

```
In [1]: 1 import numpy as np
        2 import matplotlib.pyplot as plt
        3 from scipy.stats import norm
        4 import pandas as pd
        5 from scipy.integrate import odeint
        6 from scipy.integrate import solve_ivp
```

Opening predator-prey dataset

```
In [2]: 1 df = pd.read_csv('predator-prey-data.csv', index_col=False)
        2 df.head()
```

```
Out[2]:
```

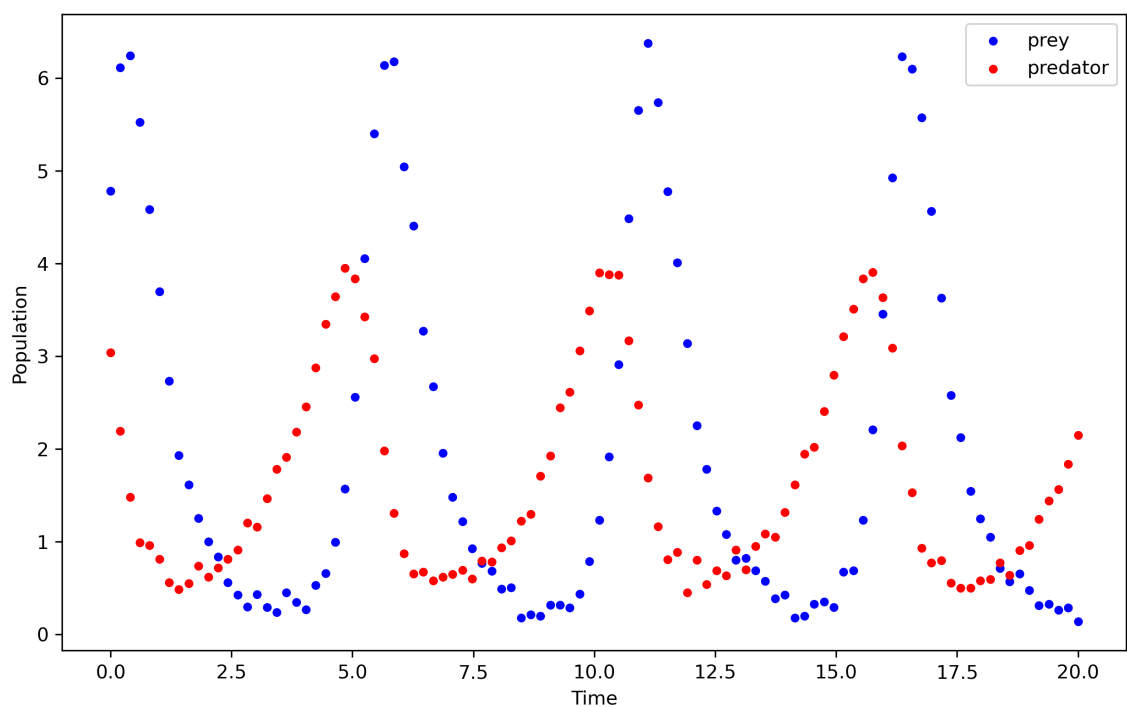
	Unnamed: 0	t	x	y
0	0	0.000000	4.781635	3.035257
1	1	0.202020	6.114005	2.189746
2	2	0.404040	6.238361	1.478907
3	3	0.606061	5.520524	0.989836
4	4	0.808081	4.582546	0.957827

```

In [3]: 1 # Loading data into read-only numpy arrays
2 data = df[['t','x','y']].values
3 # data[1], data[2] = data[2].copy(), data[1].copy()
4 data.flags.writeable = False
5
6
7 # Plotting
8 plt.figure(dpi =300, figsize=(10, 6))
9 point_width = 13
10 plt.scatter(data[:,0], data[:,1], label = 'prey', color = 'blue', s =pc
11 plt.scatter(data[:,0], data[:,2], label = 'predator', color = 'red', s=
12 plt.ylabel('Population')
13 plt.xlabel('Time')
14 plt.legend()
15

```

Out[3]: <matplotlib.legend.Legend at 0x13fc52f10>



Objective functions

Defining volterra equations function

```

In [4]: 1 def predator_prey_odes(initial_conditions,time ,alpha, beta, delta, gamma)
2     x = initial_conditions[0] # initial predator population
3     y = initial_conditions[1] # initial prey population
4     dxdt = (alpha * x) - (beta * x * y) # Prey ODE
5     dydt = (delta * x * y) - (gamma * y) # Predator ODE
6     return [dxdt, dydt]
7

```

```
In [5]: 1 #Function that will return the data for predator and prey for a given s
2 def predator_prey_integration(time,initial_conditions,parameters):
3     alpha,beta,delta,gamma = parameters
4     results = odeint(predator_prey_odes,initial_conditions, time, args=
5     predator_values,prey_values = results[:,0], results[:,1]
6     return np.array([predator_values,prey_values]).T
7
8
```

Defining objective functions

```
In [101]: 1 def weighted_sse(actual, predicted):
2     '''Weighted sum of squared erros'''
3
4     sd = 0.01
5
6     sd_list = sd*actual #List of estimated standard deviations
7
8     inv_sd = 1/sd_list #List of inverted standard deviations from sd_li
9
10    weighted_sse = np.sum(sd_list*((actual - predicted)**2))
11
12    #weighted_sse =
13
14    return weighted_sse
15
16
17 def MAE(actual, predicted):
18     '''Mean absolute error'''
19     return np.mean(np.abs(actual - predicted))
20
21
22
23 def MSE(actual, predicted):
24     '''Mean squared error'''
25     return np.mean((actual - predicted)**2)
```

```
In [7]: 1 test = np.array([1,2,3])
2
3 mult = 3*test
4 print(mult)
5
6 sum = np.sum(mult*test)
7
8 print(sum)
```

```
[3 6 9]
42
```

Algorithms & Optimisation

Simulated Annealing

```

In [8]: 1 def random_walk_annealing(parameters): #A random walk designed for anne
2         lst = [parameter + np.random.normal(0, 0.5) for parameter in paramet
3         # Ensure all elements are positive
4         while any(x <= 0 for x in lst):
5             for indx in range(len(lst)):
6                 if lst[indx] <= 0:
7                     lst[indx] = max(0, parameters[indx] + np.random.normal(
8
9         return lst
10
11
12
13 def simulated_annealing(initial_temp,cooling_constant, data, time, init
14
15     temp = initial_temp #Scaling factor for random movement. We square
16     start = parameters #Initial starting parameters
17     x_n = start
18     scores = [] #A score is just the value of the objective function ev
19
20     current_est = predator_prey_integration(time, initial_conditions, >
21     current_score = objective(data, current_est) #The current value of
22     scores.append(current_score) #Keeping track of the values of the ob
23
24     #cur = function(x) #The function value of the current x solution
25     history = [x_n] #Stores previously searched x values
26
27     for i in range (max_iterations):
28
29         proposal = random_walk_annealing(x_n) #A new proposal for the p
30         new_est = predator_prey_integration(time, initial_conditions, p
31         new_score = objective(data, new_est) #Calculate new value of ob
32
33         delta = new_score - current_score #Difference in objective func
34
35         #if proposal < 0 or proposal > 1:
36             #proposal = x_n # Reject proposal by setting it equal to pre
37
38         acceptance_probability = min(np.exp(-(delta/temp)),1)#Calculate
39
40         #if delta < 0:
41             #x_n = proposal ##Accept proposal
42             #current_score = new_score
43
44         if np.random.rand() < acceptance_probability: #else if it is no
45             x_n = proposal #Accept proposal
46             current_score = new_score
47
48         scores.append(current_score)
49         temp = cooling_constant**i * initial_temp #Cool temperature
50         #print(temp)
51         history.append(x_n) #Add to history
52
53     return x_n, scores

```

In [90]:

```

1 input_data = data[:,1:3]
2 initial_conditions = [input_data[0][0], input_data[0][1]]
3 t = data[:,0]
4
5 #Taking random draw for initial parameters (initial guess)
6 alpha = np.random.uniform(0,1)
7 beta = np.random.uniform(0,1)
8 delta = np.random.uniform(0,1)
9 gamma = np.random.uniform(0,1)
10 parameters = [alpha, beta, delta, gamma]
11 initial_temp = 20
12 cooling_constant = 0.10
13
14 # Using MSE
15 x_best, scores = simulated_annealing(initial_temp,cooling_constant, inp

```

```

/var/folders/rc/tn2ys5g55157vhhlmn0_vfwr0000gq/T/ipykernel_70535/385309489
9.py:38: RuntimeWarning: overflow encountered in exp
    acceptance_probability = min(np.exp(-(delta/temp)),1)#Calculate acceptan
ce probability
/var/folders/rc/tn2ys5g55157vhhlmn0_vfwr0000gq/T/ipykernel_70535/385309489
9.py:38: RuntimeWarning: overflow encountered in scalar divide
    acceptance_probability = min(np.exp(-(delta/temp)),1)#Calculate acceptan
ce probability
/var/folders/rc/tn2ys5g55157vhhlmn0_vfwr0000gq/T/ipykernel_70535/385309489
9.py:38: RuntimeWarning: divide by zero encountered in scalar divide
    acceptance_probability = min(np.exp(-(delta/temp)),1)#Calculate acceptan
ce probability

```

In [184]:

```

1 input_data = data[:,1:3]
2 initial_conditions = [input_data[0][0], input_data[0][1]]
3 t = data[:,0]
4
5 #Taking random draw for initial parameters (initial guess)
6 alpha = np.random.uniform(0,1)
7 beta = np.random.uniform(0,1)
8 delta = np.random.uniform(0,1)
9 gamma = np.random.uniform(0,1)
10 parameters = [alpha, beta, delta, gamma]
11 initial_temp = 20
12 cooling_constant = 0.10
13
14 # Using MSE
15 x_best, scores = simulated_annealing(initial_temp,cooling_constant, inp

```

```

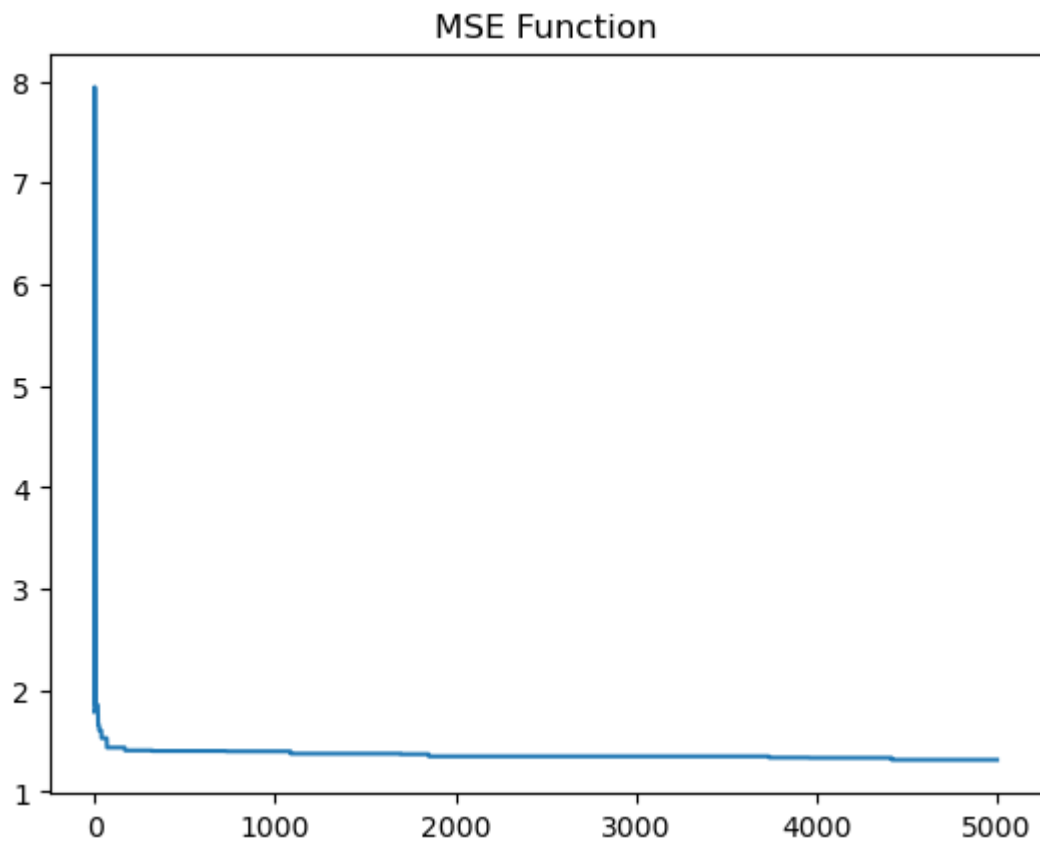
/var/folders/rc/tn2ys5g55157vhhlmn0_vfwr0000gq/T/ipykernel_70535/385309489
9.py:38: RuntimeWarning: overflow encountered in exp
    acceptance_probability = min(np.exp(-(delta/temp)),1)#Calculate acceptan
ce probability
/var/folders/rc/tn2ys5g55157vhhlmn0_vfwr0000gq/T/ipykernel_70535/385309489
9.py:38: RuntimeWarning: overflow encountered in scalar divide
    acceptance_probability = min(np.exp(-(delta/temp)),1)#Calculate acceptan
ce probability
/var/folders/rc/tn2ys5g55157vhhlmn0_vfwr0000gq/T/ipykernel_70535/385309489
9.py:38: RuntimeWarning: divide by zero encountered in scalar divide
    acceptance_probability = min(np.exp(-(delta/temp)),1)#Calculate acceptan
ce probability

```

Solution discovery over iterations

```
In [185]: 1 xvalues = [x for x in range(0,len(scores))]  
2  
3  
4  
5 plt.plot(xvalues,scores)  
6 plt.title("MSE Function")  
7  
8 print("LOWEST MSE: " + str(scores[-1]))
```

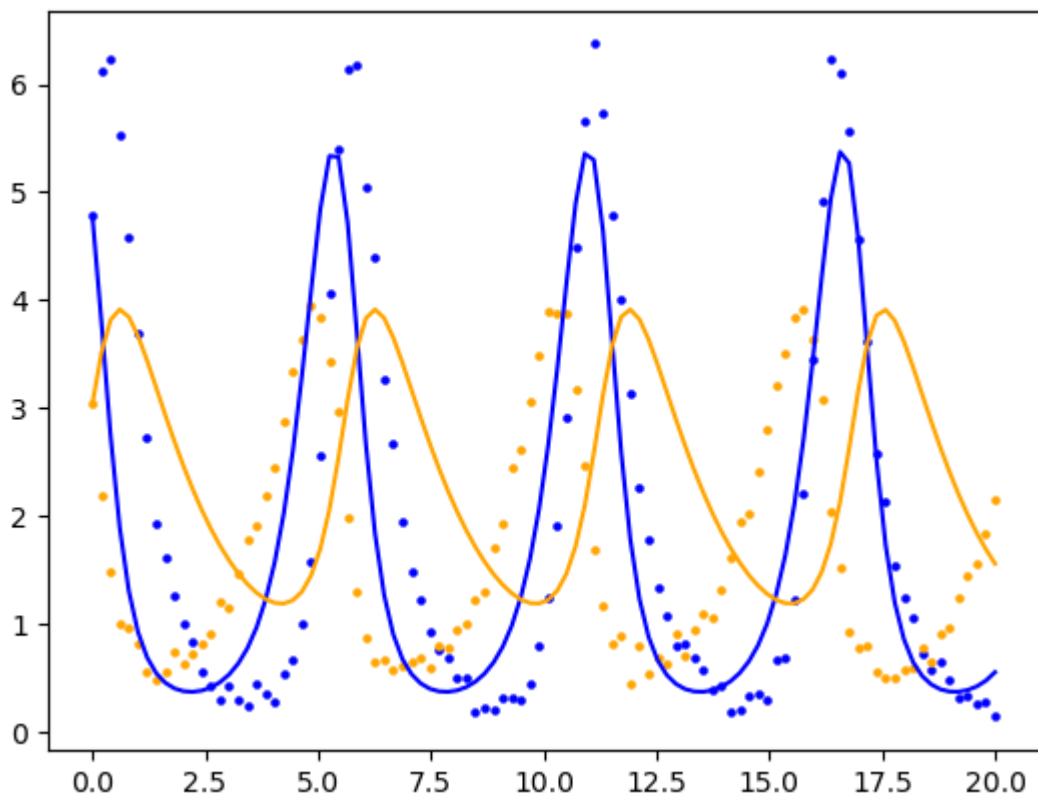
LOWEST MSE: 1.3145243092280952



Curve fit

```
In [186]: 1 # t, x, y = data
2 initial_conditions = [input_data[0][0], input_data[0][1]]
3 t = data[:,0]
4 scaling = 2
5
6 parameters = x_best
7
8 # Using MSE
9 x = predator_prey_integration(t, initial_conditions, parameters)
10
11 plt.figure(dpi=300, figsize=(6, 5))
12 point_width = 13
13 plt.plot(t, x[:,0], color="b")
14 plt.plot(t, x[:,1], color="orange")
15
16 plt.scatter(t, data[:,1], color='blue', s=5)
17 plt.scatter(t, data[:,2], color='orange', s=5)
18
19 #mse_pre = MSE(data[:,1], x[:,0]) #MSE for fitted curve
20 #mse_predator = MSE(data[:,2], x[:,1])
21 #mse_total = mse_pre + mse_predator
22 print("MAE: " + str(scores[-1]))
```

MAE: 1.3145243092280952



Calculation of mean and variance

```
In [96]: 1 ### Distribution of parameters for multiple runs
2
3 def multiple_runs_annealing(initial_temp,cooling_constant,input_data,t,
4
5     mse_total_list = []
6
7
8     for i in range(n_runs):
9
10         x_best, scores = simulated_annealing(initial_temp,cooling_const
11
12         mse = scores[-1]
13
14         mse_total_list.append(mse) #Add total MSE for this simulation t
15
16     return mse_total_list
17
```

```
In [106]: 1 input_data = data[:,1:3]
2 initial_conditions = [input_data[0][0], input_data[0][1]]
3 t = data[:,0]
4 #Taking random draw for initial parameters (initial guess)
5 alpha = np.random.uniform(0,1)
6 beta = np.random.uniform(0,1)
7 delta = np.random.uniform(0,1)
8 gamma = np.random.uniform(0,1)
9 parameters = [alpha, beta, delta, gamma]
10 parameters = [alpha, beta, delta, gamma]
11
12 initial_temp = 20
13 cooling_constant = 0.10
14
15 mse_total_list = multiple_runs_annealing(initial_temp,cooling_constant,
16
17 #print(mse_total_list)
```

```
/var/folders/rc/tn2ys5g55157vhhlmn0_vfwr0000gq/T/ipykernel_58892/385309489
9.py:38: RuntimeWarning: overflow encountered in exp
    acceptance_probability = min(np.exp(-(delta/temp)),1)#Calculate acceptan
ce probability
/var/folders/rc/tn2ys5g55157vhhlmn0_vfwr0000gq/T/ipykernel_58892/385309489
9.py:38: RuntimeWarning: overflow encountered in scalar divide
    acceptance_probability = min(np.exp(-(delta/temp)),1)#Calculate acceptan
ce probability
/var/folders/rc/tn2ys5g55157vhhlmn0_vfwr0000gq/T/ipykernel_58892/385309489
9.py:38: RuntimeWarning: divide by zero encountered in scalar divide
    acceptance_probability = min(np.exp(-(delta/temp)),1)#Calculate acceptan
ce probability
/Users/alex_1/anaconda3/lib/python3.11/site-packages/scipy/integrate/_odep
ack_py.py:248: ODEintWarning: Excess work done on this call (perhaps wrong
Dfun type). Run with full_output = 1 to get quantitative information.
    warnings.warn(warning_msg, ODEintWarning)
```

```
In [107]: 1 print(mse_total_list)

[2.6473227546360665, 2.7464957827676564, 2.8201901313461293, 2.68480725254
8282, 2.6589224841462658, 2.5022908556330266, 2.4583578426621084, 2.439019
9669793287, 2.460643513633285, 2.543829159819853]
```


In [108]:

```

1
2 mean_mse_annealing = np.mean(mse_total_list)
3 std_mse_annealing = np.std(mse_total_list)
4
5 print("Average MSE = " + str(mean_mse_annealing))
6 print("Standard deviation of MSE = " + str(std_mse_annealing))

```

Average MSE = 2.5961879744172007

Standard deviation of MSE = 0.12680792857239243

FOR WEIGHTED SSE

In [189]:

```

1
2
3 input_data = data[:,1:3]
4 initial_conditions = [input_data[0][0], input_data[0][1]]
5 t = data[:,0]
6 #Taking random draw for initial parameters (initial guess)
7 alpha = np.random.uniform(0,1)
8 beta = np.random.uniform(0,1)
9 delta = np.random.uniform(0,1)
10 gamma = np.random.uniform(0,1)
11 parameters = [alpha, beta, delta, gamma]
12 parameters = [alpha, beta, delta, gamma]
13
14 initial_temp = 20
15 cooling_constant = 0.10
16
17 mse_total_list = multiple_runs_annealing(initial_temp,cooling_constant,
18

```

```

/var/folders/rc/tn2ys5g55157vhhlmn0_vfwr0000gq/T/ipykernel_70535/385309489
9.py:38: RuntimeWarning: overflow encountered in exp

```

```

    acceptance_probability = min(np.exp(-(delta/temp)),1)#Calculate acceptance probability

```

```

/var/folders/rc/tn2ys5g55157vhhlmn0_vfwr0000gq/T/ipykernel_70535/385309489
9.py:38: RuntimeWarning: overflow encountered in scalar divide

```

```

    acceptance_probability = min(np.exp(-(delta/temp)),1)#Calculate acceptance probability

```

```

/var/folders/rc/tn2ys5g55157vhhlmn0_vfwr0000gq/T/ipykernel_70535/385309489
9.py:38: RuntimeWarning: divide by zero encountered in scalar divide

```

```

    acceptance_probability = min(np.exp(-(delta/temp)),1)#Calculate acceptance probability

```

In [190]:

```

1 print(mse_total_list)
2 print(np.std(mse_total_list))

```

```

[1.4283946885408745, 1.3067741402910857, 1.4442398586927112, 1.28940092054
71197, 1.4204590248705167, 1.4440930616053458, 1.3031423818937582, 1.35896
96758657763, 1.3465599587990993, 1.3043258816810026]
0.06051240976134452

```

Comparison of cooling schedules

```
In [221]: 1 input_data = data[:,1:3]
2 initial_conditions = [input_data[0][0], input_data[0][1]]
3 t = data[:,0]
4 #Taking random draw for initial parameters (initial guess)
5 alpha = np.random.uniform(0,1)
6 beta = np.random.uniform(0,1)
7 delta = np.random.uniform(0,1)
8 gamma = np.random.uniform(0,1)
9 parameters = [alpha, beta, delta, gamma]
10 parameters = [alpha, beta, delta, gamma]
11
12 initial_temp = 20
13
14 cooling_constants = np.arange(0.10,1,0.10)
15
16 mean_mse_per_simulation = [] #The mean MSE per simulation
17 sd_mse_per_simulation = [] #The standard deviation per simulation
18
19 for constant in cooling_constants:
20
21     mse_total_list = multiple_runs_annealing(initial_temp,constant,input_data)
22     mean_mse_annealing = np.mean(mse_total_list)
23     std_mse_annealing = np.std(mse_total_list)
24     mean_mse_per_simulation.append(mean_mse_annealing)
25     sd_mse_per_simulation.append(std_mse_annealing)
26
27
```

```
/var/folders/rc/tn2ys5g55157vhhlmn0_vfwr0000gq/T/ipykernel_70535/385309489
9.py:38: RuntimeWarning: overflow encountered in exp
  acceptance_probability = min(np.exp(-(delta/temp)),1)#Calculate acceptance probability
/Users/alex_1/anaconda3/lib/python3.11/site-packages/scipy/integrate/_odepack_py.py:248: ODEintWarning: Excess work done on this call (perhaps wrong Dfun type). Run with full_output = 1 to get quantitative information.
  warnings.warn(warning_msg, ODEintWarning)
/var/folders/rc/tn2ys5g55157vhhlmn0_vfwr0000gq/T/ipykernel_70535/385309489
9.py:38: RuntimeWarning: overflow encountered in scalar divide
  acceptance_probability = min(np.exp(-(delta/temp)),1)#Calculate acceptance probability
/var/folders/rc/tn2ys5g55157vhhlmn0_vfwr0000gq/T/ipykernel_70535/385309489
9.py:38: RuntimeWarning: divide by zero encountered in scalar divide
  acceptance_probability = min(np.exp(-(delta/temp)),1)#Calculate acceptance probability
```

In [222]:

```

1 mean_mae_per_simulation = []
2 sd_mae_per_simulation = []
3
4 for constant in cooling_constants:
5
6     mae_total_list = multiple_runs_annealing(initial_temp, constant, input_data)
7     mean_mae_annealing = np.mean(mae_total_list)
8     std_mae_annealing = np.std(mae_total_list)
9     mean_mae_per_simulation.append(mean_mae_annealing)
10    sd_mae_per_simulation.append(std_mae_annealing)
11

```

```

/var/folders/rc/tn2ys5g55157vhhlmn0_vfwr0000gq/T/ipykernel_70535/385309489
9.py:38: RuntimeWarning: overflow encountered in exp
  acceptance_probability = min(np.exp(-(delta/temp)),1)#Calculate acceptance probability
/var/folders/rc/tn2ys5g55157vhhlmn0_vfwr0000gq/T/ipykernel_70535/385309489
9.py:38: RuntimeWarning: overflow encountered in scalar divide
  acceptance_probability = min(np.exp(-(delta/temp)),1)#Calculate acceptance probability
/var/folders/rc/tn2ys5g55157vhhlmn0_vfwr0000gq/T/ipykernel_70535/385309489
9.py:38: RuntimeWarning: divide by zero encountered in scalar divide
  acceptance_probability = min(np.exp(-(delta/temp)),1)#Calculate acceptance probability

