best_i_v  = np.tile(params_Best_i_v, (10,1))
```