```

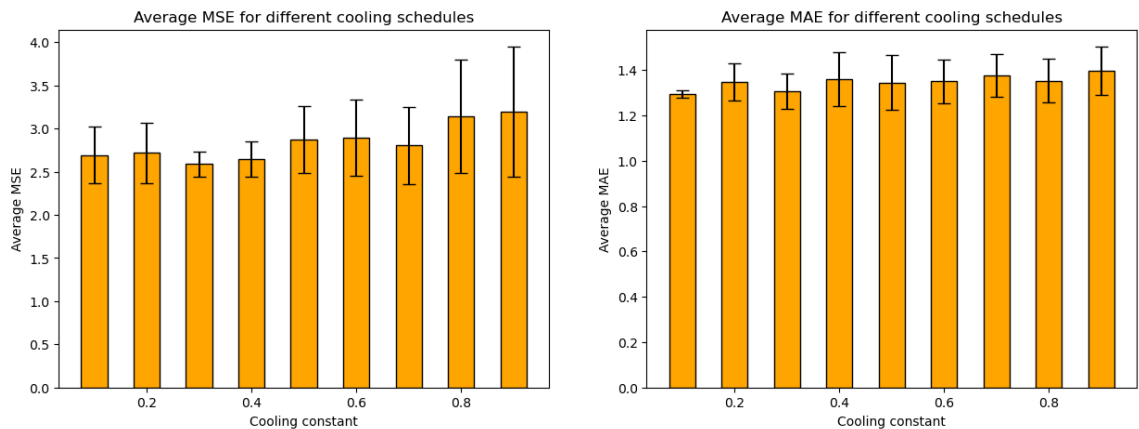
In [223]:

```

1  fig, axes = plt.subplots(1, 2, figsize=(15, 5))
2
3
4  bar_width = 0.05
5
6  axes[0].bar(cooling_constants, mean_mse_per_simulation, yerr = sd_mse_per_simulation)
7  axes[1].bar(cooling_constants, mean_mae_per_simulation, yerr = sd_mae_per_simulation)
8
9  axes[0].set_xlabel("Cooling constant")
10 axes[1].set_xlabel("Cooling constant")
11
12 axes[0].set_ylabel("Average MSE")
13 axes[1].set_ylabel("Average MAE")
14
15 axes[0].set_title("Average MSE for different cooling schedules")
16 axes[1].set_title("Average MAE for different cooling schedules")
17
18
19
20 #plt.bar(cooling_constants, mean_per_simulation, yerr = sd_per_simulation)
21 #plt.ylabel("Average MSE")
22 #plt.xlabel("Cooling constant")
23 #plt.title("Average MSE for different cooling schedules (30 simulations)")
24
25 #plt.show()
26

```

Out[223]: Text(0.5, 1.0, 'Average MAE for different cooling schedules')



Hill climbing

In [238]:

```

1
2
3 def random_walk(parameters):
4     lst = [parameter + np.random.normal(0, 0.5) for parameter in paramet
5     # Ensure all elements are positive
6     while any(x <= 0 for x in lst):
7         for indx in range(len(lst)):
8             if lst[indx] <= 0:
9                 lst[indx] = max(0, parameters[indx] + np.random.normal(
10
11     return lst
12
13 def hill_climbing(data, time, initial_conditions, parameters, objective
14     '''Tries to find the best solution using random walker'''
15     # Initialize starting parameter state
16     scores = []
17     x_n = parameters
18
19     current_est = predator_prey_integration(time, initial_conditions, x
20     current_score = objective(data, current_est)
21     scores.append(current_score)
22
23     for _ in range(max_iterations):
24         # Generate a random walk for parameters
25         x_n_1 = random_walk(x_n)
26
27         # Calculate the current and next estimations
28         current_est = predator_prey_integration(time, initial_conditions, x
29         new_estimation = predator_prey_integration(time, initial_conditions, x
30         new_score = objective(data, new_estimation)
31
32         # If the next estimation is better, update the parameters
33         if new_score < current_score:
34             current_score = new_score
35             x_n = x_n_1
36             scores.append(current_score)
37
38     return x_n, scores
39

```

In [239]:

```

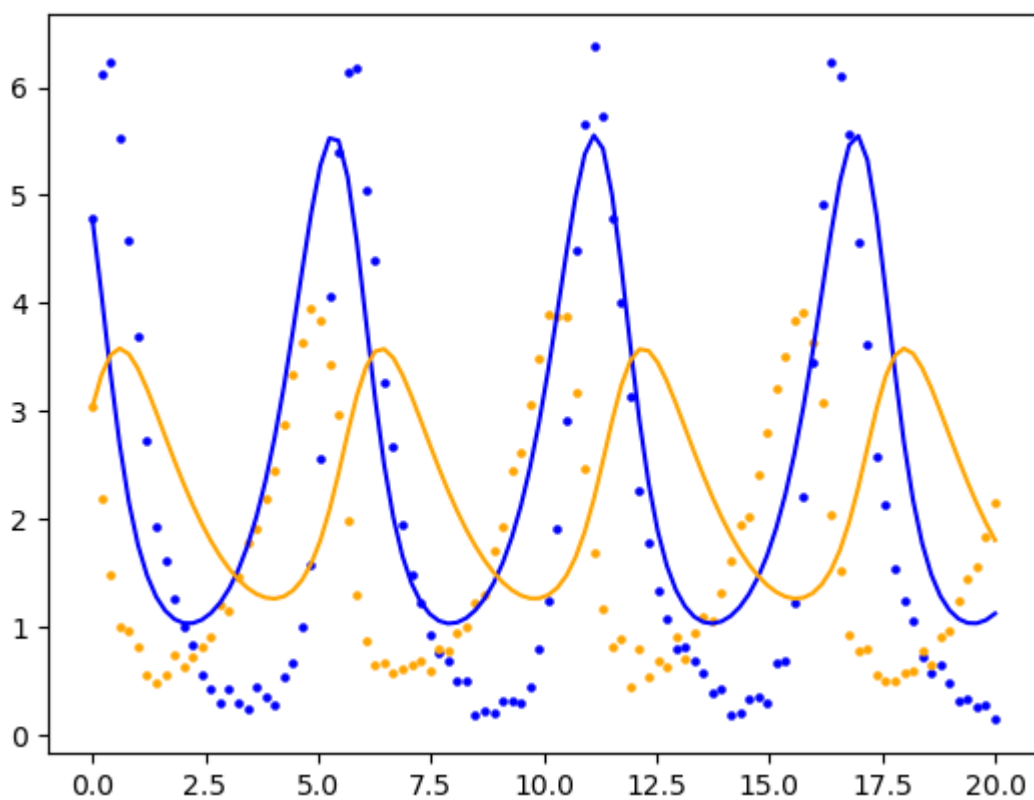
1 input_data = data[:,1:3]
2 initial_conditions = [input_data[0][0], input_data[0][1]]
3 t = data[:,0]
4 #Taking random draw for initial parameters (initial guess)
5 alpha = np.random.uniform(0,1)
6 beta = np.random.uniform(0,1)
7 delta = np.random.uniform(0,1)
8 gamma = np.random.uniform(0,1)
9 parameters = [alpha, beta, delta, gamma]
10
11 # Using MSE
12 x_best,scores = hill_climbing(input_data, t, input_data[0], parameters,

```

Curve fit

```
In [241]: 1 # t, x ,y = data
2 initial_conditions = [input_data[0][0], input_data[0][1]]
3 t = data[:,0]
4
5 parameters = x_best
6
7 # Using MSE
8 x = predator_prey_integration(t,initial_conditions,parameters)
9
10
11
12 point_width = 13
13 plt.plot(t, x[:,0],color = "b")
14 plt.plot(t, x[:,1],color = "orange")
15
16 plt.scatter(t, data[:,1], color= 'blue', s = 5)
17 plt.scatter(t, data[:,2], color= 'orange', s = 5)
18
19 mse_pre = MSE(data[:,1],x[:,0]) #MSE for fitted curve
20 mse_predator = MSE(data[:,2],x[:,1])
21 mse_total = mse_pre + mse_predator
22 print("Mean Square Error: " +str(mse_total))
23
```

Mean Square Error: 5.385424572357688



```
In [195]: 1 def multiple_runs_hill_climbing(input_data, t, initial_conditions, para
2
3     mse_total_list = []
4
5     for i in range(n_runs):
6
7         x_best = hill_climbing(input_data, t, input_data[0], parameters
8
9         x = predator_prey_integration(t, initial_conditions, x_best)
10
11         mse_prey = MSE(data[:,1], x[:,0])
12         mse_predator = MSE(data[:,2], x[:,1])
13         mse_total = mse_prey + mse_predator
14
15         mse_total_list.append(mse_total) #Add total MSE for this simul
16
17     return mse_total_list
18
```

```
In [32]: 1 mean_mse_hill_climbing = np.mean(mse_total_list)
2 std_mse_hill_climbing = np.std(mse_total_list)
3
4 print("Average MSE = " + str(mean_mse_hill_climbing))
5 print("Standard deviation of MSE = " + str(std_mse_hill_climbing))
```

Average MSE = 7.020538795747848

Standard deviation of MSE = 0.525718488225756

Comparison of Optimisation Algorithms

MSE objective function

```
In [228]: 1 input_data = data[:,1:3]
2 initial_conditions = [input_data[0][0], input_data[0][1]]
3 t = data[:,0]
4 #Taking random draw for initial parameters (initial guess)
5 alpha = np.random.uniform(0,1)
6 beta = np.random.uniform(0,1)
7 delta = np.random.uniform(0,1)
8 gamma = np.random.uniform(0,1)
9 parameters = [alpha, beta, delta, gamma]
10 parameters = [alpha, beta, delta, gamma]
11
12 initial_temp = 20
13 cooling_constant = 0.10
14
15 mse_total_list_annealing = multiple_runs_annealing(initial_temp,cooling
16 mean_mse_annealing = np.mean(mse_total_list_annealing)
17 std_mse_annealing = np.std(mse_total_list_annealing)
18
```

```
/var/folders/rc/tn2ys5g55157vhhlmn0_vfwr0000gq/T/ipykernel_70535/385309489
9.py:38: RuntimeWarning: overflow encountered in exp
    acceptance_probability = min(np.exp(-(delta/temp)),1)#Calculate acceptan
ce probability
/var/folders/rc/tn2ys5g55157vhhlmn0_vfwr0000gq/T/ipykernel_70535/385309489
9.py:38: RuntimeWarning: overflow encountered in scalar divide
    acceptance_probability = min(np.exp(-(delta/temp)),1)#Calculate acceptan
ce probability
/var/folders/rc/tn2ys5g55157vhhlmn0_vfwr0000gq/T/ipykernel_70535/385309489
9.py:38: RuntimeWarning: divide by zero encountered in scalar divide
    acceptance_probability = min(np.exp(-(delta/temp)),1)#Calculate acceptan
ce probability
/Users/alex_1/anaconda3/lib/python3.11/site-packages/scipy/integrate/_odep
ack_py.py:248: ODEintWarning: Excess work done on this call (perhaps wrong
Dfun type). Run with full_output = 1 to get quantitative information.
    warnings.warn(warning_msg, ODEintWarning)
```

```
In [229]: 1 input_data = data[:,1:3]
2 initial_conditions = [input_data[0][0], input_data[0][1]]
3 t = data[:,0]
4 #Taking random draw for initial parameters (initial guess)
5 alpha = np.random.uniform(0,1)
6 beta = np.random.uniform(0,1)
7 delta = np.random.uniform(0,1)
8 gamma = np.random.uniform(0,1)
9 parameters = [alpha, beta, delta, gamma]
10
11
12 mse_total_list_hill = multiple_runs_hill_climbing(input_data, t, initia
13 mean_mse_hill_climbing = np.mean(mse_total_list_hill)
14 std_mse_hill_climbing = np.std(mse_total_list_hill)
```

MAE objective function

In [230]:

```

1 input_data = data[:,1:3]
2 initial_conditions = [input_data[0][0], input_data[0][1]]
3 t = data[:,0]
4 #Taking random draw for initial parameters (initial guess)
5 alpha = np.random.uniform(0,1)
6 beta = np.random.uniform(0,1)
7 delta = np.random.uniform(0,1)
8 gamma = np.random.uniform(0,1)
9 parameters = [alpha, beta, delta, gamma]
10 parameters = [alpha, beta, delta, gamma]
11
12 initial_temp = 20
13 cooling_constant = 0.10
14
15 mae_total_list_annealing = multiple_runs_annealing(initial_temp,cooling
16 mean_mae_annealing = np.mean(mae_total_list_annealing)
17 std_mae_annealing = np.std(mae_total_list_annealing)

```

```

/var/folders/rc/tn2ys5g55157vhhlmn0_vfwr0000gq/T/ipykernel_70535/385309489
9.py:38: RuntimeWarning: overflow encountered in exp
    acceptance_probability = min(np.exp(-(delta/temp)),1)#Calculate acceptan
ce probability
/var/folders/rc/tn2ys5g55157vhhlmn0_vfwr0000gq/T/ipykernel_70535/385309489
9.py:38: RuntimeWarning: overflow encountered in scalar divide
    acceptance_probability = min(np.exp(-(delta/temp)),1)#Calculate acceptan
ce probability
/var/folders/rc/tn2ys5g55157vhhlmn0_vfwr0000gq/T/ipykernel_70535/385309489
9.py:38: RuntimeWarning: divide by zero encountered in scalar divide
    acceptance_probability = min(np.exp(-(delta/temp)),1)#Calculate acceptan
ce probability

```

In [231]:

```

1 input_data = data[:,1:3]
2 initial_conditions = [input_data[0][0], input_data[0][1]]
3 t = data[:,0]
4 #Taking random draw for initial parameters (initial guess)
5 alpha = np.random.uniform(0,1)
6 beta = np.random.uniform(0,1)
7 delta = np.random.uniform(0,1)
8 gamma = np.random.uniform(0,1)
9 parameters = [alpha, beta, delta, gamma]
10
11
12 mae_total_list_hill = multiple_runs_hill_climbing(input_data, t, initia
13 mean_mae_hill_climbing = np.mean(mae_total_list_hill)
14 std_mae_hill_climbing = np.std(mae_total_list_hill)

```

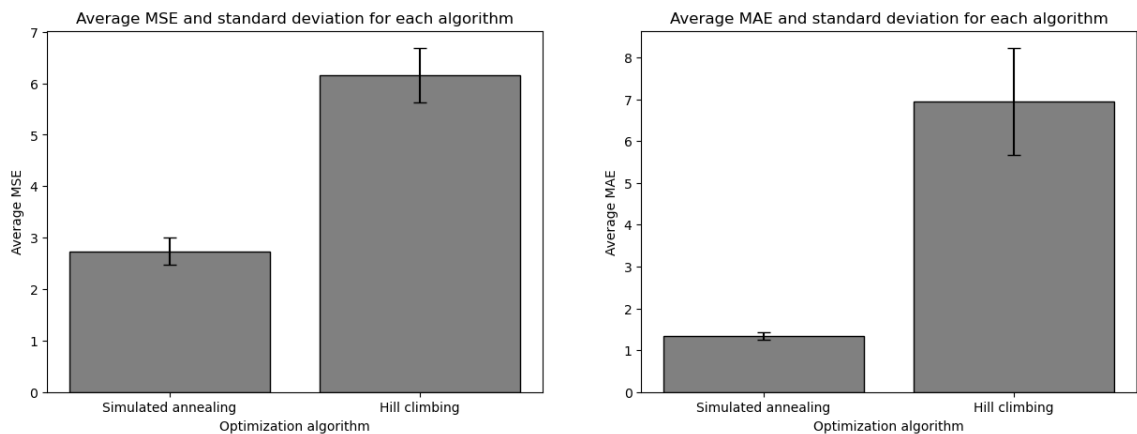
In [237]:

```

1  ##PLOTTING
2
3  fig, axes = plt.subplots(1, 2, figsize=(15, 5))
4
5
6  x = ["Simulated annealing", "Hill climbing"]
7  mse_list = [mean_mse_annealing, mean_mse_hill_climbing] #Stores the mean
8  std_mse_list = [std_mse_annealing, std_mse_hill_climbing]
9
10 mae_list = [mean_mae_annealing, mean_mae_hill_climbing] #Stores the mean
11 std_mae_list = [std_mae_annealing, std_mae_hill_climbing]
12
13 axes[0].bar(x, mse_list, yerr = std_mse_list, color = "grey", ec = "black")
14 axes[1].bar(x, mae_list, yerr = std_mae_list, color = "grey", ec = "black")
15
16
17
18 axes[0].set_xlabel("Optimization algorithm")
19 axes[1].set_xlabel("Optimization algorithm")
20
21 axes[0].set_ylabel("Average MSE")
22 axes[1].set_ylabel("Average MAE")
23
24 axes[0].set_title("Average MSE and standard deviation for each algorithm")
25 axes[1].set_title("Average MAE and standard deviation for each algorithm")
26
27
28
29 #plt.bar(x, mse_list, yerr = std_list, color = "grey", ec = "black", ecol
30 #plt.title("Bar plot comparing the means and errors of the MSE")
31 #plt.ylabel("MSE")
32 #plt.xlabel("Optimization algorithm")

```

Out[237]: Text(0.5, 1.0, 'Average MAE and standard deviation for each algorithm')



In []:

1

In []:

1

```
In [3]: 1 import numpy as np
        2 import matplotlib.pyplot as plt
        3 from scipy.stats import norm
        4 import pandas as pd
        5 from scipy.integrate import odeint
        6 from scipy.integrate import solve_ivp
        7 import statistics
        8 import random
        9 import scipy.stats
```

Opening predator-prey dataset

```
In [4]: 1 df = pd.read_csv('predator-prey-data.csv', index_col=False)
        2 df.head()
        3
```

```
Out[4]:
```

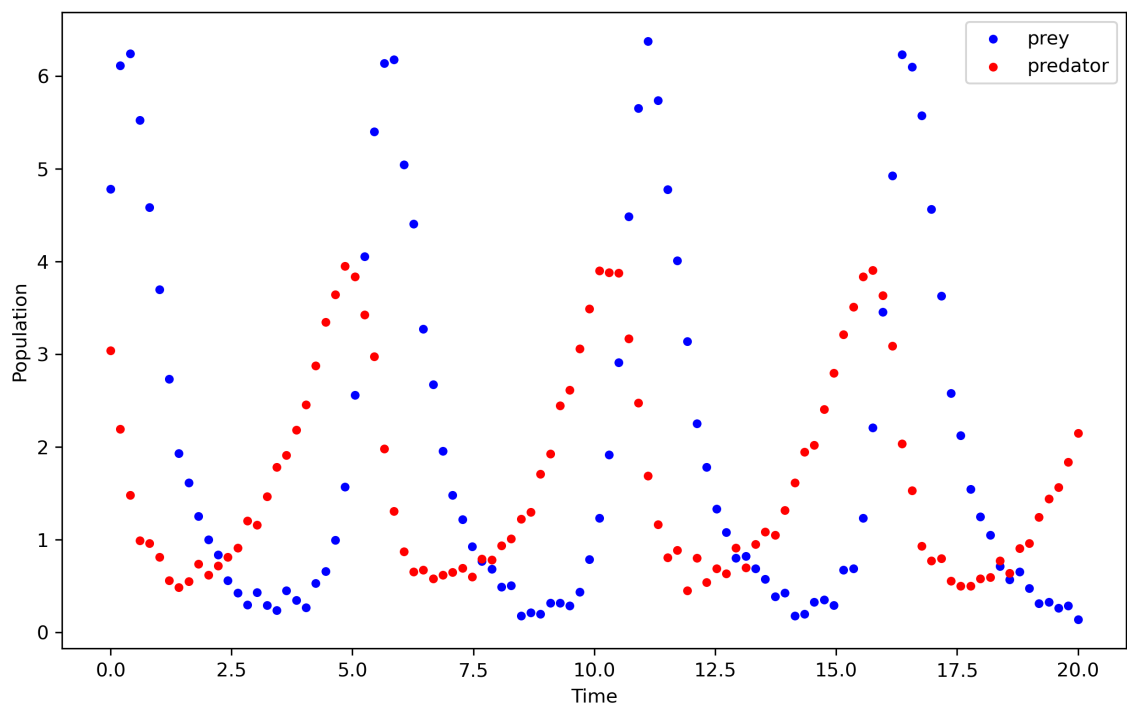
	Unnamed: 0	t	x	y
0	0	0.000000	4.781635	3.035257
1	1	0.202020	6.114005	2.189746
2	2	0.404040	6.238361	1.478907
3	3	0.606061	5.520524	0.989836
4	4	0.808081	4.582546	0.957827

```

In [5]: 1 # Loading data into read-only numpy arrays
2 data = df[['t','x','y']].values
3 # data[1], data[2] = data[2].copy(), data[1].copy()
4 data.flags.writeable = False
5
6
7 # Plotting
8 plt.figure(dpi =300, figsize=(10, 6))
9 point_width = 13
10 plt.scatter(data[:,0], data[:,1], label = 'prey', color = 'blue', s =pc
11 plt.scatter(data[:,0], data[:,2], label = 'predator', color = 'red', s=
12 plt.ylabel('Population')
13 plt.xlabel('Time')
14 plt.legend()
15

```

Out[5]: <matplotlib.legend.Legend at 0x1108369be50>



Objective functions

Defining volterra equations function

```

In [6]: 1 def predator_prey_odes(initial_conditions,time ,alpha, beta, delta, gamma)
2     x = initial_conditions[0] # initial predator population
3     y = initial_conditions[1] # initial prey population
4     dxdt = (alpha * x) - (beta * x * y) # Prey ODE
5     dydt = (delta * x * y) - (gamma * y) # Predator ODE
6     return [dxdt, dydt]
7

```

```
In [7]: 1 #Function that will return the data for predator and prey for a given s
2 def predator_prey_integration(time,initial_conditions,parameters):
3     alpha,beta,delta,gamma = parameters
4     results = odeint(predator_prey_odes,initial_conditions, time, args=
5     predator_values,prey_values = results[:,0], results[:,1]
6     return np.array([predator_values,prey_values]).T
7
8
```

Defining objective functions

```
In [8]: 1 def negative_log_likelihood(actual, predicted, variance=1.0):
2     '''Log likelyhood function'''
3     # Assuming a normal distribution for simplicity
4     log_likelihoods1 =norm.logpdf(actual[0], loc=predicted[0], scale=np
5     log_likelihoods2 = norm.logpdf(actual[1], loc=predicted[1], scale=r
6
7     return -np.sum((log_likelihoods1,log_likelihoods2))
8
9 def MSE(actual, predicted):
10     '''Mean squared error'''
11     return np.mean((actual - predicted)**2)
```

Algorithms & Optimisation

Simulated Annealing

```

In [9]: 1 def random_walk_annealing(parameters): #A random walk designed for anne
2         lst = [parameter + np.random.normal(0, 0.5) for parameter in paramet
3         # Ensure all elements are positive
4         while any(x <= 0 for x in lst):
5             for indx in range(len(lst)):
6                 if lst[indx] <= 0:
7                     lst[indx] = max(0, parameters[indx] + np.random.normal(
8
9         return lst
10
11
12
13 def simulated_annealing(initial_temp,cooling_constant, data, time, init
14
15     temp = initial_temp #Scaling factor for random movement. We square
16     start = parameters #Initial starting parameters
17     x_n = start
18     scores = [] #A score is just the value of the objective function ev
19
20     current_est = predator_prey_integration(time, initial_conditions, >
21     current_score = objective(data, current_est) #The current value of
22     scores.append(current_score) #Keeping track of the values of the ob
23
24     #cur = function(x) #The function value of the current x solution
25     history = [x_n] #Stores previously searched x values
26
27     for i in range (max_iterations):
28
29         proposal = random_walk_annealing(x_n) #A new proposal for the p
30         new_est = predator_prey_integration(time, initial_conditions, p
31         new_score = objective(data, new_est) #Calculate new value of ob
32
33         delta = new_score - current_score #Difference in objective func
34
35         #if proposal < 0 or proposal > 1:
36             #proposal = x_n # Reject proposal by setting it equal to pre
37
38         acceptance_probability = min(np.exp(-(delta/temp)),1)#Calculate
39
40         #if delta < 0:
41             #x_n = proposal ##Accept proposal
42             #current_score = new_score
43
44         if np.random.rand() < acceptance_probability: #else if it is no
45             x_n = proposal #Accept proposal
46             current_score = new_score
47
48         scores.append(current_score)
49         temp = cooling_constant**i * initial_temp #Cool temperature
50         #print(temp)
51         history.append(x_n) #Add to history
52
53     return x_n, scores

```

```

In [8]: 1 input_data = data[:,1:3]
        2 initial_conditions = [input_data[0][0], input_data[0][1]]
        3 t = data[:,0]
        4
        5 #Taking random draw for initial parameters (initial guess)
        6 alpha = np.random.uniform(0,1)
        7 beta = np.random.uniform(0,1)
        8 delta = np.random.uniform(0,1)
        9 gamma = np.random.uniform(0,1)
       10 parameters = [alpha, beta, delta, gamma]
       11 initial_temp = 20
       12 cooling_constant = 0.10
       13
       14 # Using MSE
       15 x_best, scores = simulated_annealing(initial_temp,cooling_constant, inp

```

C:\Users\kayad\AppData\Local\Temp\ipykernel_27100\3853094899.py:38: RuntimeWarning: overflow encountered in exp

acceptance_probability = min(np.exp(-(delta/temp)),1)#Calculate acceptance probability

C:\Users\kayad\AppData\Local\Temp\ipykernel_27100\3853094899.py:38: RuntimeWarning: overflow encountered in scalar divide

acceptance_probability = min(np.exp(-(delta/temp)),1)#Calculate acceptance probability

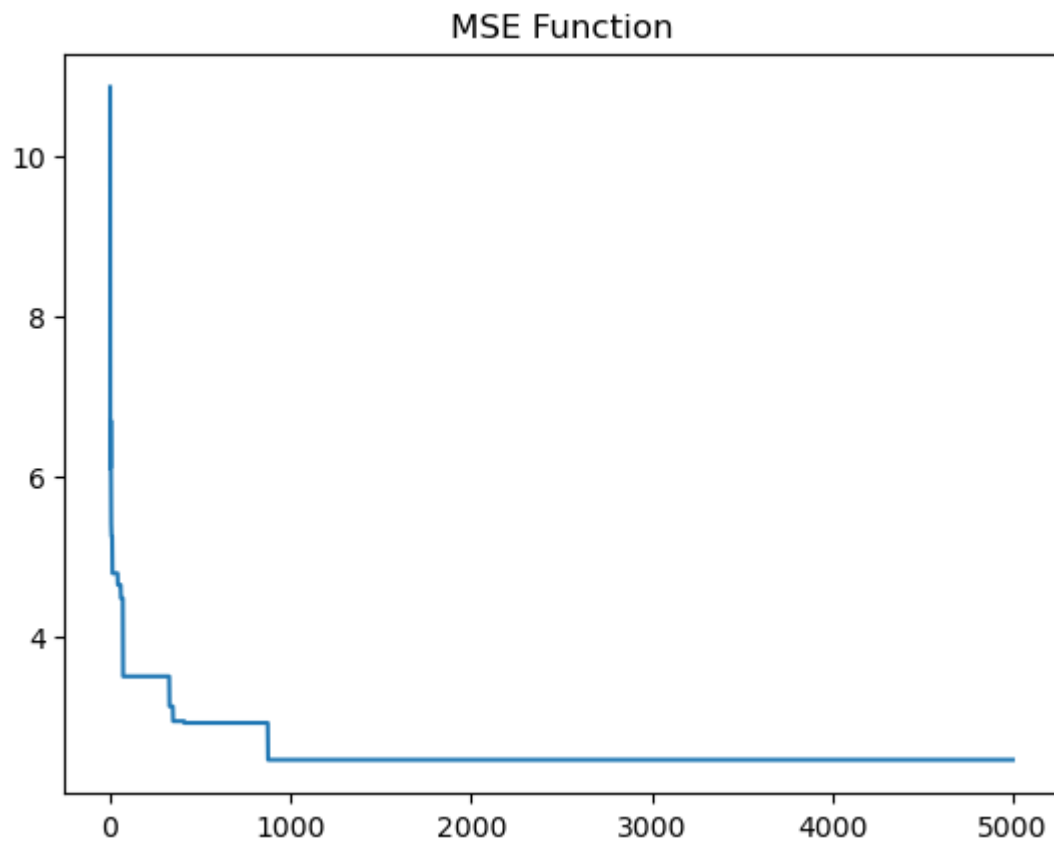
C:\Users\kayad\AppData\Local\Temp\ipykernel_27100\3853094899.py:38: RuntimeWarning: divide by zero encountered in scalar divide

acceptance_probability = min(np.exp(-(delta/temp)),1)#Calculate acceptance probability

Solution discovery over iterations

```
In [9]: 1 xvalues = [x for x in range(0,len(scores))]  
2  
3  
4  
5 plt.plot(xvalues,scores)  
6 plt.title("MSE Function")  
7  
8 print("LOWEST MSE: " + str(scores[-1]))
```

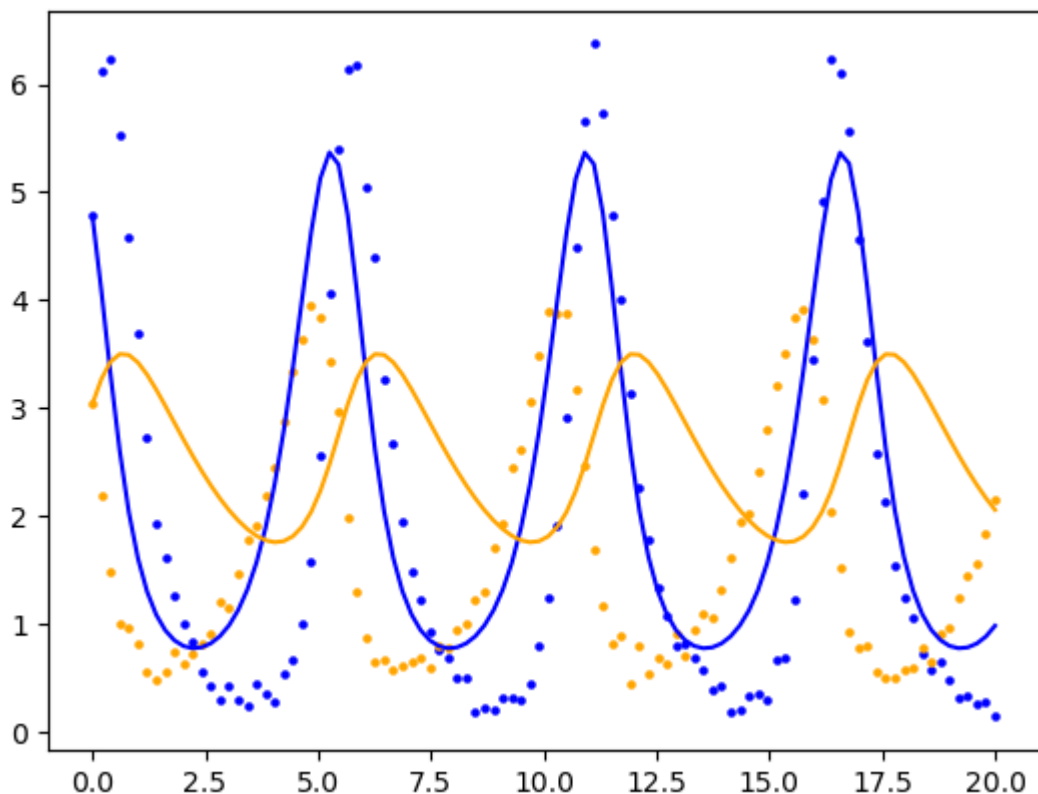
LOWEST MSE: 2.4498445634217485



Curve fit

```
In [10]: 1 # t, x, y = data
2 initial_conditions = [input_data[0][0], input_data[0][1]]
3 t = data[:,0]
4 scaling = 2
5
6 parameters = x_best
7
8 # Using MSE
9 x = predator_prey_integration(t, initial_conditions, parameters)
10
11 plt.figure(dpi=300, figsize=(6, 5))
12 point_width = 13
13 plt.plot(t, x[:,0], color="b")
14 plt.plot(t, x[:,1], color="orange")
15
16 plt.scatter(t, data[:,1], color='blue', s=5)
17 plt.scatter(t, data[:,2], color='orange', s=5)
18
19 mse_pre = MSE(data[:,1], x[:,0]) #MSE for fitted curve
20 mse_predator = MSE(data[:,2], x[:,1])
21 mse_total = mse_pre + mse_predator
22 print("Mean Square Error: " + str(mse_total))
```

Mean Square Error: 4.899689126843497



Calculation of mean and variance

```
In [10]: 1 ### Distribution of parameters for multiple runs
2
3 def multiple_runs_annealing(initial_temp,cooling_constant,input_data,t,
4
5     mse_total_list = []
6
7
8     for i in range(n_runs):
9
10         x_best, scores = simulated_annealing(initial_temp,cooling_const
11
12         x = predator_prey_integration(t,initial_conditions,x_best)
13         mse_prey = MSE(data[:,1],x[:,0])
14         mse_predator = MSE(data[:,2],x[:,1])
15         mse_total = mse_prey + mse_predator
16
17         mse_total_list.append(mse_total) #Add total MSE for this simulat
18
19     return mse_total_list
20
```

```
In [12]: 1 input_data = data[:,1:3]
2 initial_conditions = [input_data[0][0], input_data[0][1]]
3 t = data[:,0]
4 #Taking random draw for initial parameters (initial guess)
5 alpha = np.random.uniform(0,1)
6 beta = np.random.uniform(0,1)
7 delta = np.random.uniform(0,1)
8 gamma = np.random.uniform(0,1)
9 parameters = [alpha, beta, delta, gamma]
10 parameters = [alpha, beta, delta, gamma]
11
12 initial_temp = 20
13 cooling_constant = 0.10
14
15 mse_total_list = multiple_runs_annealing(initial_temp,cooling_constant,
16
17 #print(mse_total_list)
```

C:\Users\kayad\AppData\Local\Temp\ipykernel_27100\3853094899.py:38: RuntimeWarning: overflow encountered in exp

acceptance_probability = min(np.exp(-(delta/temp)),1)#Calculate acceptance probability

C:\Users\kayad\AppData\Local\Temp\ipykernel_27100\3853094899.py:38: RuntimeWarning: overflow encountered in scalar divide

acceptance_probability = min(np.exp(-(delta/temp)),1)#Calculate acceptance probability

C:\Users\kayad\AppData\Local\Temp\ipykernel_27100\3853094899.py:38: RuntimeWarning: divide by zero encountered in scalar divide

acceptance_probability = min(np.exp(-(delta/temp)),1)#Calculate acceptance probability

C:\Users\kayad\anaconda3\Lib\site-packages\scipy\integrate\odepack_py.py:248: ODEintWarning: Excess work done on this call (perhaps wrong Dfun type). Run with full_output = 1 to get quantitative information.

warnings.warn(warning_msg, ODEintWarning)

In [13]:

```
1
2 mean_mse_annealing = np.mean(mse_total_list)
3 std_mse_annealing = np.std(mse_total_list)
4
5 print("Average MSE = " + str(mean_mse_annealing))
6 print("Standard deviation of MSE = " + str(std_mse_annealing))
```

Average MSE = 5.337976129896431

Standard deviation of MSE = 0.7116981017582293

Comparison of cooling schedules

```
In [11]: 1 input_data = data[:,1:3]
2 initial_conditions = [input_data[0][0], input_data[0][1]]
3 t = data[:,0]
4 #Taking random draw for initial parameters (initial guess)
5 alpha = np.random.uniform(0,1)
6 beta = np.random.uniform(0,1)
7 delta = np.random.uniform(0,1)
8 gamma = np.random.uniform(0,1)
9 parameters = [alpha, beta, delta, gamma]
10 parameters = [alpha, beta, delta, gamma]
11
12 initial_temp = 20
13
14 cooling_constants = np.arange(0.10,1,0.10)
15
16 mean_and_sd_per_simulation = []
17
18 for constant in cooling_constants:
19     print(constant)
20     mse_total_list = multiple_runs_annealing(initial_temp,constant,input_data)
21     mean_mse_annealing = np.mean(mse_total_list)
22     std_mse_annealing = np.std(mse_total_list)
23     mean_and_sd_per_simulation.append([mean_mse_annealing,std_mse_annealing])
24
25
```

0.1

C:\Users\Aleks\AppData\Local\Temp\ipykernel_1540\1335953606.py:38: Runtime Warning: overflow encountered in exp

```
    acceptance_probability = min(np.exp(-(delta/temp)),1)#Calculate acceptance probability
```

C:\Users\Aleks\AppData\Local\anaconda3\Lib\site-packages\scipy\integrate\odepack_py.py:248: ODEintWarning: Excess work done on this call (perhaps wrong Dfun type). Run with full_output = 1 to get quantitative information. warnings.warn(warning_msg, ODEintWarning)

C:\Users\Aleks\AppData\Local\Temp\ipykernel_1540\1335953606.py:38: Runtime Warning: overflow encountered in scalar divide

```
    acceptance_probability = min(np.exp(-(delta/temp)),1)#Calculate acceptance probability
```

C:\Users\Aleks\AppData\Local\Temp\ipykernel_1540\1335953606.py:38: Runtime Warning: divide by zero encountered in scalar divide

```
    acceptance_probability = min(np.exp(-(delta/temp)),1)#Calculate acceptance probability
```

-
KeyboardInterrupt

Traceback (most recent call last)

t)

Cell In[11], line 20

```

    18 for constant in cooling_constants:
    19     print(constant)
--> 20     mse_total_list = multiple_runs_annealing(initial_temp, constant,
input_data, t, parameters, MSE, 10)
    21     mean_mse_annealing = np.mean(mse_total_list)
    22     std_mse_annealing = np.std(mse_total_list)

```

Cell In[10], line 10, in multiple_runs_annealing(initial_temp, cooling_constant, input_data, t, parameters, objective_function, n_runs)

```

    5 mse_total_list = []
    8 for i in range(n_runs):
--> 10     x_best, scores = simulated_annealing(initial_temp, cooling_constant,
input_data, t, input_data[0], parameters, objective_function, max_ite
rations=5000)
    12     x = predator_prey_integration(t, initial_conditions, x_best)
    13     mse_prey = MSE(data[:,1], x[:,0])

```

Cell In[9], line 30, in simulated_annealing(initial_temp, cooling_constant, data, time, initial_conditions, parameters, objective, max_iterations)

```

    27 for i in range(max_iterations):
    29     proposal = random_walk_annealing(x_n) #A new proposal for the
parameters is generated by taking a random walk scaled by the scale
--> 30     new_est = predator_prey_integration(time, initial_conditions,
proposal) #Calculate new function values based on proposal parameters
    31     new_score = objective(data, new_est) #Calculate new value of o
bjective function based on new function values
    33     delta = new_score - current_score #Difference in objective fun
ction values

```

Cell In[7], line 4, in predator_prey_integration(time, initial_conditions, parameters)

```

    2 def predator_prey_integration(time, initial_conditions, parameters):
    3     alpha, beta, delta, gamma = parameters
----> 4     results = odeint(predator_prey_odes, initial_conditions, time,
args=(alpha, beta, delta, gamma))
    5     predator_values, prey_values = results[:,0], results[:,1]
    6     return np.array([predator_values, prey_values]).T

```

File ~\AppData\Local\anaconda3\Lib\site-packages\scipy\integrate\odepack_py.py:241, in odeint(func, y0, t, args, Dfun, col_deriv, full_output, ml, mu, rtol, atol, tcrit, h0, hmax, hmin, ixpr, mxstep, mxhnil, mxordn, mxords, printmessg, tfirst)

```

    236     raise ValueError("The values in t must be monotonically increa
sing "
    237                        "or monotonically decreasing; repeated values
are "
    238                        "allowed.")
    240 t = copy(t)
--> 241 y0 = copy(y0)
    242 output = _odepack.odeint(func, y0, t, args, Dfun, col_deriv, ml, m
u,
    243                        full_output, rtol, atol, tcrit, h0, hmax,
hmin,
    244                        ixpr, mxstep, mxhnil, mxordn, mxords,
    245                        int(bool(tfirst)))
    246 if output[-1] < 0:

```

```
File ~\AppData\Local\anaconda3\Lib\copy.py:84, in copy(x)
    82 copier = getattr(cls, "__copy__", None)
    83 if copier is not None:
--> 84     return copier(x)
    86 reductor = dispatch_table.get(cls)
    87 if reductor is not None:
```

KeyboardInterrupt:

```
In [12]: 1 print(mean_and_sd_per_simulation)
          2
          3 plt.bar()
```

[]

```
-----
-
TypeError                                Traceback (most recent call last)
Cell In[12], line 3
      1 print(mean_and_sd_per_simulation)
----> 3 plt.bar()
```

TypeError: bar() missing 2 required positional arguments: 'x' and 'height'

Hill climbing

In [13]:

```

1
2
3 def random_walk(parameters):
4     lst = [parameter + np.random.normal(0, 1) for parameter in parameters]
5     # Ensure all elements are positive
6     while any(x <= 0 for x in lst):
7         for indx in range(len(lst)):
8             if lst[indx] <= 0:
9                 lst[indx] = max(0, parameters[indx] + np.random.normal(
10
11     return lst
12
13 def hill_climbing(initial_temp, cooling_constant, data, time, initial_conditions):
14     '''Tries to find the best solution using random walker'''
15     # Initialize starting parameter state
16     scores = []
17     x_n = parameters
18
19     current_est = predator_prey_integration(time, initial_conditions, x_n)
20     current_score = objective(data, current_est)
21     scores.append(current_score)
22
23     for _ in range(max_iterations):
24         # Generate a random walk for parameters
25         x_n_1 = random_walk(x_n)
26
27         # Calculate the current and next estimations
28         current_est = predator_prey_integration(time, initial_conditions, x_n)
29         new_estimation = predator_prey_integration(time, initial_conditions, x_n_1)
30         new_score = objective(data, new_estimation)
31
32         # If the next estimation is better, update the parameters
33         if new_score < current_score:
34             current_score = new_score
35             x_n = x_n_1
36             scores.append(current_score)
37
38     return [x_n, scores]
39

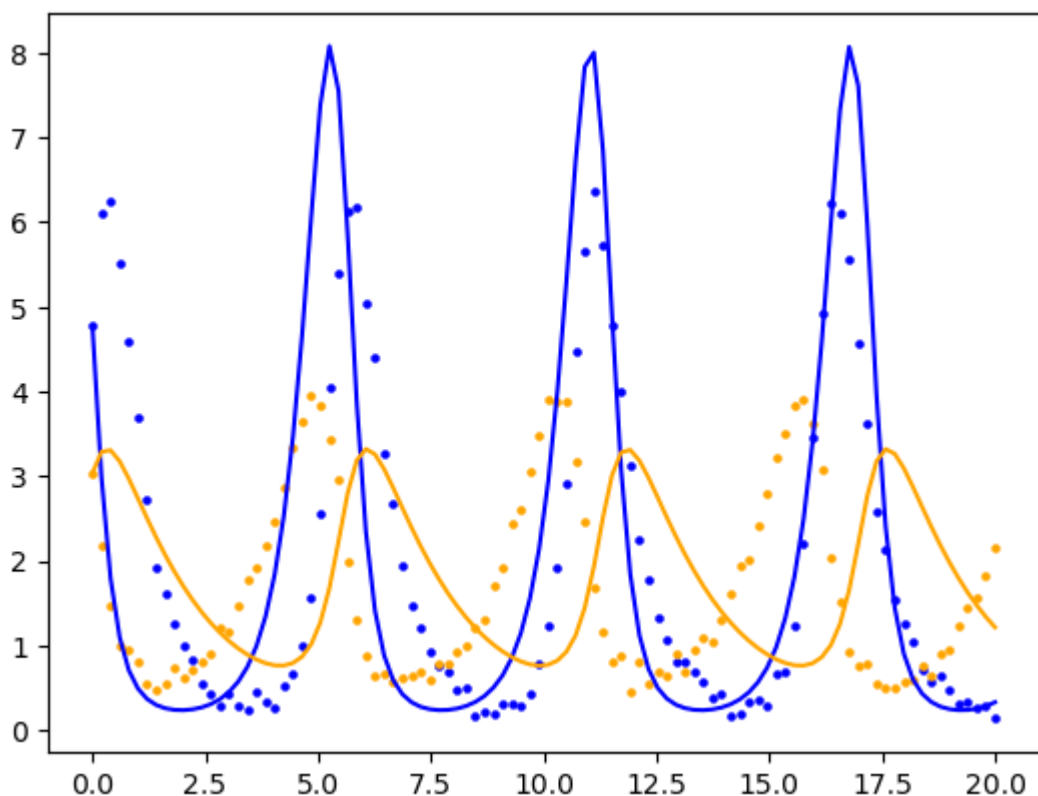
```

```
In [35]: 1 input_data = data[:,1:3]
2 initial_conditions = [input_data[0][0], input_data[0][1]]
3 t = data[:,0]
4 #Taking random draw for initial parameters (initial guess)
5 alpha = np.random.uniform(0,1)
6 beta = np.random.uniform(0,1)
7 delta = np.random.uniform(0,1)
8 gamma = np.random.uniform(0,1)
9 parameters = [alpha, beta, delta, gamma]
10
11 # Using MSE
12 x_best = hill_climbing(initial_temp,cooling_constant, input_data, t, ir
13
```


Curve fit

```
In [17]: 1 # t, x ,y = data
2 initial_conditions = [input_data[0][0], input_data[0][1]]
3 t = data[:,0]
4
5 parameters = x_best
6
7 # Using MSE
8 x = predator_prey_integration(t,initial_conditions,parameters)
9
10 point_width = 13
11 plt.plot(t, x[:,0],color = "b")
12 plt.plot(t, x[:,1],color = "orange")
13
14 plt.scatter(t, data[:,1], color= 'blue', s = 5)
15 plt.scatter(t, data[:,2], color= 'orange', s = 5)
16
17 mse_prey = MSE(data[:,1],x[:,0]) #MSE for fitted curve
18 mse_predator = MSE(data[:,2],x[:,1])
19 mse_total = mse_prey + mse_predator
20 print("Mean Square Error: " +str(mse_total))
21
```

Mean Square Error: 6.016318958977161



```
In [36]: 1 def multiple_runs_hill_climbing(initial_temp,cooling_constant, input_data)
2
3     mse_total_list = []
4
5     for i in range(n_runs):
6
7         x_best = hill_climbing(initial_temp,cooling_constant, input_data)
8
9         x = predator_prey_integration(t,initial_conditions,x_best)
10
11         mse_prey = MSE(data[:,1],x[:,0])
12         mse_predator = MSE(data[:,2],x[:,1])
13         mse_total = mse_prey + mse_predator
14
15         mse_total_list.append(mse_total) #Add total MSE for this simulation
16
17     return mse_total_list
18
```

```
In [37]: 1 input_data = data[:,1:3]
2 initial_conditions = [input_data[0][0], input_data[0][1]]
3 t = data[:,0]
4 #Taking random draw for initial parameters (initial guess)
5 alpha = np.random.uniform(0,1)
6 beta = np.random.uniform(0,1)
7 delta = np.random.uniform(0,1)
8 gamma = np.random.uniform(0,1)
9 parameters = [alpha, beta, delta, gamma]
10
11
12 mse_total_list = multiple_runs_hill_climbing(initial_temp,cooling_constant,
```

```
In [20]: 1 mean_mse_hill_climbing = np.mean(mse_total_list)
2 std_mse_hill_climbing = np.std(mse_total_list)
3
4 print("Average MSE = " + str(mean_mse_hill_climbing))
5 print("Standard deviation of MSE = " + str(std_mse_hill_climbing))
```

Average MSE = 7.258042040055488

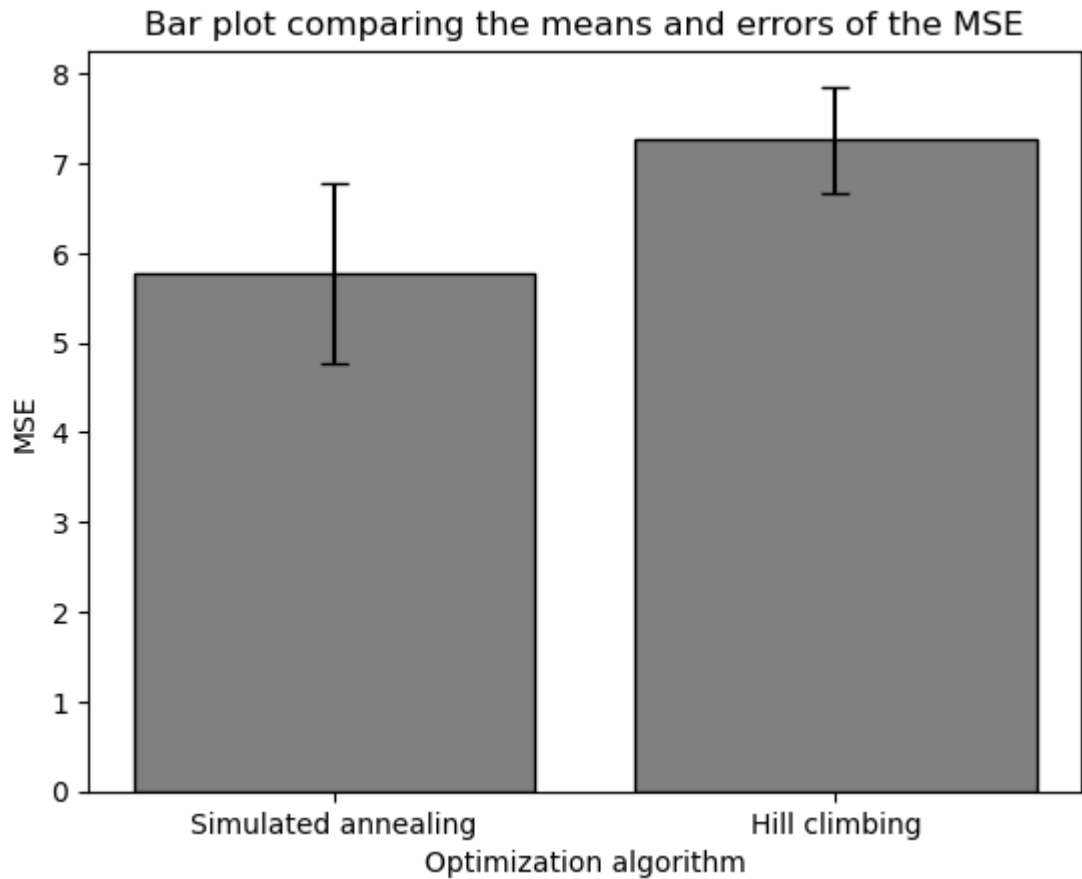
Standard deviation of MSE = 0.5962174191554426

Comparison of Optimisation Algorithms

MSE objective function

```
In [21]: 1 x = ["Simulated annealing", "Hill climbing"]
2 mse_list = [mean_mse_annealing, mean_mse_hill_climbing] #Stores the mean
3 std_list = [std_mse_annealing, std_mse_hill_climbing]
4
5 plt.bar(x, mse_list, yerr = std_list, color = "grey", ec = "black", ecolor = "black")
6 plt.title("Bar plot comparing the means and errors of the MSE")
7 plt.ylabel("MSE")
8 plt.xlabel("Optimization algorithm")
```

Out[21]: Text(0.5, 0, 'Optimization algorithm')



```
In [48]: 1
2 good= data[:,1:3]
3 bad = np.zeros_like(good)
4
5 print(negative_log_likelihood(x,good))
6 print()
7 print(negative_log_likelihood(x,bad))
```

Reverse engineering with removal of points targeted

In [18]:

```

1  def MSE2(actual, predicted):
2      '''Mean squared error'''
3      x1 = actual[:, 0]
4      y1 = actual[:, 1]
5
6      #Getting useful indexes
7      indx_x = np.where(~np.isnan(x1))
8      indx_y = np.where(~np.isnan(y1))
9
10     x2, y2 = predicted[:, 0], predicted[:, 1]
11     err1 = (x1[indx_x] - x2[indx_x])**2
12     err2 = (y1[indx_y] - y2[indx_y])**2
13     err = np.concatenate([err1, err2])
14
15     return np.nanmean([err])
16
17 def MAE(actual, predicted):
18     '''Mean absolute error'''
19     return np.mean(np.abs(actual - predicted))
20
21 def MAE2(actual, predicted):
22     x1 = actual[0]
23     y1 = actual[1]
24
25     #Getting useful indexes
26     indx_x = np.where(~np.isnan(x1))
27     indx_y = np.where(~np.isnan(y1))
28
29     x2, y2 = predicted[:, 0], predicted[:, 1]
30     err1 = abs(x1[indx_x] - x2[indx_x])
31     err2 = abs(y1[indx_y] - y2[indx_y])
32     err = np.concatenate([err1, err2])
33     err = np.concatenate([err1, err2])
34
35     return np.nanmean([err])
36
37
38 def point_removal(time, input_data, points_removed, Focus = 0):
39     prey = input_data.T[0].copy()
40     predator = input_data.T[1].copy()
41
42     # initialize set up for removing points randomly given the bounds
43     removal_options = np.arange(0, len(time))
44
45     # choose points to be removed randomly
46     if points_removed > len(removal_options):
47         points_removed = len(removal_options)
48         print('WARNING: Maximum number of points that can be removed has been reached')
49     removed_points_indices = random.choices(removal_options, k = points_removed)
50
51     # remove points based on choices for points to be removed
52     if Focus == 0:
53         for i in removed_points_indices:
54             prey[i] = None
55             predator[i] = None
56
57     elif Focus == 1:
58         for i in removed_points_indices:
59             prey[i] = None
60
61     elif Focus == 2:

```

```
62         for i in removed_points_indices:
63             predator[i] = None
64
65     return np.array(time), np.array(pre), np.array(predator)
66
```



```

In [19]: 1 def extrema_removal(time, input_data, points_removed, Focus = 0):
2         prey = input_data.T[0].copy()
3         predator = input_data.T[1].copy()
4
5         # Calculate mean and variance to set regions for data
6         mean_prey_population, mean_predator_population = np.mean(prey), np.
7         variance_prey, variance_predator = statistics.variance(prey), stati
8
9         # set upper bound and lower bound for point removals
10        ub_prey, lb_prey = mean_prey_population + 0.45 * variance_prey, mea
11        ub_predator, lb_predator = mean_predator_population + 0.9 *variance
12
13        # initialize set up for removing points randomly given the bounds
14        prey_options = []
15        predator_options = []
16        # enumerate through list of stored points
17        for index, prey_count in enumerate(prey):
18            # check if they are in specified region
19            if prey_count >= ub_prey or prey_count <= lb_prey:
20                prey_options.append([index, prey_count, predator[index]])
21        for index, predator_count in enumerate(predator):
22            if predator_count >= ub_predator or predator_count <= lb_predat
23                predator_options.append([index, prey[index], predator_count
24
25        # remove points from list depending on which focus is set
26        removal_options = []
27        if Focus == 0:
28            removal_options = removal_options + prey_options + predator_opt
29        elif Focus == 1:
30            removal_options = removal_options + prey_options
31        elif Focus == 2:
32            removal_options = removal_options + predator_options
33        else:
34            print('Error: Removal option not known. Try either both, prey,
35
36        # choose points to be removed randomly
37        if points_removed > len(removal_options):
38            points_removed = len(removal_options)
39            print('WARNING: Maximum number of points that can be removed ha
40        removed_points_indices = random.choices(np.array(removal_options).T
41
42        # turn the list into integers so we can remove them based on the in
43        integer_array = []
44        for counter in range(len(removed_points_indices)):
45            integer_array.append(int(removed_points_indices[counter]))
46
47        # update the lists based on points we wanted to remove
48        if Focus == 0:
49            for i in integer_array:
50                prey[i] = None
51                predator[i] = None
52
53        elif Focus == 1:
54            for i in integer_array:
55                prey[i] = None
56
57        elif Focus == 2:
58            for i in integer_array:
59                predator[i] = None
60
61        return np.array(time), np.array(prey), np.array(predator)

```


62	
63	


```

In [314]: 1 def midpoint_removal(time, input_data, points_removed, Focus = 0):
2         prey = input_data.T[0].copy()
3         predator = input_data.T[1].copy()
4
5         # Calculate mean and variance to set regions for data
6         mean_prey_population, mean_predator_population = np.mean(prey), np.
7         variance_prey, variance_predator = statistics.variance(prey), stati
8
9         # set upper bound and lower bound for point removals
10        ub_prey, lb_prey = mean_prey_population + 0.45 * variance_prey, mea
11        ub_predator, lb_predator = mean_predator_population + 0.9 *variance
12
13        # initialize set up for removing points randomly given the bounds
14        prey_options = []
15        predator_options = []
16        # enumerate through list of stored points
17        for index, prey_count in enumerate(prey):
18            # check if they are in specified region
19            if prey_count <= ub_prey and prey_count >= lb_prey:
20                prey_options.append([index, prey_count, predator[index]])
21        for index, predator_count in enumerate(predator):
22            if predator_count <= ub_predator and predator_count >= lb_preda
23                predator_options.append([index, prey[index], predator_count
24
25        # remove points from list depending on which focus is set
26        removal_options = []
27        if Focus == 0:
28            removal_options = removal_options + prey_options + predator_opt
29        elif Focus == 1:
30            removal_options = removal_options + prey_options
31        elif Focus == 2:
32            removal_options = removal_options + predator_options
33        else:
34            print('Error: Removal option not known. Try either both, prey,
35
36        # choose points to be removed randomly
37        if points_removed > len(removal_options):
38            points_removed = len(removal_options)
39            print('WARNING: Maximum number of points that can be removed ha
40        removed_points_indices = random.choices(np.array(removal_options).T
41
42        # turn the list into integers so we can remove them based on the in
43        integer_array = []
44        for counter in range(len(removed_points_indices)):
45            integer_array.append(int(removed_points_indices[counter]))
46
47        # update the lists based on points we wanted to remove
48        if Focus == 0:
49            for i in integer_array:
50                prey[i] = None
51                predator[i] = None
52
53        elif Focus == 1:
54            for i in integer_array:
55                prey[i] = None
56
57        elif Focus == 2:
58            for i in integer_array:
59                predator[i] = None
60
61        return np.array(time), np.array(prey), np.array(predator)

```

```
62  
63 midpoint_removal(time, input_data, points_removed, 0)
```

```

Out[314]: (array([ 0.          ,  0.2020202 ,  0.4040404 ,  0.60606061,  0.80808081,
    1.01010101,  1.21212121,  1.41414141,  1.61616162,  1.81818182,
    2.02020202,  2.22222222,  2.42424242,  2.62626263,  2.82828283,
    3.03030303,  3.23232323,  3.43434343,  3.63636364,  3.83838384,
    4.04040404,  4.24242424,  4.44444444,  4.64646465,  4.84848485,
    5.05050505,  5.25252525,  5.45454545,  5.65656566,  5.85858586,
    6.06060606,  6.26262626,  6.46464646,  6.66666667,  6.86868687,
    7.07070707,  7.27272727,  7.47474747,  7.67676768,  7.87878788,
    8.08080808,  8.28282828,  8.48484848,  8.68686869,  8.88888889,
    9.09090909,  9.29292929,  9.49494949,  9.6969697 ,  9.8989899 ,
   10.1010101 , 10.3030303 , 10.50505051, 10.70707071, 10.90909091,
   11.11111111, 11.31313131, 11.51515152, 11.71717172, 11.91919192,
   12.12121212, 12.32323232, 12.52525253, 12.72727273, 12.92929293,
   13.13131313, 13.33333333, 13.53535354, 13.73737374, 13.93939394,
   14.14141414, 14.34343434, 14.54545455, 14.74747475, 14.94949495,
   15.15151515, 15.35353535, 15.55555556, 15.75757576, 15.95959596,
   16.16161616, 16.36363636, 16.56565657, 16.76767677, 16.96969697,
   17.17171717, 17.37373737, 17.57575758, 17.77777778, 17.97979798,
   18.18181818, 18.38383838, 18.58585859, 18.78787879, 18.98989899,
   19.19191919, 19.39393939, 19.5959596 , 19.7979798 , 20.          ]),
array([4.78163509, 6.11400461, 6.23836095, 5.52052405, 4.58254575,
    3.69549338, 2.73241063, 1.93007859,          nan, 1.25280509,
    0.99647646, 0.83616635, 0.55796464, 0.42574735, 0.29284544,
    0.42998578, 0.29047429, 0.23623108,          nan, 0.34584789,
    0.26683052, 0.52841771, 0.65807025,          nan,          nan,
    2.55628162, 4.05272265, 5.39827216, 6.13430431, 6.17559672,
    5.04337145, 4.40335325, 3.26795378, 2.66918471, 1.95164064,
    1.47972638, 1.21762543, 0.92610618, 0.76317457, 0.68336557,
    0.48930524, 0.50292464, 0.17631951, 0.21193155, 0.19391247,
    0.31392994,          nan, 0.2876866 , 0.43413798, 0.78532619,
    1.23239764, 1.91250492,          nan, 4.48294454, 5.65027253,
    6.37322741, 5.73552279, 4.77653998, 4.00597789, 3.13453968,
    2.25261307, 1.78222451, 1.33145956, 1.07897012, 0.79924317,
    0.82105872, 0.68730727, 0.57397704,          nan, 0.42231647,
    0.17771466, 0.19526324, 0.32527476, 0.34995521, 0.28914043,
    0.67309272, 0.68659273, 1.23055182, 2.20768819, 3.45148561,
    4.92302552, 6.22992039, 6.0991116 , 5.57323872, 4.56083925,
    3.62460823, 2.57661795, 2.1230354 , 1.5435279 , 1.24764464,
          nan, 0.71257577,          nan, 0.65039322,          nan,
    0.31034585, 0.32459509, 0.25808311, 0.28367985, 0.13595587]),
array([3.03525736, 2.18974589, 1.47890677, 0.98983604, 0.95782741,
    0.8089764 , 0.5550991 , 0.48259774,          nan, 0.7378781 ,
    0.61912135, 0.71381421, 0.80913925, 0.90709674, 1.20019798,
    1.15551425, 1.4620071 , 1.7790317 ,          nan, 2.18036662,
    2.45451334, 2.87645734, 3.34293736,          nan,          nan,
    3.83569514, 3.42440295, 2.97234104, 1.97992361, 1.30265342,
    0.8702643 , 0.65378973, 0.67275523, 0.57794837, 0.61721216,
    0.64514385, 0.69083686, 0.59946108, 0.79222153, 0.78077762,
    0.93441085, 1.00700856, 1.21962625, 1.29428758, 1.70666312,
    1.92431547,          nan, 2.61408082, 3.05851749, 3.4901669 ,
    3.90039401, 3.88133056,          nan, 3.16769416, 2.47120556,
    1.68769771, 1.16042605, 0.80690056, 0.88570405, 0.44751714,
    0.7982129 , 0.53985993, 0.68495287, 0.63356906, 0.90963702,
    0.69373712, 0.94734264, 1.08333741,          nan, 1.31674353,
    1.61225858, 1.9424163 , 2.01694512, 2.40634827, 2.79703811,
    3.21140628, 3.50568978, 3.83378881, 3.90396118, 3.63260061,
    3.08637832, 2.03180558, 1.52594296, 0.93100161, 0.77101994,
    0.79666992, 0.55440616, 0.49615381, 0.49720865, 0.57677885,
          nan, 0.77056314,          nan, 0.9048621 ,          nan,
    1.24188957, 1.43725675, 1.56281031, 1.83417075, 2.14706546]))

```

```
In [16]: 1 def inference_removal(initial_temp,cooling_constant, objective, algorithm)
2         # define input data based on number of points we want to remove
3         new_data_set = removal_type(time, input_data, points_removed, Focus)
4         time_new = new_data_set[0]
5         prey_new = new_data_set[1]
6         predator_new = new_data_set[2]
7
8         # prepare data to match input requirements of algorithm (so far only)
9         input_data_new = []
10        for i in range(len(time_new)):
11            input_data_tuple = [prey_new[i], predator_new[i]]
12            input_data_new.append(input_data_tuple)
13
14        # apply algorithm to find best fitted parameters given new data set
15        x_best, scores = algorithm(initial_temp,cooling_constant, np.array(input_data_new),
16                                   objective, algorithm)
17
18        return x_best, scores
19
20 # parameters = [0.5, 0.6, 0.7, 0.8]
21 # x = inference_removal(initial_temp,cooling_constant, MSE2, simulated_data, parameters)
22 # print(x[0])
```

```

In [268]: 1 initial_conditions = [input_data[0][0], input_data[0][1]]
          2 t = data[:,0]
          3 time = t
          4 input_data = data[:,1:3]
          5
          6 best = inference_removal(initial_temp,cooling_constant, MSE2, simulated
          7 removed = inference_removal(initial_temp,cooling_constant, MSE2, simula
          8
          9 best_params_tests = best[0]
         10 reduced_params_test = removed[0]
         11
         12 res1 = predator_prey_integration(time,initial_conditions,best_params_te
         13 res2 = predator_prey_integration(time,initial_conditions,reduced_params
         14
         15 MSE_ = MSE2(res1, input_data)
         16 MSE = MSE2(res2, input_data)
         17
         18 print(MSE_, MSE)
         19
         20 plt.figure()
         21 plt.subplot(311)
         22 plt.plot(data[:,0], res1[:,0])
         23 plt.plot(data[:,0], res1[:,1])
         24
         25 plt.subplot(312)
         26 plt.scatter(time, data[:,1])
         27 plt.scatter(time, data[:, 2])
         28
         29 plt.subplot(313)
         30 plt.plot(data[:,0], res2[:,0])
         31 plt.plot(data[:,0], res2[:,1])
         32 plt.show()
         33 plt.close

```

C:\Users\kayad\AppData\Local\Temp\ipykernel_27100\3853094899.py:38: RuntimeWarning: overflow encountered in exp

acceptance_probability = min(np.exp(-(delta/temp)),1)#Calculate acceptance probability

C:\Users\kayad\AppData\Local\Temp\ipykernel_27100\3853094899.py:38: RuntimeWarning: overflow encountered in scalar divide

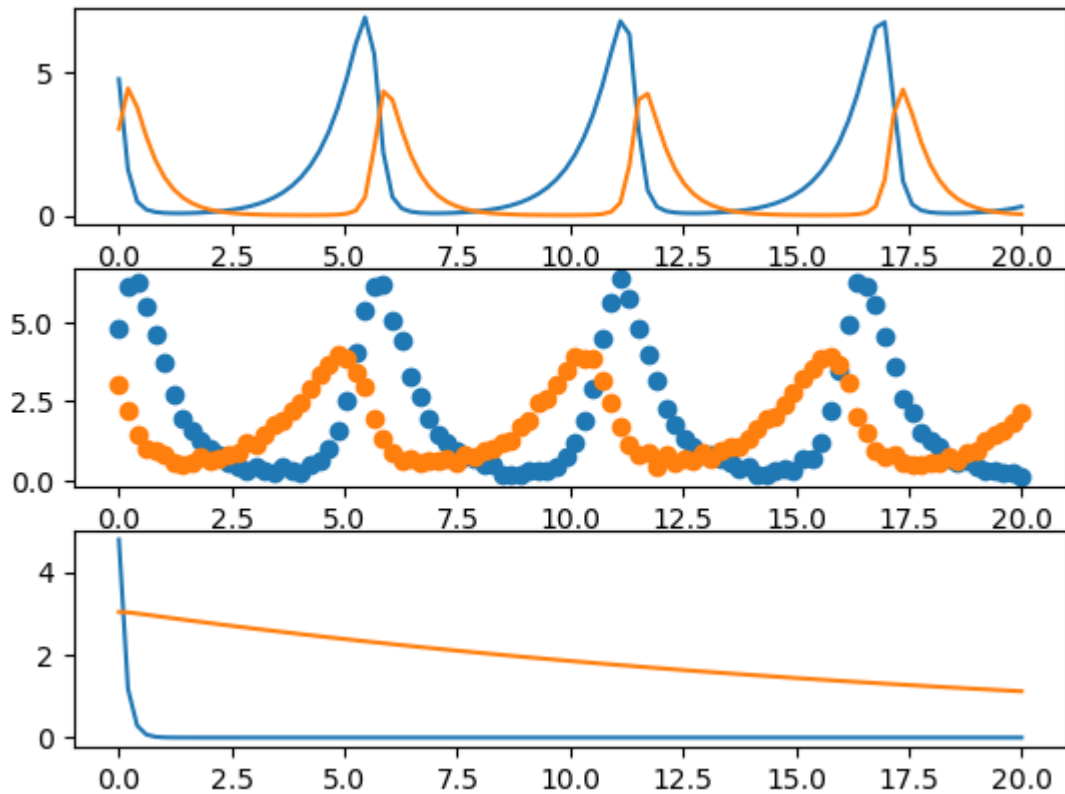
acceptance_probability = min(np.exp(-(delta/temp)),1)#Calculate acceptance probability

C:\Users\kayad\AppData\Local\Temp\ipykernel_27100\3853094899.py:38: RuntimeWarning: divide by zero encountered in scalar divide

acceptance_probability = min(np.exp(-(delta/temp)),1)#Calculate acceptance probability

WARNING: Maximum number of points that can be removed has been exceeded. All points possible within the given bound have now been removed.

3.7074011653569787 4.557827773152935



Out[268]: <function matplotlib.pyplot.close(fig=None)>

```
In [378]: 1 def multirun_SA_point_removal(iterations, simulations, initial_temperat
2         # initialize objective score sufficiently high
3         Lowest_MSE = [100000]
4         best_param_overall = []
5         # start simulated annealing from varying initial parameters
6         for simulation_count in range(simulations):
7             init_conditions = original_data[0]
8             initial_parameter_guess = [np.random.uniform(0,1), np.random.ur
9             # iterate several times for each initial condition
10            for iteration_count in range(iterations):
11                # find parameters best fitted
12                best_found_parameters = inference_removal(initial_temperatu
13                # integrate
14                Reversed_curve = predator_prey_integration(timestep, init_co
15                # compute objective function
16                MSE = objective(original_data, Reversed_curve)
17                # check if current objective lower than initial ones and if
18                if MSE < Lowest_MSE[-1]:
19                    best_param_overall.append(best_found_parameters[0])
20                    Lowest_MSE.append(MSE)
21            return best_param_overall[-1], Lowest_MSE[-1]
22
```



```

In [425]: 1 # Points removed with aggressive cooling for prey population, simulated
2 simulations = 2
3 iterations = 1
4 init_temp = 20
5 cooling = 0.1 # cooling strategy more aggressive
6 t = data[:,0]
7 time = t
8 inp = data[:,1:3]
9
10 all_scores_pre = []
11 plt.figure(dpi = 300)
12 for counter in range(25):
13     # storage
14     params_pre = []
15     objective_results_pre = []
16     # point removal loop (removing points at steps of 2)
17     npoints = np.arange(0,100,5)
18     for p_removed in npoints:
19         best_parameters_found_pre_removal = multirun_SA_point_removal(
20             params_pre.append(best_parameters_found_pre_removal[0])
21             objective_results_pre.append(best_parameters_found_pre_removal[1])
22         all_scores_pre.append(objective_results_pre)
23 means_scores_pre = []
24 variance_scores_pre = []
25 for index in range(len(npoints)):
26     current_score_count = np.array(all_scores_pre)[: ,index]
27     means_scores_pre.append(np.mean(current_score_count))
28     variance_scores_pre.append(statistics.variance(current_score_count))
29 ci = 1.96 * np.array(variance_scores_pre)/25
30 fig, ax = plt.subplots()
31 ax.plot(npoints,means_scores_pre, linewidth = '1.2', color = 'red')
32 ax.fill_between(npoints, (np.array(means_scores_pre)-ci), (np.array(means_scores_pre)+ci))
33 #plt.plot(npoints, means_scores_pre, linewidth = '1.2', color = 'red')
34 plt.xlabel('Number of points removed')
35 plt.ylabel('Mean Squared Error')
36 plt.show()
37 plt.close()

```

C:\Users\kayad\AppData\Local\Temp\ipykernel_27100\3853094899.py:38: RuntimeWarning: overflow encountered in exp

acceptance_probability = min(np.exp(-(delta/temp)),1)#Calculate acceptance probability

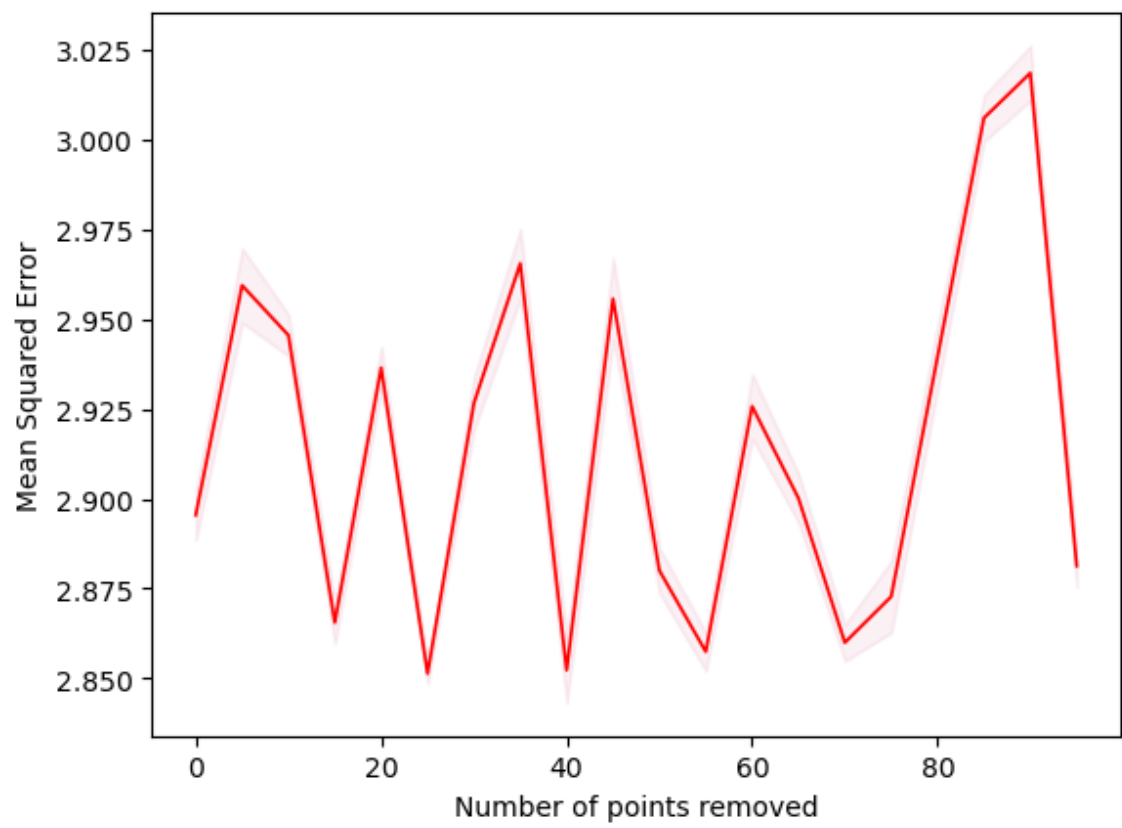
C:\Users\kayad\AppData\Local\Temp\ipykernel_27100\3853094899.py:38: RuntimeWarning: overflow encountered in scalar divide

acceptance_probability = min(np.exp(-(delta/temp)),1)#Calculate acceptance probability

C:\Users\kayad\AppData\Local\Temp\ipykernel_27100\3853094899.py:38: RuntimeWarning: divide by zero encountered in scalar divide

acceptance_probability = min(np.exp(-(delta/temp)),1)#Calculate acceptance probability

<Figure size 1920x1440 with 0 Axes>



```

In [429]: 1 # Points removed with aggressive cooling for prey population, simulated
2 simulations = 2
3 iterations = 1
4 init_temp = 20
5 cooling = 0.1 # cooling strategy more aggressive
6 t = data[:,0]
7 time = t
8 inp = data[:,1:3]
9
10 all_scores_predprey = []
11 plt.figure(dpi = 300)
12 for counter in range(25):
13     # storage
14     params_predprey = []
15     objective_results_predprey = []
16     # point removal loop (removing points at steps of 2)
17     npoints = np.arange(0,100,5)
18     for p_removed in npoints:
19         best_parameters_found_predprey_removal = multirun_SA_point_removal
20         params_predprey.append(best_parameters_found_predprey_removal)
21         objective_results_predprey.append(best_parameters_found_predprey_removal)
22     all_scores_predprey.append(objective_results_predprey)
23 means_scores_predprey = []
24 variance_scores_predprey = []
25 for index in range(len(npoints)):
26     current_score_count_predprey = np.array(all_scores_predprey)[: ,index]
27     means_scores_predprey.append(np.mean(current_score_count))
28     variance_scores_predprey.append(statistics.variance(current_score_count))
29 ci_predprey = 1.96 * np.array(variance_scores_predprey)/25
30 plt.figure(dpi = 300)
31 fig, ax = plt.subplots()
32 ax.plot(npoints,means_scores_predprey, linewidth = '1.2', color = 'red')
33 ax.fill_between(npoints, (np.array(means_scores_predprey)-ci_predprey),
34 #plt.plot(npoints, means_scores_predprey, linewidth = '1.2', color = 'red')
35 plt.xlabel('Number of points removed')
36 plt.ylabel('Mean Squared Error')
37 plt.show()
38 plt.close()

```

C:\Users\kayad\AppData\Local\Temp\ipykernel_27100\3853094899.py:38: RuntimeWarning: overflow encountered in exp

acceptance_probability = min(np.exp(-(delta/temp)),1)#Calculate acceptance probability

C:\Users\kayad\AppData\Local\Temp\ipykernel_27100\3853094899.py:38: RuntimeWarning: overflow encountered in scalar divide

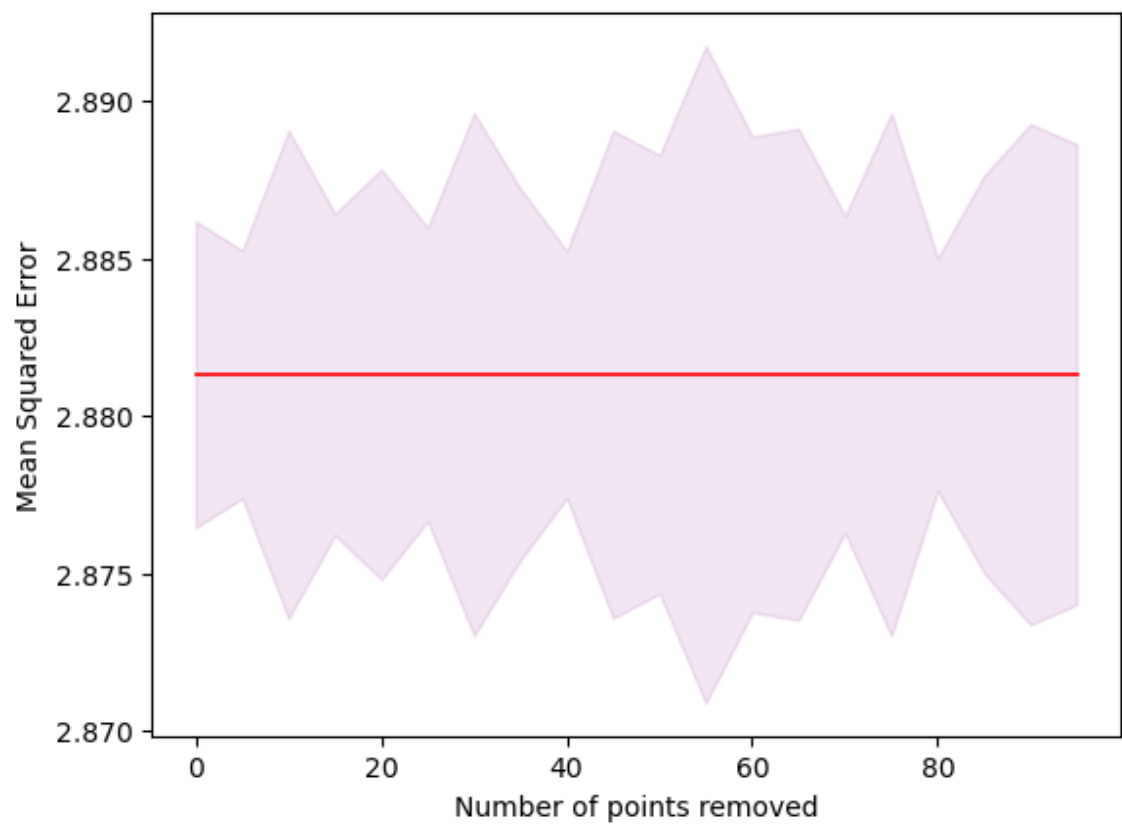
acceptance_probability = min(np.exp(-(delta/temp)),1)#Calculate acceptance probability

C:\Users\kayad\AppData\Local\Temp\ipykernel_27100\3853094899.py:38: RuntimeWarning: divide by zero encountered in scalar divide

acceptance_probability = min(np.exp(-(delta/temp)),1)#Calculate acceptance probability

<Figure size 1920x1440 with 0 Axes>

<Figure size 1920x1440 with 0 Axes>



```

In [424]: 1 # Points removed with slower cooling simulated annealing and slower cooling
2 simulations = 2
3 iterations = 1
4 init_temp = 20
5 cooling = 0.9
6 t = data[:,0]
7 time = t
8 inp = data[:,1:3]
9
10 all_scores_predator = []
11 plt.figure(dpi = 300)
12 for counter in range(25):
13     # storage
14     params_predator = []
15     objective_results_predator = []
16     # point removal loop (removing points at steps of 2)
17     npoints = np.arange(0,100,5)
18     for p_removed in npoints:
19         best_parameters_found_predator_removal = multirun_SA_point_removal
20         params_predator.append(best_parameters_found_predator_removal)
21         objective_results_predator.append(objective_results_predator_removal)
22         all_scores_predator.append(objective_results_predator)
23         #plt.plot(npoints,objective_results_pre, linewidth = '0.6', color = 'red')
24     means_scores_predator = []
25     variance_scores_predator = []
26     for index in range(len(npoints)):
27         current_score_count_predator = np.array(all_scores_predator)[:,index]
28         means_scores_predator.append(np.mean(current_score_count_predator))
29         variance_scores_predator.append(statistics.variance(current_score_count_predator))
30     ci_predator = 1.96 * np.array(variance_scores_predator)/25
31     fig, ax = plt.subplots()
32     ax.plot(npoints,means_scores_predator, linewidth = '1', color = 'red')
33     ax.fill_between(npoints, (np.array(means_scores_predator)-ci_predator),
34                    (np.array(means_scores_predator)+ci_predator), color = 'red')
35     plt.xlabel('Number of points removed')
36     plt.ylabel('Mean Squared Error')
37     plt.show()
38     plt.close()

```

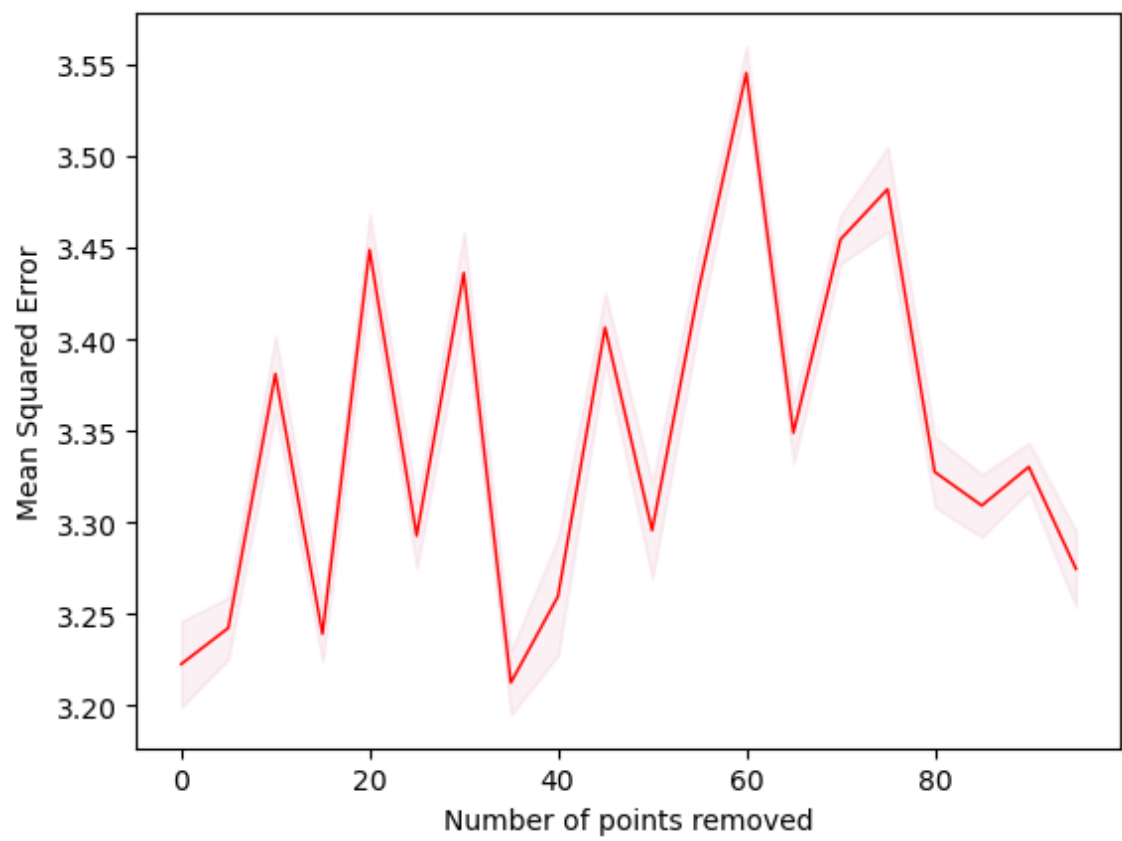
C:\Users\kayad\AppData\Local\Temp\ipykernel_27100\3853094899.py:38: RuntimeWarning: overflow encountered in exp

acceptance_probability = min(np.exp(-(delta/temp)),1)#Calculate acceptance probability

C:\Users\kayad\AppData\Local\Temp\ipykernel_27100\528580857.py:11: RuntimeWarning: overflow encountered in square

err1 = (x1[indx_x] - x2[indx_x])**2

<Figure size 1920x1440 with 0 Axes>



In [423]:

```

1  # Points removed with aggressive cooling
2  simulations = 2
3  iterations = 1
4  init_temp = 20
5  cooling = 0.1 # cooling strategy more aggressive
6  t = data[:,0]
7  time = t
8  inp = data[:,1:3]
9
10 all_scores = []
11 plt.figure(dpi = 300)
12 for counter in range(25):
13     # storage
14     params_all = []
15     objective_results_all = []
16     # point removal loop (removing points at steps of 2)
17     npoints = np.arange(0,100,5)
18     for p_removed in npoints:
19         best_parameters_found_all_removal = multirun_SA_point_removal(i
20         params_all.append(best_parameters_found_all_removal[0])
21         objective_results_all.append(best_parameters_found_all_removal[
22     all_scores.append(objective_results_all)
23     #plt.plot(npoints ,objective_results_pre, linewidth = '0.6', color
24 means_scores_all = []
25 variance_scores_all = []
26 for index in range(len(npoints)):
27     current_score_count_all = np.array(all_scores)[: ,index]
28     means_scores_all.append(np.mean(current_score_count_all))
29     variance_scores_all.append(statistics.variance(current_score_count_
30 ci_all = 1.96 * np.array(variance_scores_all)/25
31 fig, ax = plt.subplots()
32 ax.plot(npoints,means_scores_all, linewidth = '1.2', color = 'red')
33 ax.fill_between(npoints, (np.array(means_scores_all)-ci_all), (np.array
34 #plt.plot(npoints, means_scores_pre, linewidth = '1.2', color = 'red')
35 plt.xlabel('Number of points removed')
36 plt.ylabel('Mean Squared Error')
37 plt.show()
38 plt.close()

```

C:\Users\kayad\AppData\Local\Temp\ipykernel_27100\3853094899.py:38: RuntimeWarning: overflow encountered in exp

acceptance_probability = min(np.exp(-(delta/temp)),1)#Calculate acceptance probability

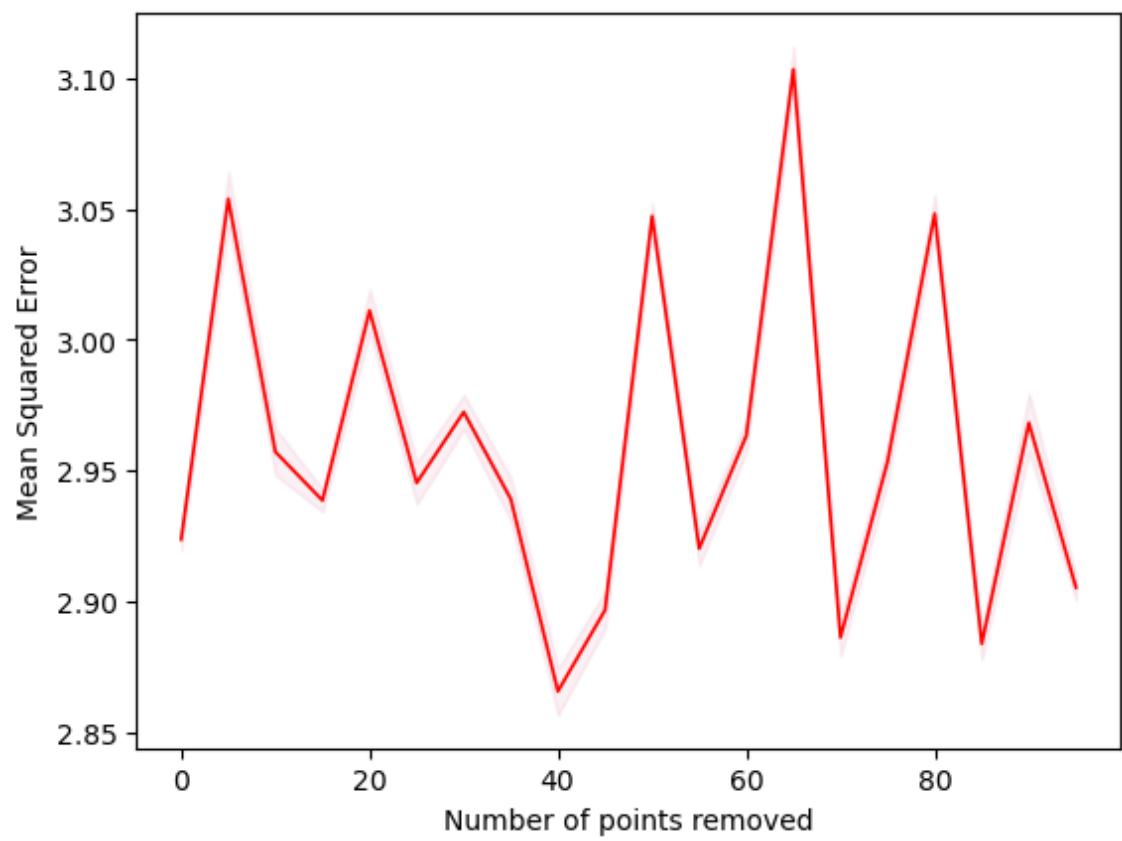
C:\Users\kayad\AppData\Local\Temp\ipykernel_27100\3853094899.py:38: RuntimeWarning: overflow encountered in scalar divide

acceptance_probability = min(np.exp(-(delta/temp)),1)#Calculate acceptance probability

C:\Users\kayad\AppData\Local\Temp\ipykernel_27100\3853094899.py:38: RuntimeWarning: divide by zero encountered in scalar divide

acceptance_probability = min(np.exp(-(delta/temp)),1)#Calculate acceptance probability

<Figure size 1920x1440 with 0 Axes>




```

In [418]: 1 # Point extrema removed with aggressive cooling MSE2 function prey popu
2 simulations = 2
3 iterations = 1
4 init_temp = 20
5 cooling = 0.1 # cooling strategy more aggressive
6 t = data[:,0]
7 time = t
8 inp = data[:,1:3]
9
10 all_scores_pre = []
11 plt.figure(dpi = 300)
12 for counter in range(25):
13     # storage
14     params_pre = []
15     objective_results_pre = []
16     # point removal loop (removing points at steps of 2)
17     npoints = np.arange(0,30,5)
18     for p_removed in npoints:
19         best_parameters_found_pre_removal = multirun_SA_point_removal(
20             params_pre.append(best_parameters_found_pre_removal[0])
21             objective_results_pre.append(best_parameters_found_pre_removal[1])
22         all_scores_pre.append(objective_results_pre)
23         #plt.plot(npoints, objective_results_pre, linewidth = '0.6', color = 'red')
24     means_scores_pre = []
25     variance_scores_pre = []
26     for index in range(len(npoints)):
27         current_score_count = np.array(all_scores_pre)[:,index]
28         means_scores_pre.append(np.mean(current_score_count))
29         variance_scores_pre.append(statistics.variance(current_score_count))
30     ci = 1.96 * np.array(variance_scores_pre)/25
31     fig, ax = plt.subplots()
32     ax.plot(npoints, means_scores_pre, linewidth = '1.2', color = 'red')
33     ax.fill_between(npoints, (np.array(means_scores_pre)-ci), (np.array(means_scores_pre)+ci), color = 'red')
34     #plt.plot(npoints, means_scores_pre, linewidth = '1.2', color = 'red')
35     plt.xlabel('Number of points removed')
36     plt.ylabel('Mean Squared Error')
37     plt.show()
38     plt.close()

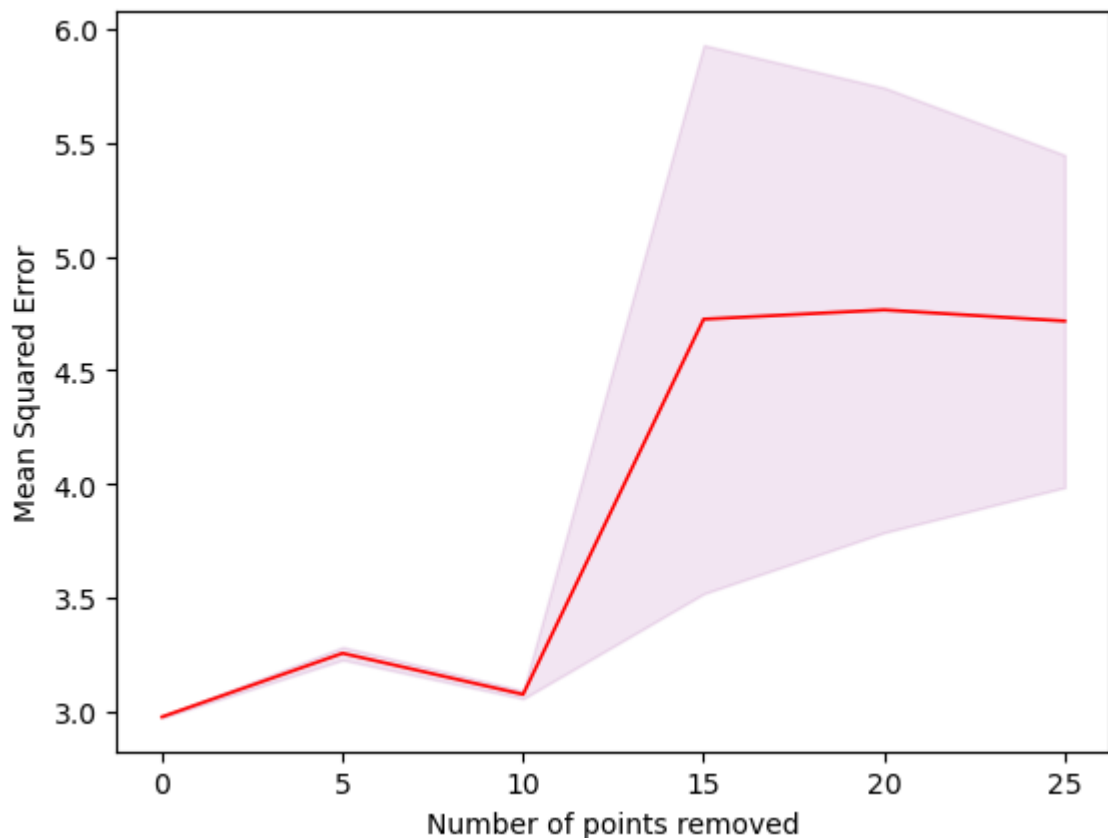
```

```

C:\Users\kayad\AppData\Local\Temp\ipykernel_27100\3853094899.py:38: RuntimeWarning: overflow encountered in exp
  acceptance_probability = min(np.exp(-(delta/temp)),1)#Calculate acceptance probability
C:\Users\kayad\AppData\Local\Temp\ipykernel_27100\3853094899.py:38: RuntimeWarning: overflow encountered in scalar divide
  acceptance_probability = min(np.exp(-(delta/temp)),1)#Calculate acceptance probability
C:\Users\kayad\AppData\Local\Temp\ipykernel_27100\3853094899.py:38: RuntimeWarning: divide by zero encountered in scalar divide
  acceptance_probability = min(np.exp(-(delta/temp)),1)#Calculate acceptance probability
C:\Users\kayad\AppData\Local\Temp\ipykernel_27100\3853094899.py:38: RuntimeWarning: invalid value encountered in scalar divide
  acceptance_probability = min(np.exp(-(delta/temp)),1)#Calculate acceptance probability
C:\Users\kayad\AppData\Local\Temp\ipykernel_27100\528580857.py:11: RuntimeWarning: overflow encountered in square
  err1 = (x1[indx_x] - x2[indx_x])**2
C:\Users\kayad\AppData\Local\Temp\ipykernel_27100\528580857.py:12: RuntimeWarning: overflow encountered in square
  err2 = (y1[indx_y] - y2[indx_y])**2

```

<Figure size 1920x1440 with 0 Axes>



```

In [419]: 1 # Point extrema removed with aggressive cooling, prey population and MA
2 simulations = 2
3 iterations = 1
4 init_temp = 20
5 cooling = 0.1 # cooling strategy more aggressive
6 t = data[:,0]
7 time = t
8 inp = data[:,1:3]
9
10 all_scores_preymae = []
11 for counter in range(25):
12     # storage
13     params_preymae = []
14     obj_results_preymae = []
15     # point removal loop (removing points at steps of 2)
16     npoints = np.arange(0,30,5)
17     for p_removed in npoints:
18         best_params_found_preymae = multirun_SA_point_removal
19         params_preymae.append(best_params_found_preymae)
20         obj_results_preymae.append(best_params_found_preymae)
21     all_scores_preymae.append(obj_results_preymae)
22
23 means_scores_preymae = []
24 variance_scores_preymae = []
25 for index in range(len(npoints)):
26     current_score_count_mae = np.array(all_scores_preymae[:,index])
27     means_scores_preymae.append(np.mean(current_score_count_mae))
28     variance_scores_preymae.append(statistics.variance(current_score_count_mae))
29 ci_mae = 1.96 * np.array(variance_scores_preymae)/25
30
31 plt.figure(dpi = 300)
32 fig, ax = plt.subplots()
33 ax.plot(npoints,means_scores_preymae, linewidth = '1.2', color = 'red')
34 ax.fill_between(npoints, (np.array(means_scores_preymae)-ci_mae), (np.array(means_scores_preymae)+ci_mae), color = 'red')
35 plt.xlabel('Number of points removed')
36 plt.ylabel('Mean Absolute Error')
37 plt.show()
38 plt.close()

```

C:\Users\kayad\AppData\Local\Temp\ipykernel_27100\3853094899.py:38: RuntimeWarning: overflow encountered in exp

acceptance_probability = min(np.exp(-(delta/temp)),1)#Calculate acceptance probability

C:\Users\kayad\AppData\Local\Temp\ipykernel_27100\3853094899.py:38: RuntimeWarning: overflow encountered in scalar divide

acceptance_probability = min(np.exp(-(delta/temp)),1)#Calculate acceptance probability

C:\Users\kayad\AppData\Local\Temp\ipykernel_27100\3853094899.py:38: RuntimeWarning: divide by zero encountered in scalar divide

acceptance_probability = min(np.exp(-(delta/temp)),1)#Calculate acceptance probability

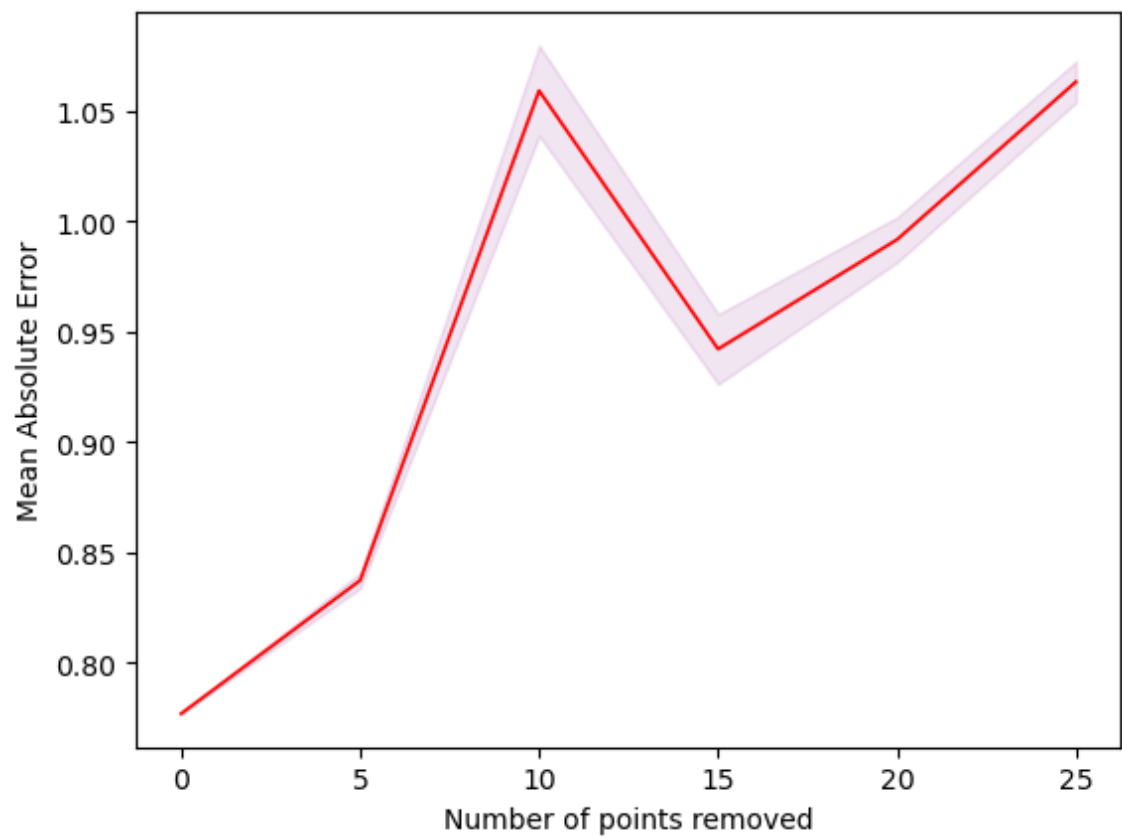
C:\Users\kayad\AppData\Local\Temp\ipykernel_27100\3853094899.py:38: RuntimeWarning: invalid value encountered in scalar divide

acceptance_probability = min(np.exp(-(delta/temp)),1)#Calculate acceptance probability

C:\Users\kayad\AppData\Local\Temp\ipykernel_27100\528580857.py:35: RuntimeWarning: Mean of empty slice

return np.nanmean([err])

<Figure size 1920x1440 with 0 Axes>



```

In [420]: 1 # Point extrema removed with not aggressive cooling, prey population an
2 simulations = 2
3 iterations = 1
4 init_temp = 20
5 cooling = 0.9 # cooling strategy more aggressive
6 t = data[:,0]
7 time = t
8 inp = data[:,1:3]
9
10 all_scores_prej_MAE = []
11 for counter in range(25):
12     # storage
13     params_prej_MAE = []
14     objective_results_prej_MAE = []
15     # point removal loop (removing points at steps of 2)
16     npoints = np.arange(0,30,5)
17     for p_removed in npoints:
18         best_parameters_found_prej_removal_MAE = multirun_SA_point_remo
19         params_prej_MAE.append(best_parameters_found_prej_removal_MAE)
20         objective_results_prej_MAE.append(best_parameters_found_prej_re
21         all_scores_prej_MAE.append(objective_results_prej_MAE)
22
23 means_scores_prej_MAE = []
24 variance_scores_prej_MAE = []
25 for index in range(len(npoints)):
26     current_score_count_MAE = np.array(all_scores_prej_MAE[:,index])
27     means_scores_prej_MAE.append(np.mean(current_score_count_MAE))
28     variance_scores_prej_MAE.append(statistics.variance(current_score_c
29 ci_MAE = 1.96 * np.array(variance_scores_prej_MAE)/25
30
31 plt.figure(dpi = 300)
32 fig, ax = plt.subplots()
33 ax.plot(npoints,means_scores_prej_MAE, linewidth = '1.2', color = 'red')
34 ax.fill_between(npoints, (np.array(means_scores_prej_MAE)-ci_MAE), (np.
35 plt.xlabel('Number of points removed')
36 plt.ylabel('Mean Absolute Error')
37 plt.show()
38 plt.close()

```

C:\Users\kayad\AppData\Local\Temp\ipykernel_27100\3853094899.py:38: RuntimeWarning: overflow encountered in exp

acceptance_probability = min(np.exp(-(delta/temp)),1)#Calculate acceptance probability

C:\Users\kayad\AppData\Local\Temp\ipykernel_27100\1517521670.py:4: RuntimeWarning: overflow encountered in scalar multiply

dxdt = (alpha * x) - (beta * x * y) # Prey ODE

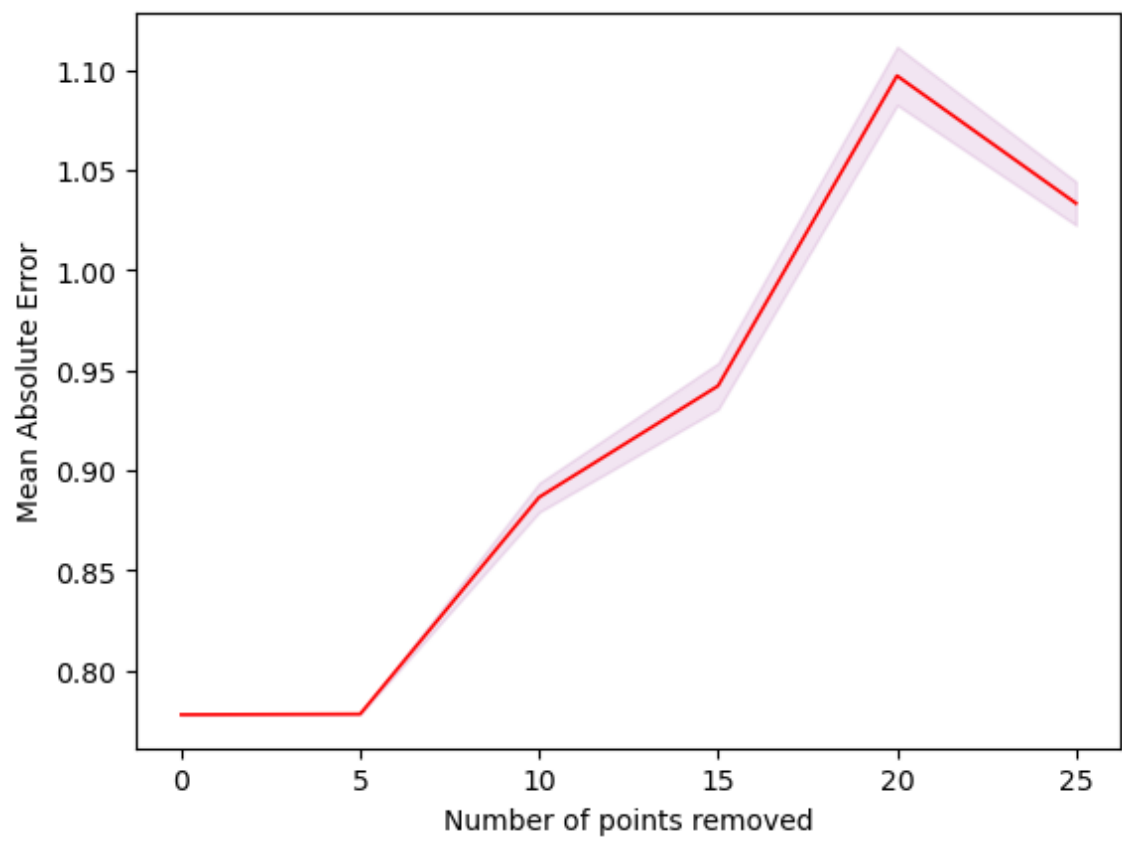
C:\Users\kayad\AppData\Local\Temp\ipykernel_27100\1517521670.py:5: RuntimeWarning: overflow encountered in scalar multiply

dydt = (delta * x * y) - (gamma * y) # Predator ODE

C:\Users\kayad\AppData\Local\Temp\ipykernel_27100\528580857.py:35: RuntimeWarning: Mean of empty slice

return np.nanmean([err])

<Figure size 1920x1440 with 0 Axes>



```

In [428]: 1 # Point extrema removed with not aggressive cooling, prey population an
2 simulations = 2
3 iterations = 1
4 init_temp = 20
5 cooling = 0.9 # cooling strategy more aggressive
6 t = data[:,0]
7 time = t
8 inp = data[:,1:3]
9
10 all_scores_prej_MAE = []
11 for counter in range(25):
12     # storage
13     params_prej_MAE = []
14     objective_results_prej_MAE = []
15     # point removal loop (removing points at steps of 2)
16     npoints = np.arange(0,30,2)
17     for p_removed in npoints:
18         best_parameters_found_prej_removal_MAE = multirun_SA_point_remo
19         params_prej_MAE.append(best_parameters_found_prej_removal_MAE)
20         objective_results_prej_MAE.append(best_parameters_found_prej_re
21         all_scores_prej_MAE.append(objective_results_prej_MAE)
22
23 means_scores_prej_MAE = []
24 variance_scores_prej_MAE = []
25 for index in range(len(npoints)):
26     current_score_count_MAE = np.array(all_scores_prej_MAE[:,index])
27     means_scores_prej_MAE.append(np.mean(current_score_count_MAE))
28     variance_scores_prej_MAE.append(statistics.variance(current_score_c
29 ci_MAE = 1.96 * np.array(variance_scores_prej_MAE)/25
30
31 plt.figure(dpi = 300)
32 fig, ax = plt.subplots()
33 ax.plot(npoints,means_scores_prej_MAE, linewidth = '1.2', color = 'red')
34 ax.fill_between(npoints, (np.array(means_scores_prej_MAE)-ci_MAE), (np.
35 plt.xlabel('Number of points removed')
36 plt.ylabel('Mean Absolute Error')
37 plt.show()
38 plt.close()

```

C:\Users\kayad\AppData\Local\Temp\ipykernel_27100\3853094899.py:38: RuntimeWarning: overflow encountered in exp

acceptance_probability = min(np.exp(-(delta/temp)),1)#Calculate acceptance probability

C:\Users\kayad\AppData\Local\Temp\ipykernel_27100\528580857.py:35: RuntimeWarning: Mean of empty slice

return np.nanmean([err])

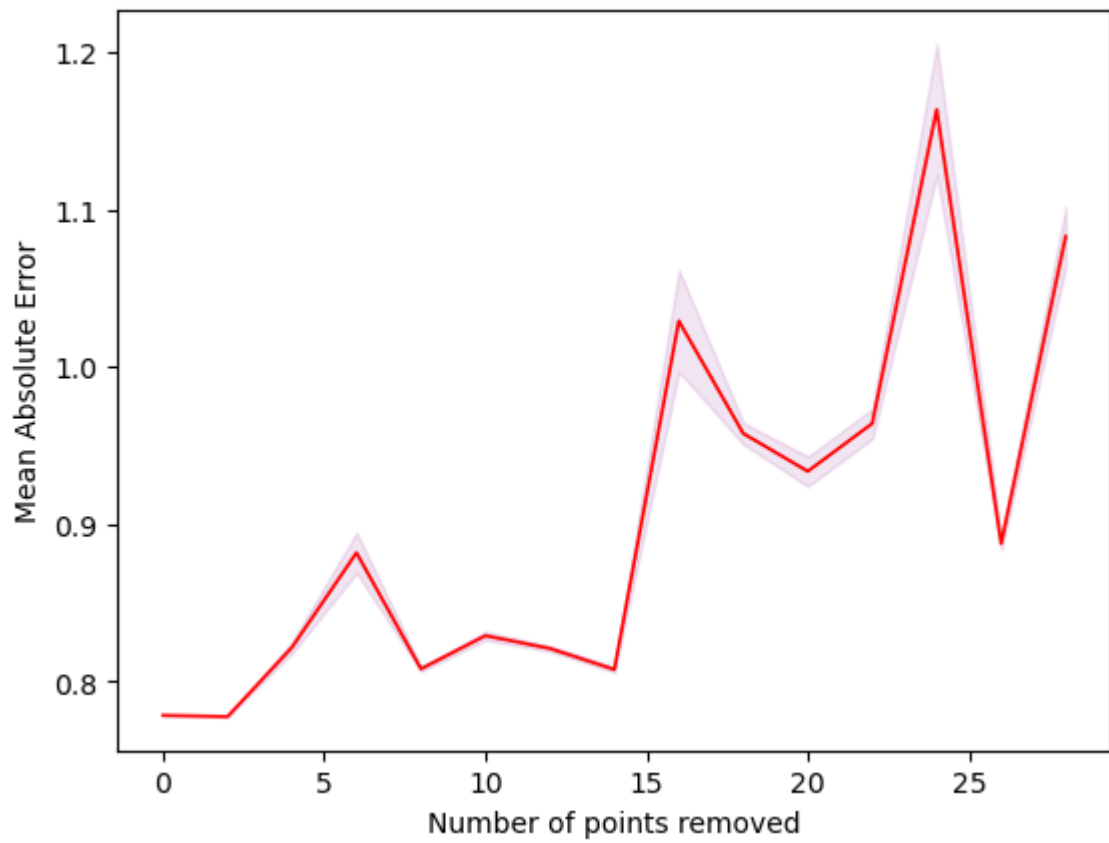
C:\Users\kayad\AppData\Local\Temp\ipykernel_27100\1517521670.py:4: RuntimeWarning: overflow encountered in scalar multiply

dxdt = (alpha * x) - (beta * x * y) # Prey ODE

C:\Users\kayad\AppData\Local\Temp\ipykernel_27100\1517521670.py:5: RuntimeWarning: overflow encountered in scalar multiply

dydt = (delta * x * y) - (gamma * y) # Predator ODE

<Figure size 1920x1440 with 0 Axes>



In [421]:

```

1  # Point extrema removed with aggressive cooling for both species
2  simulations = 2
3  iterations = 1
4  init_temp = 20
5  cooling = 0.1 # cooling strategy more aggressive
6  t = data[:,0]
7  time = t
8  inp = data[:,1:3]
9
10 all_scores_predprey = []
11 for counter in range(25):
12     # storage
13     params_predprey = []
14     objective_results_predprey = []
15     # point removal loop (removing points at steps of 2)
16     npoints = np.arange(0,60,5)
17     for p_removed in npoints:
18         best_parameters_found_predprey_removal = multirun_SA_point_removal
19         params_predprey.append(best_parameters_found_predprey_removal)
20         objective_results_predprey.append(best_parameters_found_predprey_removal)
21         all_scores_predprey.append(objective_results_predprey)
22         #plt.plot(npoints,objective_results_predprey, linewidth = '0.6', color = 'red')
23 means_scores_predprey = []
24 variance_scores_predprey = []
25 for index in range(len(npoints)):
26     current_score_count_predprey = np.array(all_scores_predprey[:,index])
27     means_scores_predprey.append(np.mean(current_score_count_predprey))
28     variance_scores_predprey.append(statistics.variance(current_score_count_predprey))
29 ci_predprey = 1.96 * np.array(variance_scores_predprey)/15
30 plt.figure(dpi = 300)
31 fig, ax = plt.subplots()
32 ax.plot(npoints,means_scores_predprey, linewidth = '1.2', color = 'red')
33 ax.fill_between(npoints, (np.array(means_scores_predprey)-ci_predprey),
34                (np.array(means_scores_predprey)+ci_predprey),
35                #plt.plot(npoints, means_scores_predprey, linewidth = '1.2', color = 'red')
36                plt.xlabel('Number of points removed')
37                plt.ylabel('Mean Squared Error')
38                plt.show()
39                plt.close()

```

C:\Users\kayad\AppData\Local\Temp\ipykernel_27100\3853094899.py:38: RuntimeWarning: overflow encountered in exp

acceptance_probability = min(np.exp(-(delta/temp)),1)#Calculate acceptance probability

C:\Users\kayad\AppData\Local\Temp\ipykernel_27100\3853094899.py:38: RuntimeWarning: overflow encountered in scalar divide

acceptance_probability = min(np.exp(-(delta/temp)),1)#Calculate acceptance probability

C:\Users\kayad\AppData\Local\Temp\ipykernel_27100\3853094899.py:38: RuntimeWarning: divide by zero encountered in scalar divide

acceptance_probability = min(np.exp(-(delta/temp)),1)#Calculate acceptance probability

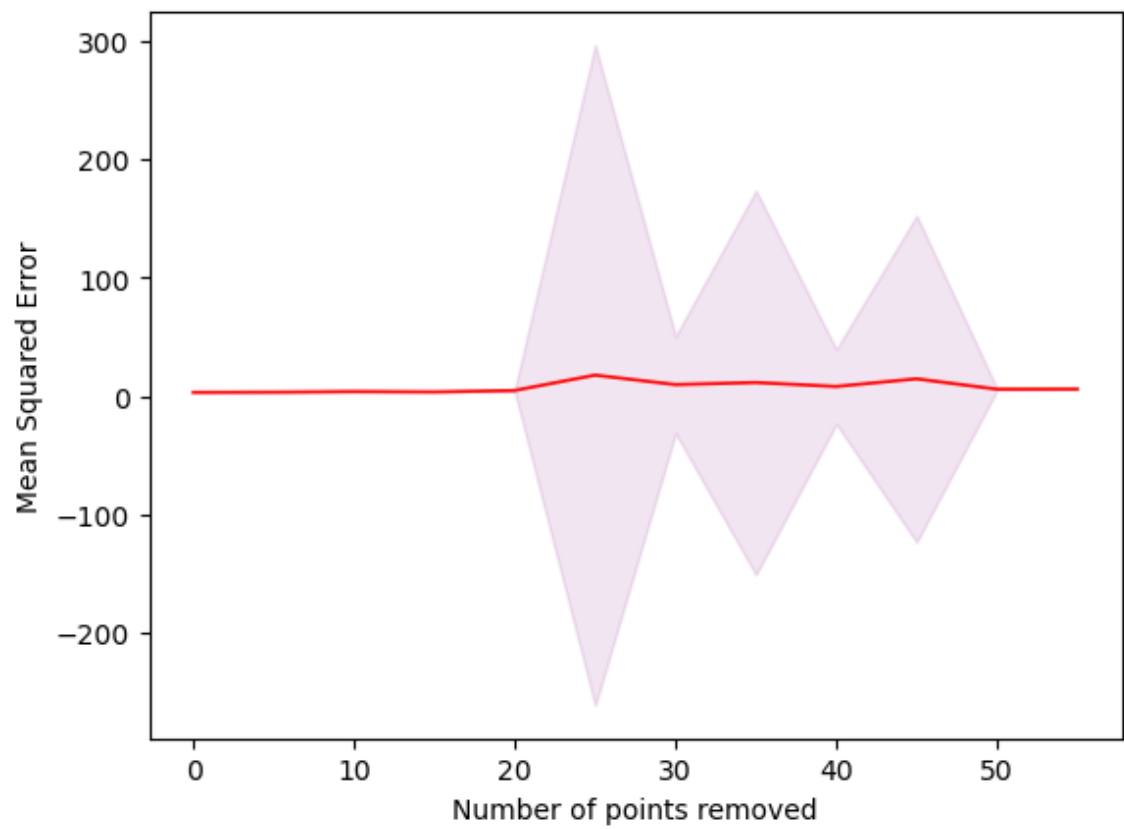
C:\Users\kayad\AppData\Local\Temp\ipykernel_27100\3853094899.py:38: RuntimeWarning: invalid value encountered in scalar divide

acceptance_probability = min(np.exp(-(delta/temp)),1)#Calculate acceptance probability

C:\Users\kayad\AppData\Local\Temp\ipykernel_27100\528580857.py:15: RuntimeWarning: Mean of empty slice

return np.nanmean([err])

<Figure size 1920x1440 with 0 Axes>



In [422]:

```

1  # Point extrema removed with aggressive cooling for both species
2  simulations = 2
3  iterations = 1
4  init_temp = 20
5  cooling = 0.9 # cooling strategy more aggressive
6  t = data[:,0]
7  time = t
8  inp = data[:,1:3]
9
10 all_scores_predprey = []
11 for counter in range(25):
12     # storage
13     params_predprey = []
14     objective_results_predprey = []
15     # point removal loop (removing points at steps of 2)
16     npoints = np.arange(0,60,5)
17     for p_removed in npoints:
18         best_parameters_found_predprey_removal = multirun_SA_point_removal
19         params_predprey.append(best_parameters_found_predprey_removal)
20         objective_results_predprey.append(best_parameters_found_predprey_removal)
21         all_scores_predprey.append(objective_results_predprey)
22         #plt.plot(npoints,objective_results_predprey, linewidth = '0.6', color = 'red')
23 means_scores_predprey = []
24 variance_scores_predprey = []
25 for index in range(len(npoints)):
26     current_score_count_predprey = np.array(all_scores_predprey[:,index])
27     means_scores_predprey.append(np.mean(current_score_count_predprey))
28     variance_scores_predprey.append(statistics.variance(current_score_count_predprey))
29 ci_predprey = 1.96 * np.array(variance_scores_predprey)/15
30 plt.figure(dpi = 300)
31 fig, ax = plt.subplots()
32 ax.plot(npoints,means_scores_predprey, linewidth = '1.2', color = 'red')
33 ax.fill_between(npoints, (np.array(means_scores_predprey)-ci_predprey),
34                (np.array(means_scores_predprey)+ci_predprey), color = 'red')
35 plt.xlabel('Number of points removed')
36 plt.ylabel('Mean Squared Error')
37 plt.show()
38 plt.close()

```

C:\Users\kayad\AppData\Local\Temp\ipykernel_27100\3853094899.py:38: RuntimeWarning: overflow encountered in exp

acceptance_probability = min(np.exp(-(delta/temp)),1)#Calculate acceptance probability

C:\Users\kayad\AppData\Local\Temp\ipykernel_27100\528580857.py:15: RuntimeWarning: Mean of empty slice

return np.nanmean([err])

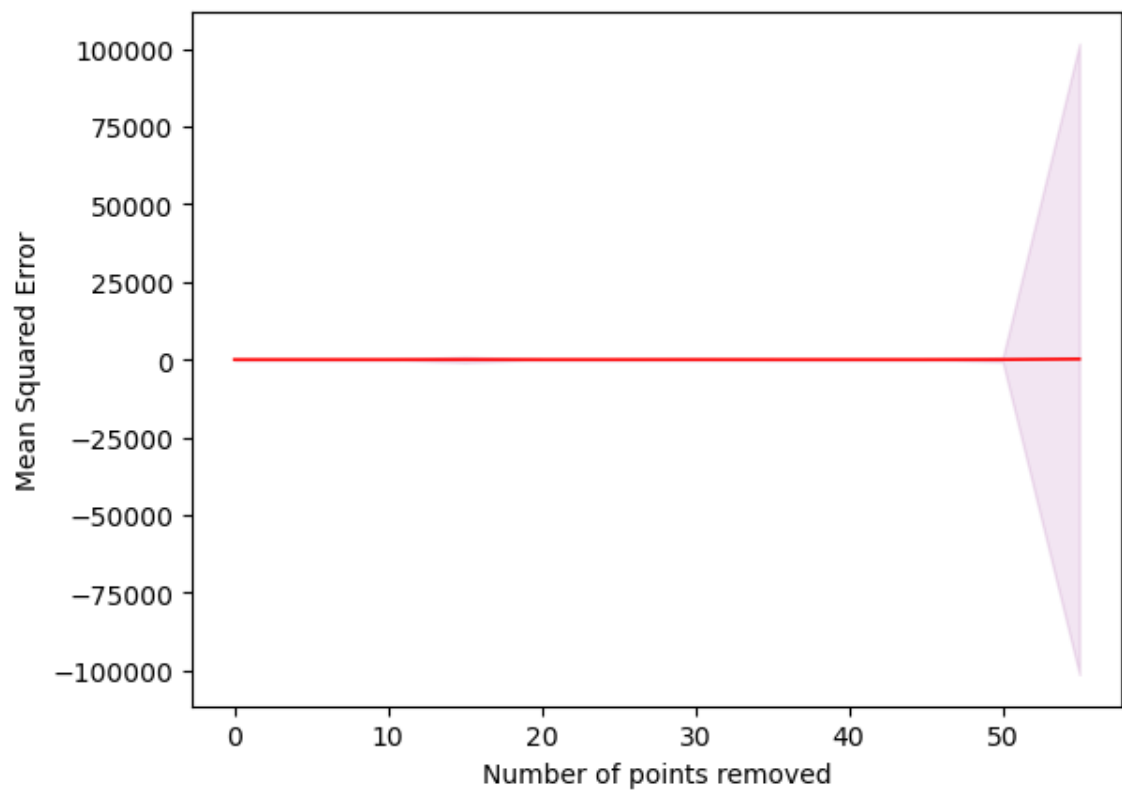
C:\Users\kayad\AppData\Local\Temp\ipykernel_27100\528580857.py:11: RuntimeWarning: overflow encountered in square

err1 = (x1[indx_x] - x2[indx_x])**2

C:\Users\kayad\AppData\Local\Temp\ipykernel_27100\528580857.py:12: RuntimeWarning: overflow encountered in square

err2 = (y1[indx_y] - y2[indx_y])**2

<Figure size 1920x1440 with 0 Axes>



```

In [426]: 1 # Midpoint removal using MSE and aggressive cooling for the prey popula
2
3 simulations = 2
4 iterations = 1
5 init_temp = 20
6 cooling = 0.1 # cooling strategy more aggressive
7 t = data[:,0]
8 time = t
9 inp = data[:,1:3]
10
11 all_scores_preymid = []
12 for counter in range(25):
13     # storage
14     params_preymid = []
15     objective_results_preymid = []
16     # point removal loop (removing points at steps of 2)
17     npoints = np.arange(0,75,5)
18     for p_removed in npoints:
19         best_parameters_found_preymidremoval = multirun_SA_point_removal
20         params_preymid.append(best_parameters_found_preymidremoval[
21         objective_results_preymid.append(best_parameters_found_preymidremoval[
22         all_scores_preymid.append(objective_results_preymid)
23 means_scores_preymid = []
24 variance_scores_preymid = []
25 for index in range(len(npoints)):
26     current_score_count_preymid = np.array(all_scores_preymid)[: ,index]
27     means_scores_preymid.append(np.mean(current_score_count_preymid))
28     variance_scores_preymid.append(statistics.variance(current_score_count_preymid))
29 ci_preymid = 1.96 * np.array(variance_scores_preymid)/25
30 plt.figure(dpi = 300)
31 fig, ax = plt.subplots()
32 ax.plot(npoints,means_scores_preymid, linewidth = '1.2', color = 'red')
33 ax.fill_between(npoints, (np.array(means_scores_preymid)-ci_preymid),
34 plt.xlabel('Number of points removed')
35 plt.ylabel('Mean Squared Error')
36 plt.show()
37 plt.close()

```

C:\Users\kayad\AppData\Local\Temp\ipykernel_27100\3853094899.py:38: RuntimeWarning: overflow encountered in exp

acceptance_probability = min(np.exp(-(delta/temp)),1)#Calculate acceptance probability

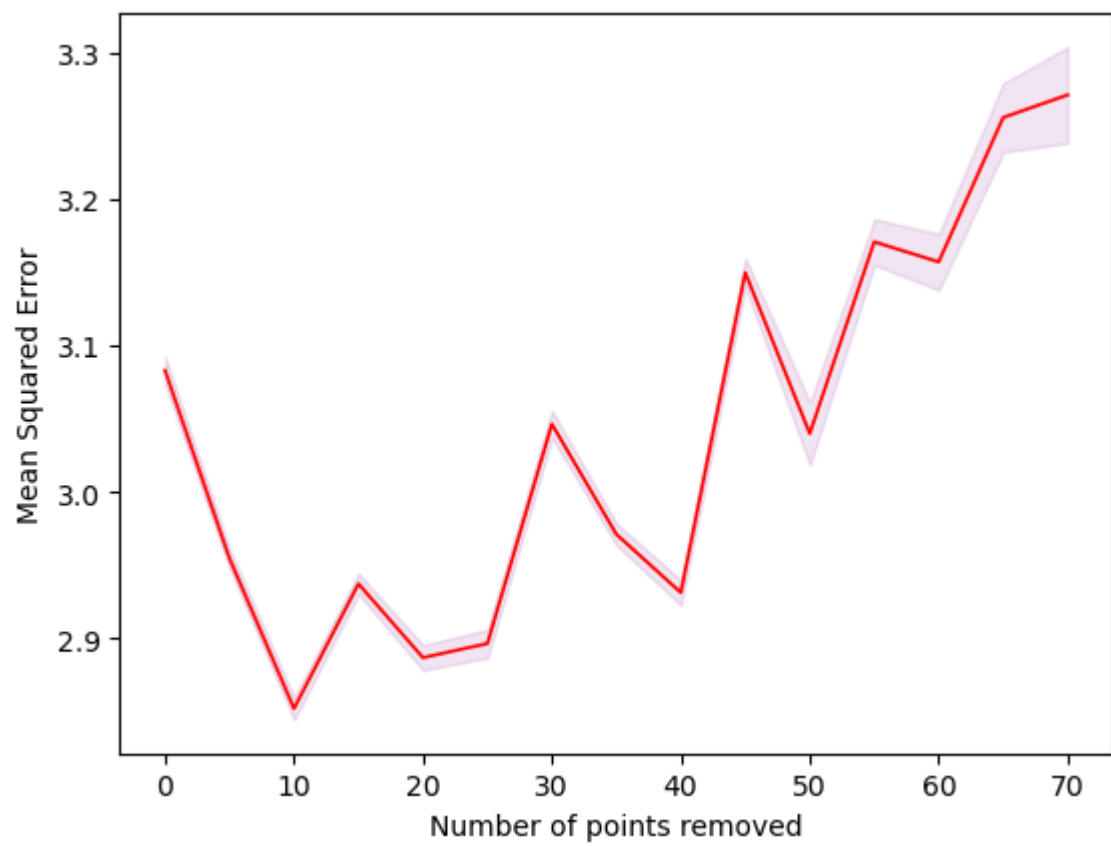
C:\Users\kayad\AppData\Local\Temp\ipykernel_27100\3853094899.py:38: RuntimeWarning: overflow encountered in scalar divide

acceptance_probability = min(np.exp(-(delta/temp)),1)#Calculate acceptance probability

C:\Users\kayad\AppData\Local\Temp\ipykernel_27100\3853094899.py:38: RuntimeWarning: divide by zero encountered in scalar divide

acceptance_probability = min(np.exp(-(delta/temp)),1)#Calculate acceptance probability

<Figure size 1920x1440 with 0 Axes>



```

In [427]: 1 # Midpoint removal using MSE and not aggressive cooling for both popula
2
3 simulations = 2
4 iterations = 1
5 init_temp = 20
6 cooling = 0.1 # cooling strategy more aggressive
7 t = data[:,0]
8 time = t
9 inp = data[:,1:3]
10
11 all_scores_predprey_mid = []
12 for counter in range(25):
13     # storage
14     params_predprey_mid = []
15     objective_results_predprey_mid = []
16     # point removal loop
17     npoints = np.arange(0,145,5)
18     for p_removed in npoints:
19         best_parameters_found_predprey_midremoval = multirun_SA_point_r
20         params_predprey_mid.append(best_parameters_found_predprey_mid
21         objective_results_predprey_mid.append(best_parameters_found_pre
22         all_scores_predprey_mid.append(objective_results_predprey_mid)
23 means_scores_predprey_mid = []
24 variance_scores_predprey_mid = []
25 for index in range(len(npoints)):
26     current_score_count_predprey_mid = np.array(all_scores_predprey_mid
27     means_scores_predprey_mid.append(np.mean(current_score_count_predpr
28     variance_scores_predprey_mid.append(statistics.variance(current_sco
29 ci_predprey_mid = 1.96 * np.array(variance_scores_predprey_mid)/25
30 plt.figure(dpi = 300)
31 fig, ax = plt.subplots()
32 ax.plot(npoints,means_scores_predprey_mid, linewidth = '1.2', color = '
33 ax.fill_between(npoints, (np.array(means_scores_predprey_mid)-ci_predpr
34 plt.xlabel('Number of points removed')
35 plt.ylabel('Mean Squared Error')
36 plt.show()
37 plt.close()

```

C:\Users\kayad\AppData\Local\Temp\ipykernel_27100\3853094899.py:38: RuntimeWarning: overflow encountered in exp

acceptance_probability = min(np.exp(-(delta/temp)),1)#Calculate acceptance probability

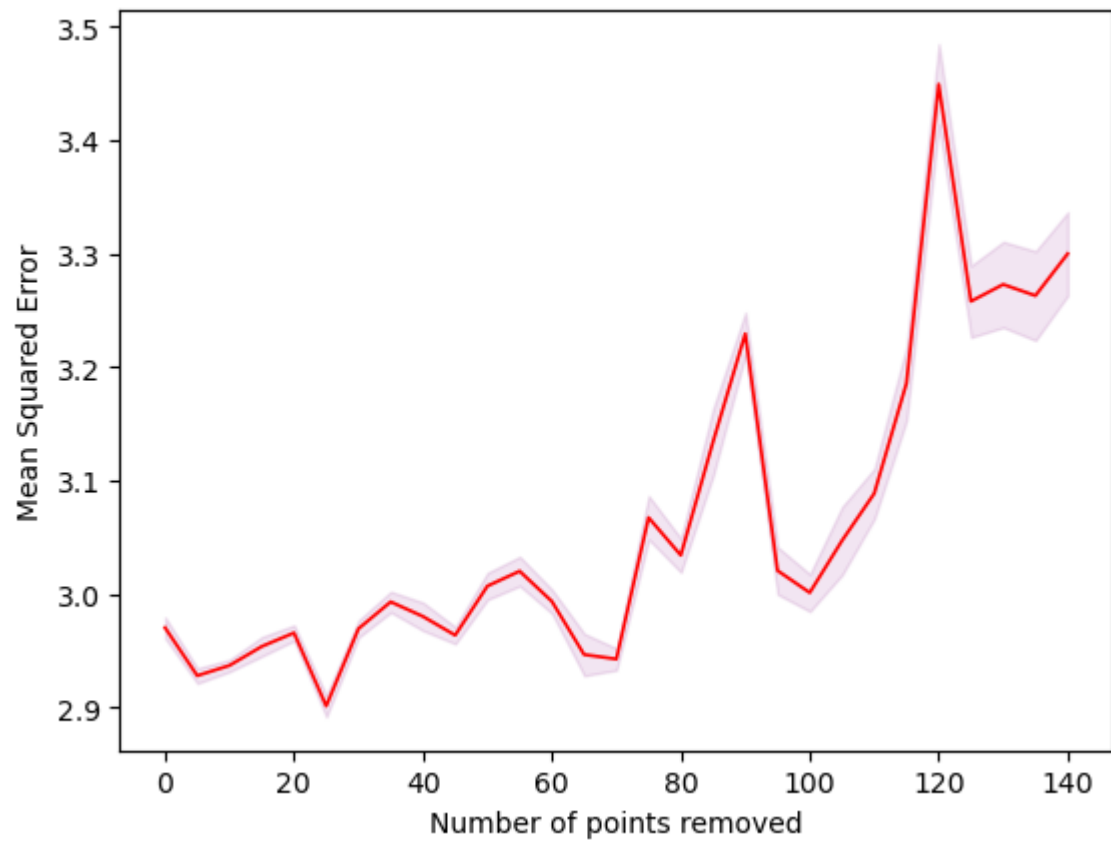
C:\Users\kayad\AppData\Local\Temp\ipykernel_27100\3853094899.py:38: RuntimeWarning: overflow encountered in scalar divide

acceptance_probability = min(np.exp(-(delta/temp)),1)#Calculate acceptance probability

C:\Users\kayad\AppData\Local\Temp\ipykernel_27100\3853094899.py:38: RuntimeWarning: divide by zero encountered in scalar divide

acceptance_probability = min(np.exp(-(delta/temp)),1)#Calculate acceptance probability

<Figure size 1920x1440 with 0 Axes>




```

In [457]: 1 # Point extrema removed with aggressive cooling MSE2 function prey popu
2
3 simulations = 2
4 iterations = 1
5 init_temp = 20
6 cooling = 0.1 # cooling strategy more aggressive
7 t = data[:,0]
8 time = t
9 inp = data[:,1:3]
10
11 all_scores_pre = []
12 all_params_pre = []
13 for counter in range(25):
14     # storage
15     params_pre = []
16     objective_results_pre = []
17     # point removal loop
18     npoints = np.arange(0,30,5)
19     for p_removed in npoints:
20         best_parameters_found_pre_removal = multirun_SA_point_removal(
21             params_pre.append(best_parameters_found_pre_removal[0])
22             objective_results_pre.append(best_parameters_found_pre_removal[1])
23         all_scores_pre.append(objective_results_pre)
24         all_params_pre.append(params_pre)
25         #plt.plot(npoints ,objective_results_pre, linewidth = '0.6', color = 'blue')
26 means_params_pre = []
27
28
29 for index in range(len(npoints)):
30     current_params_pre = np.array(all_params_pre)[:,index]
31     means_params_pre.append([np.mean(current_params_pre[:,0]), np.mean(current_params_pre[:,1]), np.mean(current_params_pre[:,2]), np.mean(current_params_pre[:,3])])
32
33 alpha_all_means = np.array(means_params_pre)[:,0]
34 beta_all_means = np.array(means_params_pre)[:,1]
35 delta_all_means = np.array(means_params_pre)[:,2]
36 gamma_all_means = np.array(means_params_pre)[:,3]
37
38 variance_alpha_pre = statistics.variance(alpha_all_means)
39 variance_beta_pre = statistics.variance(beta_all_means)
40 variance_delta_pre = statistics.variance(delta_all_means)
41 variance_gamma_pre = statistics.variance(gamma_all_means)
42
43 ci_alpha_pre = 1.96 * np.array(variance_alpha_pre)/25
44 ci_beta_pre = 1.96 * np.array(variance_beta_pre)/25
45 ci_delta_pre = 1.96 * np.array(variance_delta_pre)/25
46 ci_gamma_pre = 1.96 * np.array(variance_gamma_pre)/25
47
48 print(beta_all_means, npoints)
49 plt.figure(dpi = 300)
50 fig, axes = plt.subplots(2, 2, figsize=(15, 15))
51
52 bar_width = 0.5
53 plt.subplot(221)
54 plt.title(r'$\alpha$')
55 plt.bar(npoints, alpha_all_means, yerr = ci_alpha_pre,color = "orange")
56 plt.subplot(222)
57 plt.title(r'$\beta$')
58 plt.bar(npoints, beta_all_means, yerr = ci_beta_pre,color = "orange",e
59 plt.subplot(223)
60 plt.title(r'$\delta$')
61 plt.bar(npoints, delta_all_means, yerr = ci_delta_pre,color = "orange")

```

```

62 plt.subplot(224)
63 plt.title(r'$\gamma$')
64 plt.bar(npoints, gamma_all_means, yerr = ci_gamma_pre, color = "orange")

```

C:\Users\kayad\AppData\Local\Temp\ipykernel_27100\3853094899.py:38: RuntimeWarning: overflow encountered in exp

acceptance_probability = min(np.exp(-(delta/temp)),1)#Calculate acceptance probability

C:\Users\kayad\AppData\Local\Temp\ipykernel_27100\3853094899.py:38: RuntimeWarning: overflow encountered in scalar divide

acceptance_probability = min(np.exp(-(delta/temp)),1)#Calculate acceptance probability

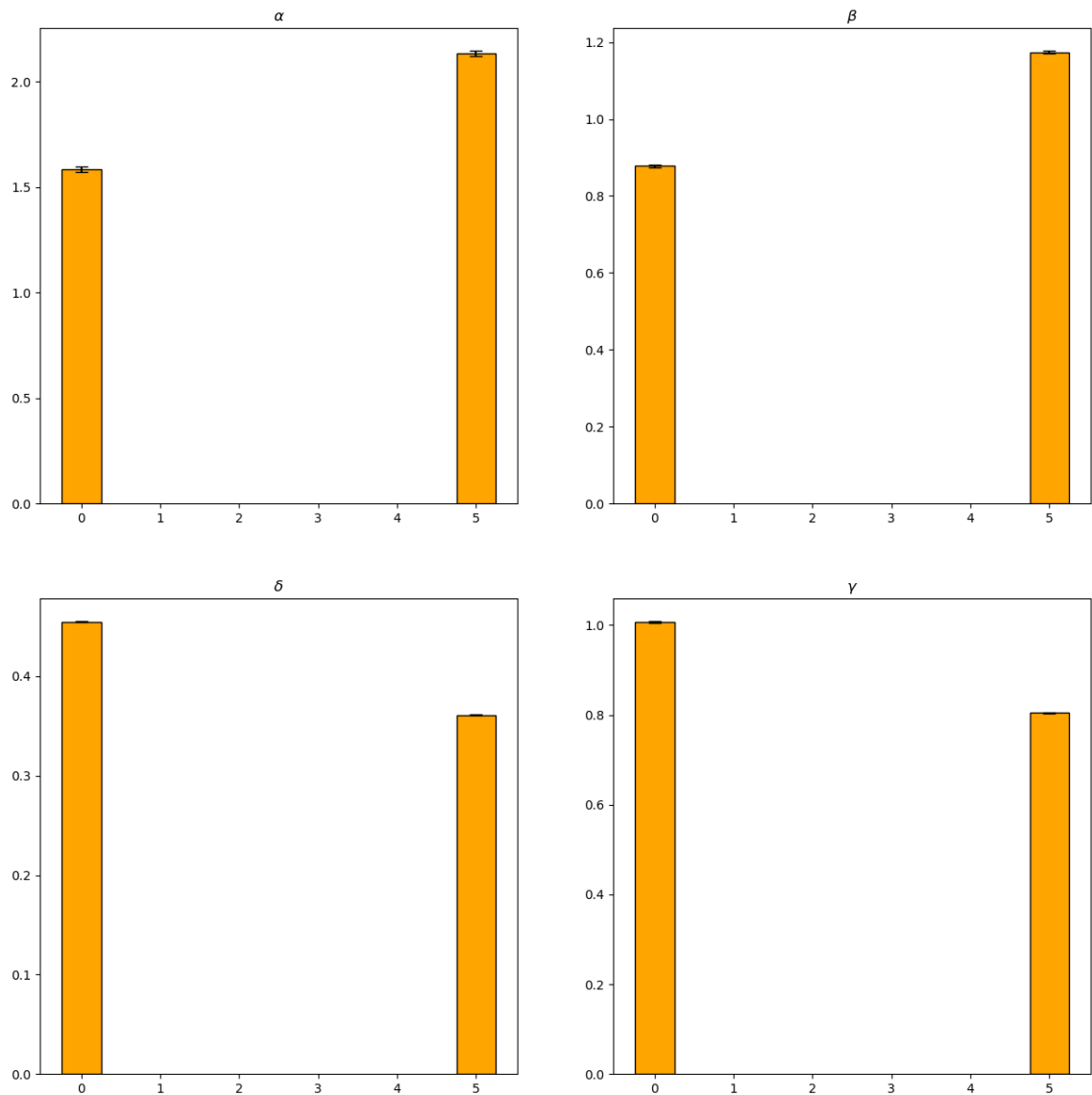
C:\Users\kayad\AppData\Local\Temp\ipykernel_27100\3853094899.py:38: RuntimeWarning: divide by zero encountered in scalar divide

acceptance_probability = min(np.exp(-(delta/temp)),1)#Calculate acceptance probability

[0.87772397 1.17375944] [0 5]

Out[457]: <BarContainer object of 2 artists>

<Figure size 1920x1440 with 0 Axes>




```

In [ ]: 1 # Point extrema removed with aggressive cooling MSE2 function both popu
2
3 simulations = 2
4 iterations = 1
5 init_temp = 20
6 cooling = 0.1 # cooling strategy more aggressive
7 t = data[:,0]
8 time = t
9 inp = data[:,1:3]
10
11 all_scores_pre = []
12 all_params_pre = []
13 for counter in range(25):
14     # storage
15     params_pre = []
16     objective_results_pre = []
17     # point removal loop
18     npoints = np.arange(0,60,5)
19     for p_removed in npoints:
20         best_parameters_found_pre_removal = multirun_SA_point_removal(
21             params_pre.append(best_parameters_found_pre_removal[0])
22             objective_results_pre.append(best_parameters_found_pre_removal[1])
23         all_scores_pre.append(objective_results_pre)
24         all_params_pre.append(params_pre)
25 means_params_pre = []
26
27
28 for index in range(len(npoints)):
29     current_params_pre = np.array(all_params_pre)[:,index]
30     means_params_pre.append([np.mean(current_params_pre[:,0]), np
31
32 alpha_all_means = np.array(means_params_pre)[:,0]
33 beta_all_means = np.array(means_params_pre)[:,1]
34 delta_all_means = np.array(means_params_pre)[:,2]
35 gamma_all_means = np.array(means_params_pre)[:,3]
36
37 variance_alpha_pre = statistics.variance(alpha_all_means)
38 variance_beta_pre = statistics.variance(beta_all_means)
39 variance_delta_pre = statistics.variance(delta_all_means)
40 variance_gamma_pre = statistics.variance(gamma_all_means)
41
42 ci_alpha_pre = 1.96 * np.array(variance_alpha_pre)/25
43 ci_beta_pre = 1.96 * np.array(variance_beta_pre)/25
44 ci_delta_pre = 1.96 * np.array(variance_delta_pre)/25
45 ci_gamma_pre = 1.96 * np.array(variance_gamma_pre)/25
46
47 print(beta_all_means, npoints)
48 plt.figure(dpi = 300)
49 fig, axes = plt.subplots(2, 2, figsize=(15, 15))
50
51 bar_width = 0.5
52 plt.subplot(221)
53 plt.title(r'$\alpha$')
54 plt.bar(npoints, alpha_all_means, yerr = ci_alpha_pre,color = "orange")
55 plt.subplot(222)
56 plt.title(r'$\beta$')
57 plt.bar(npoints, beta_all_means, yerr = ci_beta_pre,color = "orange",e
58 plt.subplot(223)
59 plt.title(r'$\delta$')
60 plt.bar(npoints, delta_all_means, yerr = ci_delta_pre,color = "orange")
61 plt.subplot(224)

```

```
62 plt.title(r'$\gamma$')  
63 plt.bar(npoints, gamma_all_means, yerr = ci_gamma_prey,color = "orange")
```



```

In [458]: 1 # Points removed randomly with aggressive cooling MSE2 function both p
2
3 simulations = 2
4 iterations = 1
5 init_temp = 20
6 cooling = 0.1 # cooling strategy more aggressive
7 t = data[:,0]
8 time = t
9 inp = data[:,1:3]
10
11 all_scores = []
12 all_params = []
13 for counter in range(25):
14     # storage
15     params = []
16     objective_results = []
17     # point removal loop
18     npoints = np.arange(0,100,5)
19     for p_removed in npoints:
20         best_parameters_found_removal = multirun_SA_point_removal(itera
21         params.append(best_parameters_found_removal[0])
22         objective_results.append(best_parameters_found_removal[1])
23     all_scores.append(objective_results)
24     all_params.append(params)
25 means_params = []
26
27
28 for index in range(len(npoints)):
29     current_params = np.array(all_params)[:,index]
30     means_params.append([np.mean(current_params[:,0]), np.mean(curr
31
32 alpha_all_means = np.array(means_params)[:,0]
33 beta_all_means = np.array(means_params)[:,1]
34 delta_all_means = np.array(means_params)[:,2]
35 gamma_all_means = np.array(means_params)[:,3]
36
37 variance_alpha = statistics.variance(alpha_all_means)
38 variance_beta = statistics.variance(beta_all_means)
39 variance_delta = statistics.variance(delta_all_means)
40 variance_gamma = statistics.variance(gamma_all_means)
41
42 ci_alpha = 1.96 * np.array(variance_alpha)/25
43 ci_beta = 1.96 * np.array(variance_beta)/25
44 ci_delta = 1.96 * np.array(variance_delta)/25
45 ci_gamma = 1.96 * np.array(variance_gamma)/25
46
47 print(beta_all_means, npoints)
48 plt.figure(dpi = 300)
49 fig, axes = plt.subplots(2, 2, figsize=(15, 15))
50
51 bar_width = 0.5
52 plt.subplot(221)
53 plt.title(r'$\alpha$')
54 plt.bar(npoints, alpha_all_means, yerr = ci_alpha,color = "orange",ec =
55 plt.subplot(222)
56 plt.title(r'$\beta$')
57 plt.bar(npoints, beta_all_means, yerr = ci_beta,color = "orange",ec =
58 plt.subplot(223)
59 plt.title(r'$\delta$')
60 plt.bar(npoints, delta_all_means, yerr = ci_delta,color = "orange",ec =
61 plt.subplot(224)

```



```

62 plt.title(r'$\gamma$')
63 plt.bar(npoints, gamma_all_means, yerr = ci_gamma,color = "orange",ec =

```

C:\Users\kayad\AppData\Local\Temp\ipykernel_27100\3853094899.py:38: RuntimeWarning: overflow encountered in exp

acceptance_probability = min(np.exp(-(delta/temp)),1)#Calculate acceptance probability

C:\Users\kayad\AppData\Local\Temp\ipykernel_27100\3853094899.py:38: RuntimeWarning: overflow encountered in scalar divide

acceptance_probability = min(np.exp(-(delta/temp)),1)#Calculate acceptance probability

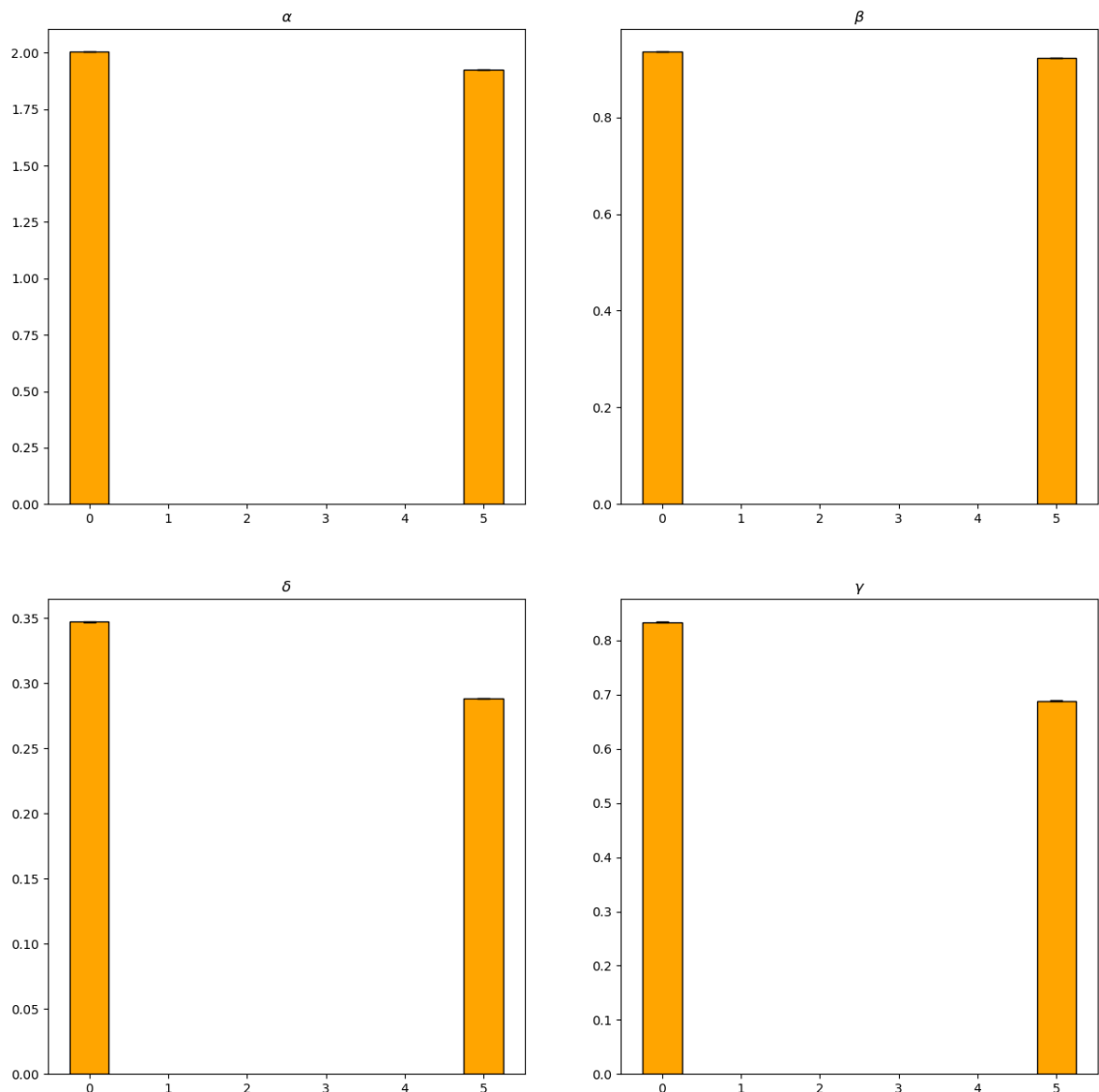
C:\Users\kayad\AppData\Local\Temp\ipykernel_27100\3853094899.py:38: RuntimeWarning: divide by zero encountered in scalar divide

acceptance_probability = min(np.exp(-(delta/temp)),1)#Calculate acceptance probability

[0.93545137 0.92197714] [0 5]

Out[458]: <BarContainer object of 2 artists>

<Figure size 1920x1440 with 0 Axes>




```

In [ ]: 1 # Random point removed with aggressive cooling MSE2 function prey popul
2
3 simulations = 2
4 iterations = 1
5 init_temp = 20
6 cooling = 0.1 # cooling strategy more aggressive
7 t = data[:,0]
8 time = t
9 inp = data[:,1:3]
10
11 all_scores_pre = []
12 all_params_pre = []
13 for counter in range(25):
14     # storage
15     params_pre = []
16     objective_results_pre = []
17     # point removal loop
18     npoints = np.arange(0,100,5)
19     for p_removed in npoints:
20         best_parameters_found_pre_removal = multirun_SA_point_removal(
21             params_pre.append(best_parameters_found_pre_removal[0])
22             objective_results_pre.append(best_parameters_found_pre_removal[1])
23         all_scores_pre.append(objective_results_pre)
24         all_params_pre.append(params_pre)
25 means_params_pre = []
26
27
28 for index in range(len(npoints)):
29     current_params_pre = np.array(all_params_pre)[:,index]
30     means_params_pre.append([np.mean(current_params_pre[:,0]), np
31
32 alpha_all_means = np.array(means_params_pre)[:,0]
33 beta_all_means = np.array(means_params_pre)[:,1]
34 delta_all_means = np.array(means_params_pre)[:,2]
35 gamma_all_means = np.array(means_params_pre)[:,3]
36
37 variance_alpha_pre = statistics.variance(alpha_all_means)
38 variance_beta_pre = statistics.variance(beta_all_means)
39 variance_delta_pre = statistics.variance(delta_all_means)
40 variance_gamma_pre = statistics.variance(gamma_all_means)
41
42 ci_alpha_pre = 1.96 * np.array(variance_alpha_pre)/25
43 ci_beta_pre = 1.96 * np.array(variance_beta_pre)/25
44 ci_delta_pre = 1.96 * np.array(variance_delta_pre)/25
45 ci_gamma_pre = 1.96 * np.array(variance_gamma_pre)/25
46
47 print(beta_all_means, npoints)
48 plt.figure(dpi = 300)
49 fig, axes = plt.subplots(2, 2, figsize=(15, 15))
50
51 bar_width = 0.5
52 plt.subplot(221)
53 plt.title(r'$\alpha$')
54 plt.bar(npoints, alpha_all_means, yerr = ci_alpha_pre,color = "orange")
55 plt.subplot(222)
56 plt.title(r'$\beta$')
57 plt.bar(npoints, beta_all_means, yerr = ci_beta_pre,color = "orange",e
58 plt.subplot(223)
59 plt.title(r'$\delta$')
60 plt.bar(npoints, delta_all_means, yerr = ci_delta_pre,color = "orange")
61 plt.subplot(224)

```

```
62 plt.title(r'$\gamma$')
63 plt.bar(npoints, gamma_all_means, yerr = ci_gamma_pre, color = "orange")
```

In [14]:

```
1 def MSE2(actual, predicted):
2     '''Mean squared error'''
3     x1 = actual[:, 0]
4     y1 = actual[:, 1]
5
6     #Getting useful indexes
7     indx_x = np.where(~np.isnan(x1))
8     indx_y = np.where(~np.isnan(y1))
9
10    x2, y2 = predicted[:, 0], predicted[:, 1]
11    err1 = (x1[indx_x] - x2[indx_x])**2
12    err2 = (y1[indx_y] - y2[indx_y])**2
13    err = np.concatenate([err1, err2])
14
15    return np.nanmean(err)
```

In [20]:

```

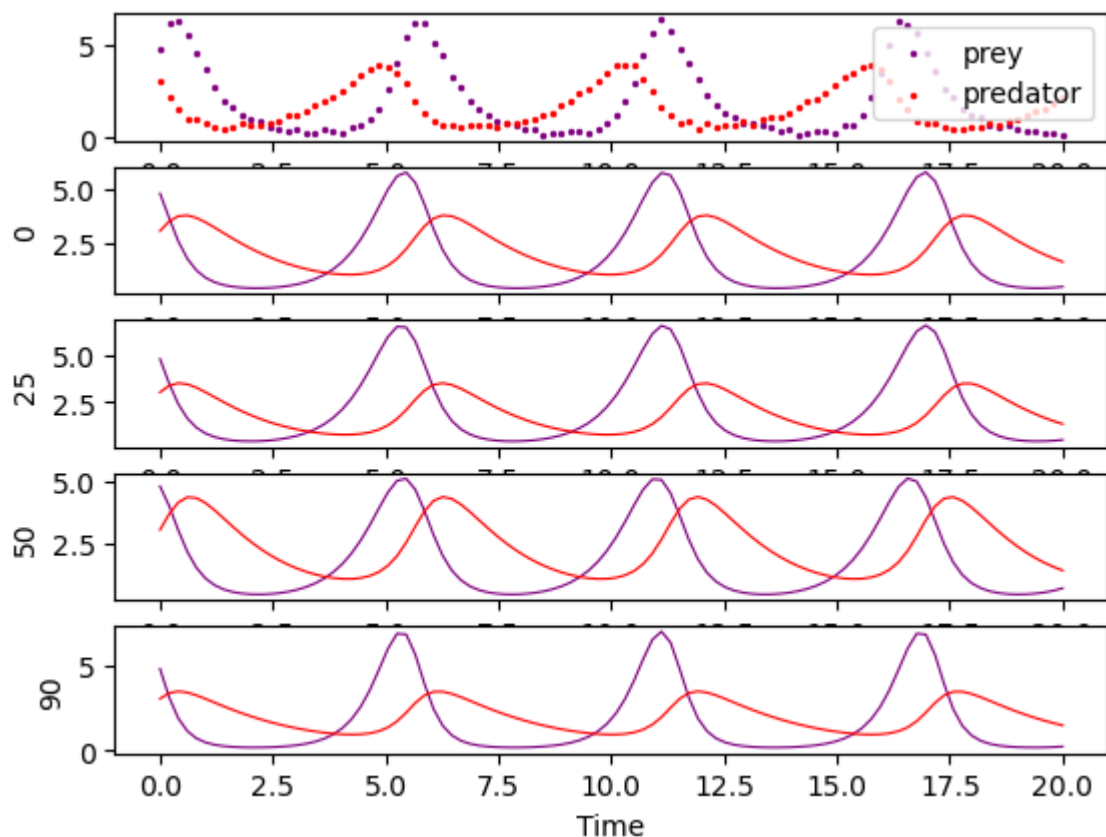
1 simulations = 2
2 iterations = 1
3 init_temp = 20
4 cooling = 0.1 # cooling strategy more aggressive
5 t = data[:,0]
6 time = t
7 inp = data[:,1:3]
8 cooling_constant = 0.10
9 parameters = [np.random.uniform(0,1), np.random.uniform(0,1), np.random.uniform(0,1)]
10
11 Result_full = inference_removal(initial_temp,cooling_constant, MSE2, simulations, iterations, parameters)
12 Result_quarter = inference_removal(initial_temp,cooling_constant, MSE2, simulations, iterations, parameters)
13 Result_half = inference_removal(initial_temp,cooling_constant, MSE2, simulations, iterations, parameters)
14 Result_90 = inference_removal(initial_temp,cooling_constant, MSE2, simulations, iterations, parameters)
15
16 Integration_full = predator_prey_integration(time,inp[0],Result_full[0])
17 Integration_quarter = predator_prey_integration(time,inp[0],Result_quarter[0])
18 Integration_half = predator_prey_integration(time,inp[0],Result_half[0])
19 Integration_90 = predator_prey_integration(time,inp[0],Result_90[0])
20
21 plt.plot(dpi = 300, figsize = (20, 5))
22 plt.subplot(511)
23 plt.scatter(time, inp.T[0], color = 'purple', s = 2, label = 'prey')
24 plt.scatter(time, inp.T[1], color = 'red', s = 2, label = 'predator')
25 plt.legend()
26 plt.subplot(512)
27 plt.plot(time, Integration_full[:,0], color = 'purple', label = 'prey',)
28 plt.plot(time, Integration_full[:,1], color = 'red', label = 'predator',)
29 plt.ylabel('0')
30 plt.subplot(513)
31 plt.plot(time, Integration_quarter[:,0], color = 'purple', label = 'prey',)
32 plt.plot(time, Integration_quarter[:,1], color = 'red', label = 'predator',)
33 plt.ylabel('25')
34 plt.subplot(514)
35 plt.plot(time, Integration_half[:,0], color = 'purple', label = 'prey',)
36 plt.plot(time, Integration_half[:,1], color = 'red', label = 'predator',)
37 plt.ylabel('50')
38 plt.subplot(515)
39 plt.plot(time, Integration_90[:,0], color = 'purple', label = 'prey',)
40 plt.plot(time, Integration_90[:,1], color = 'red', label = 'predator',)
41 plt.ylabel('90')
42 plt.xlabel('Time')
43 plt.show()
44 plt.close()
45

```

```

C:\Users\Aleks\AppData\Local\Temp\ipykernel_1540\1335953606.py:38: Runtime
Warning: overflow encountered in exp
  acceptance_probability = min(np.exp(-(delta/temp)),1)#Calculate acceptan
ce probability
C:\Users\Aleks\AppData\Local\Temp\ipykernel_1540\1335953606.py:38: Runtime
Warning: overflow encountered in scalar divide
  acceptance_probability = min(np.exp(-(delta/temp)),1)#Calculate acceptan
ce probability
C:\Users\Aleks\AppData\Local\Temp\ipykernel_1540\1335953606.py:38: Runtime
Warning: divide by zero encountered in scalar divide
  acceptance_probability = min(np.exp(-(delta/temp)),1)#Calculate acceptan
ce probability
C:\Users\Aleks\AppData\Local\Temp\ipykernel_1540\2635120008.py:22: Matplot
libDeprecationWarning: Auto-removal of overlapping axes is deprecated sinc
e 3.6 and will be removed two minor releases later; explicitly call ax.rem
ove() as needed.
  plt.subplot(511)

```



In [21]:

```

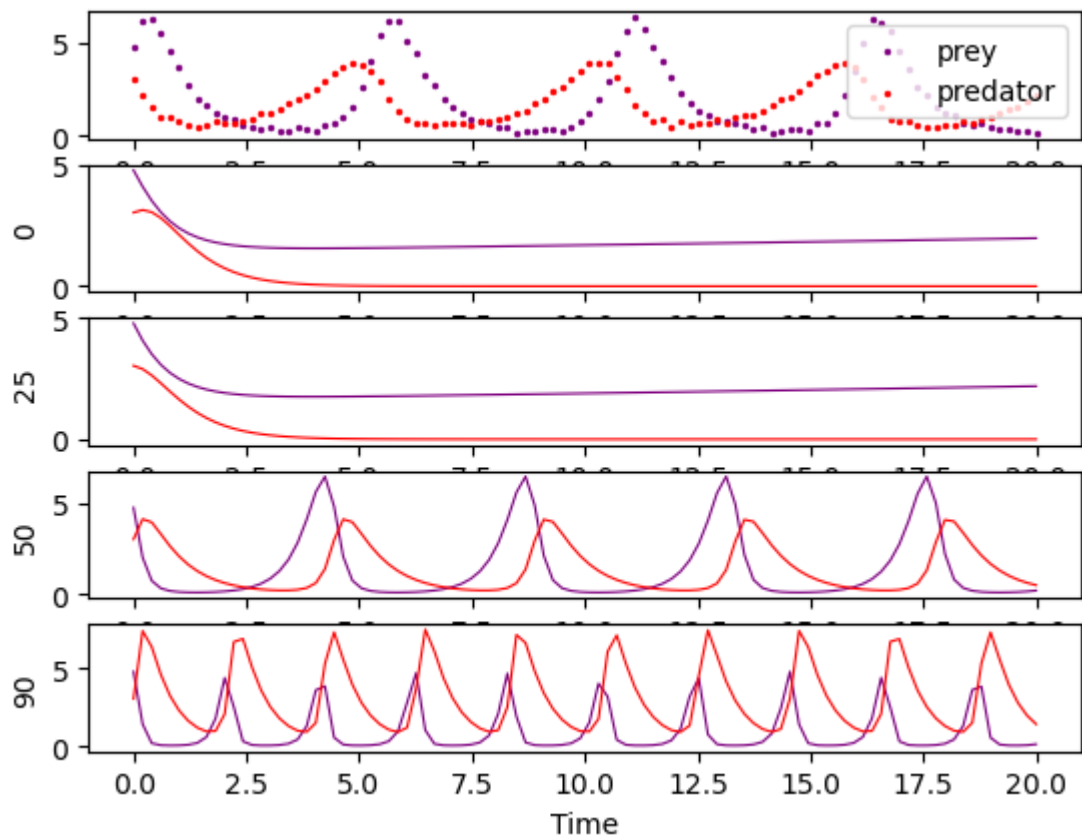
1 simulations = 2
2 iterations = 1
3 init_temp = 20
4 cooling = 0.1 # cooling strategy more aggressive
5 t = data[:,0]
6 time = t
7 inp = data[:,1:3]
8 cooling_constant = 0.10
9 parameters = [np.random.uniform(0,1), np.random.uniform(0,1), np.random.uniform(0,1)]
10
11 Result_full = inference_removal(initial_temp,cooling_constant, MSE2, simulations, iterations, parameters)
12 Result_quarter = inference_removal(initial_temp,cooling_constant, MSE2, simulations, iterations, parameters)
13 Result_half = inference_removal(initial_temp,cooling_constant, MSE2, simulations, iterations, parameters)
14 Result_90 = inference_removal(initial_temp,cooling_constant, MSE2, simulations, iterations, parameters)
15
16 Integration_full = predator_prey_integration(time,inp[0],Result_full[0])
17 Integration_quarter = predator_prey_integration(time,inp[0],Result_quarter[0])
18 Integration_half = predator_prey_integration(time,inp[0],Result_half[0])
19 Integration_90 = predator_prey_integration(time,inp[0],Result_90[0])
20
21 plt.plot(dpi = 300, figsize = (20, 5))
22 plt.subplot(511)
23 plt.scatter(time, inp.T[0], color = 'purple', s = 2, label = 'prey')
24 plt.scatter(time, inp.T[1], color = 'red', s = 2, label = 'predator')
25 plt.legend()
26 plt.subplot(512)
27 plt.plot(time, Integration_full[:,0], color = 'purple', label = 'prey',)
28 plt.plot(time, Integration_full[:,1], color = 'red', label = 'predator',)
29 plt.ylabel('0')
30 plt.subplot(513)
31 plt.plot(time, Integration_quarter[:,0], color = 'purple', label = 'prey',)
32 plt.plot(time, Integration_quarter[:,1], color = 'red', label = 'predator',)
33 plt.ylabel('25')
34 plt.subplot(514)
35 plt.plot(time, Integration_half[:,0], color = 'purple', label = 'prey',)
36 plt.plot(time, Integration_half[:,1], color = 'red', label = 'predator',)
37 plt.ylabel('50')
38 plt.subplot(515)
39 plt.plot(time, Integration_90[:,0], color = 'purple', label = 'prey',)
40 plt.plot(time, Integration_90[:,1], color = 'red', label = 'predator',)
41 plt.ylabel('90')
42 plt.xlabel('Time')
43 plt.show()
44 plt.close()
45

```

```

C:\Users\Aleks\AppData\Local\Temp\ipykernel_1540\1335953606.py:38: Runtime
Warning: overflow encountered in exp
    acceptance_probability = min(np.exp(-(delta/temp)),1)#Calculate acceptan
ce probability
C:\Users\Aleks\AppData\Local\Temp\ipykernel_1540\1335953606.py:38: Runtime
Warning: overflow encountered in scalar divide
    acceptance_probability = min(np.exp(-(delta/temp)),1)#Calculate acceptan
ce probability
C:\Users\Aleks\AppData\Local\Temp\ipykernel_1540\1335953606.py:38: Runtime
Warning: divide by zero encountered in scalar divide
    acceptance_probability = min(np.exp(-(delta/temp)),1)#Calculate acceptan
ce probability
C:\Users\Aleks\AppData\Local\anaconda3\Lib\site-packages\scipy\integrate\
odepack_py.py:248: ODEintWarning: Excess accuracy requested (tolerances to
o small). Run with full_output = 1 to get quantitative information.
    warnings.warn(warning_msg, ODEintWarning)
C:\Users\Aleks\AppData\Local\Temp\ipykernel_1540\631426890.py:22: Matplotlib
DeprecationWarning: Auto-removal of overlapping axes is deprecated since
3.6 and will be removed two minor releases later; explicitly call ax.remov
e() as needed.
    plt.subplot(511)

```



In []:

1


```
In [2]: 1 import numpy as np
        2 import matplotlib.pyplot as plt
        3 from scipy.stats import norm
        4 import pandas as pd
        5 from scipy.integrate import odeint
        6 import time
        7 import statistics
        8 import random
        9 import scipy.stats
       10
```

Opening predator-prey dataset

```
In [3]: 1 df = pd.read_csv('predator-prey-data.csv', index_col=False)
        2 df.head()
        3
```

```
Out[3]:
```

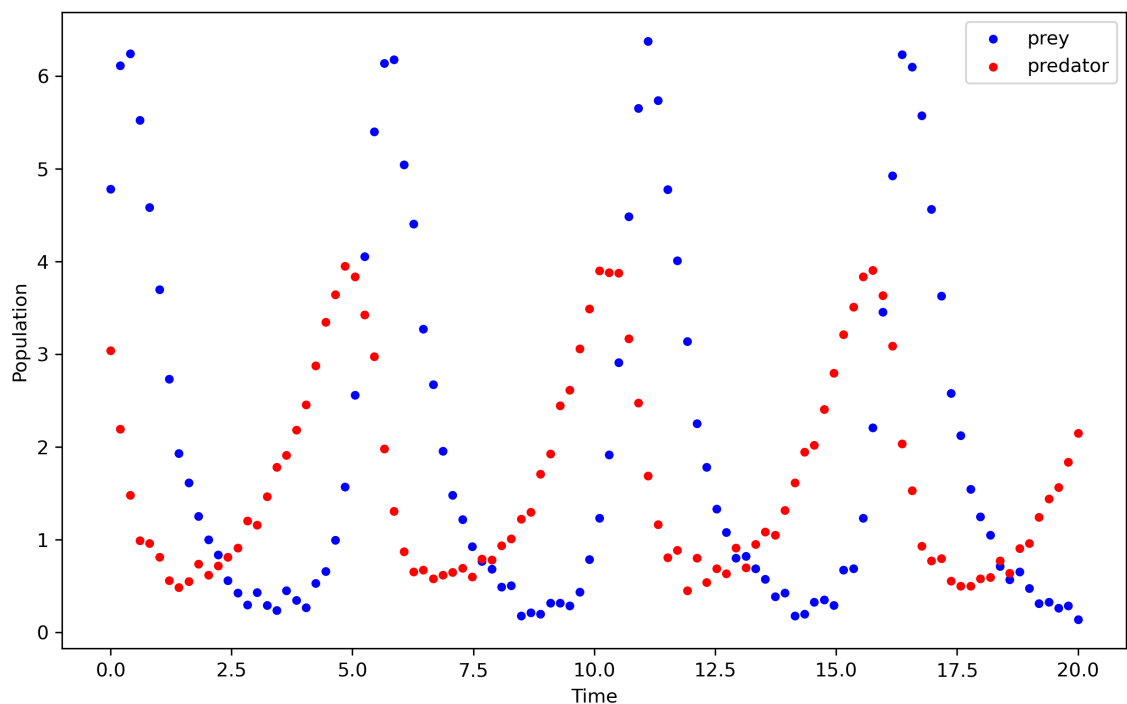
	Unnamed: 0	t	x	y
0	0	0.000000	4.781635	3.035257
1	1	0.202020	6.114005	2.189746
2	2	0.404040	6.238361	1.478907
3	3	0.606061	5.520524	0.989836
4	4	0.808081	4.582546	0.957827

```

In [4]: 1 # Loading data into read-only numpy arrays
2 data = df[['t', 'x', 'y']].values
3 # data[1], data[2] = data[2].copy(), data[1].copy()
4 data.flags.writeable = False
5
6
7 # Plotting
8 plt.figure(dpi = 300, figsize=(10, 6))
9 point_width = 13
10 # X should be prey
11 plt.scatter(data[:,0], data[:,1], label = 'prey', color = 'blue', s = pc
12 plt.scatter(data[:,0], data[:,2], label = 'predator', color = 'red', s=
13 plt.ylabel('Population')
14 plt.xlabel('Time')
15 plt.legend()
16

```

Out[4]: <matplotlib.legend.Legend at 0x14c151ad390>



Objective functions

Defining volterra equations function

```
In [5]: 1 def predator_prey_odes(initial_conditions,time ,alpha, beta, delta, gamma)
2       x = initial_conditions[0] # initial predator population
3       y = initial_conditions[1] # initial prey population
4       dxdt = (alpha * x) - (beta * x * y) # Predator ODE
5       dydt = (delta * x * y) - (gamma * y) # Predator ODE
6       return [dxdt, dydt]
7
8       #Function that will return the data for predator and prey for a given s
9       def predator_prey_integration(time,initial_conditions,parameters):
10          alpha,beta,delta,gamma = parameters
11          #odeint is now used as part of this function which returns the # of
12          results = odeint(predator_prey_odes,initial_conditions, time, args=
13          predator_values,prey_values = results[:,0], results[:,1]
14          return np.array([predator_values,prey_values]).T
15
```

```
In [ ]: 1
2
3
```

Defining objective functions

```
In [6]: 1 # modulo linear error
2 def MSE(actual, predicted):
3     '''Mean squared error'''
4     return np.mean((actual - predicted)**2)
5
6 def MSE2(actual, predicted):
7     '''Mean squared error, handles nan values'''
8     x1, y1 = actual[:, 0], actual[:, 1]
9
10    # Getting useful indexes
11    indx_x = np.where(~np.isnan(x1))
12    indx_y = np.where(~np.isnan(y1))
13    x2, y2 = predicted[:, 0], predicted[:, 1]
14
15    err1 = (x1[indx_x] - x2[indx_x])**2
16    err2 = (y1[indx_y] - y2[indx_y])**2
17
18    # Concatenate the arrays before calculating the mean
19    errors = np.concatenate([err1, err2])
20
21    # Use np.nanmean to handle NaN values during the mean calculation
22    return np.nanmean(errors)
23
24
25 def MAE(actual, predicted):
26     '''Calculate Mean Absolute Error (MAE) for multidimensional data.'''
27     mae = np.mean(np.abs(actual - predicted))
28     return mae
29
30 def MAE2(actual, predicted):
31     '''Calculate Mean Absolute Error (MAE) for multidimensional data, h
32     x1, y1 = actual[:, 0], actual[:, 1]
33
34     # Getting useful indexes
35     indx_x = np.where(~np.isnan(x1))
36     indx_y = np.where(~np.isnan(y1))
37
38     x2, y2 = predicted[:, 0], predicted[:, 1]
39
40     err1 = np.abs(x1[indx_x] - x2[indx_x])
41     err2 = np.abs(y1[indx_y] - y2[indx_y])
42
43     mae = np.nanmean(np.concatenate([err1, err2]))
44
45     return mae
46
47
```

Algorithms & Optimisation

Defining minimization algorithms

In [7]:

```

1  def random_walk(parameters, variance = 0.5):
2      lst = [parameter + np.random.normal(0, 1) for parameter in parameters]
3      # Ensure all elements are positive
4      while any(x <= 0 for x in lst):
5          for indx in range(len(lst)):
6              if lst[indx] <= 0:
7                  while lst[indx] < 0:
8                      lst[indx] = parameters[indx] + np.random.normal(0,
9
10     return lst
11
12
13 def hill_climbing(data, time, initial_conditions, parameters, objective)
14     '''Tries to find the best solution using random walker'''
15     # Initialize starting parameter state
16     scores = []
17     x_n = parameters
18     all_scores = []
19
20     current_est = predator_prey_integration(time, initial_conditions, x
21     current_score = objective(data, current_est)
22     scores.append(current_score)
23     number_iterations = 1
24
25     for k in range(max_iterations):
26         # Generate a random walk for parameters
27         x_n_1 = random_walk(x_n, variance)
28
29         # Calculate the current and next estimations
30         current_est = predator_prey_integration(time, initial_conditions, x
31         new_estimation = predator_prey_integration(time, initial_conditions, x
32
33         new_score = objective(data, new_estimation)
34
35         # If the next estimation is better, update the parameters
36         if new_score < current_score:
37             number_iterations = k
38             current_score = new_score
39             x_n = x_n_1
40             scores.append(current_score)
41
42     return x_n, scores, number_iterations

```

```

In [8]: 1 def simulated_annealing(initial_temp,cooling_constant, data, time, init
2
3     temp = initial_temp #Scaling factor for random movement. We square
4     start = parameters #Initial starting parameters
5     x_n = start
6     scores = [] #A score is just the value of the objective function ev
7
8     current_est = predator_prey_integration(time, initial_conditions, >
9     current_score = objective(data, current_est) #The current value of
10    scores.append(current_score) #Keeping track of the values of the ob
11
12    #cur = function(x) #The function value of the current x solution
13    history = [x_n] #Stores previously searched x values
14
15    for i in range (max_iterations):
16        proposal = random_walk(x_n) #A new proposal for the parameters
17        new_est = predator_prey_integration(time, initial_conditions, p
18        new_score = objective(data, new_est) #Calculate new value of ob
19
20        delta = new_score - current_score #Difference in objective func
21
22        #if proposal < 0 or proposal > 1:
23            #proposal = x_n # Reject proposal by setting it equal to pre
24
25        acceptance_probability = min(np.exp(-(delta/temp)),1)#Calculate
26
27        #if delta < 0:
28            #x_n = proposal ##Accept proposal
29            #current_score = new_score
30
31        if np.random.rand() < acceptance_probability: #else if it is no
32            x_n = proposal #Accept proposal
33            current_score = new_score
34
35        scores.append(current_score)
36        temp = cooling_constant**i * initial_temp #Cool temperature
37        #print(temp)
38        history.append(x_n) #Add to history
39
40    return x_n, scores
41
42
43

```

Multiple run algorithms

```

In [9]: 1 def uniform_draw_g(lower_bound, upper_bound):
2         while True:
3             yield np.random.uniform(lower_bound, upper_bound)
4
5 def multiple_runs_annealing(initial_temp, cooling_constant, input_data, t,
6
7     mse_total_list = []
8     all_all_best = []
9
10    for i in range(n_runs):
11
12        x_best, scores = simulated_annealing(initial_temp, cooling_const
13        all_all_best.append(x_best)
14
15        x = predator_prey_integration(t, initial_conditions, x_best)
16        mse_prey = MSE(data[:,1], x[:,0])
17        mse_predator = MSE(data[:,2], x[:,1])
18        mse_total = mse_prey + mse_predator
19
20        mse_total_list.append(mse_total) #Add total MSE for this simul
21
22    return np.array(all_all_best) , mse_total_list
23
24
25
26 def multi_run_hill_climbing(data, objective, nruns = 50, nsamples=100,
27     initial_conditions = data[0][1:3]
28     time = data[:,0]
29
30     # Defining generators for variables
31     alpha = uniform_draw_g(0,1)
32     beta = uniform_draw_g(0,1)
33     delta = uniform_draw_g(0,1)
34     gamma = uniform_draw_g(0,1)
35
36     # Lists for storing values
37     parameter_list = []
38     best = []
39     best_score = float('inf')
40     best_param = None
41     num_iterations = []
42
43     # Running simulation for
44     for __ in range(nruns):
45
46         parameters = [next(alpha), next(beta), next(delta), next(gamma)]
47         params, score, iterations = hill_climbing(data[:,1:3], time, ir
48         parameter_list.append(params)
49         num_iterations.append(iterations)
50         scores.append(score)
51
52         #Saving best parameter combination
53         if score[-1] < best_score:
54             best_score = score[-1]
55             best_param = params
56
57     parameter_list = np.array(parameter_list)
58
59     return parameter_list, best_param, scores, best_score, num_iteratic

```


60

Plotting hill climbing

In [10]:

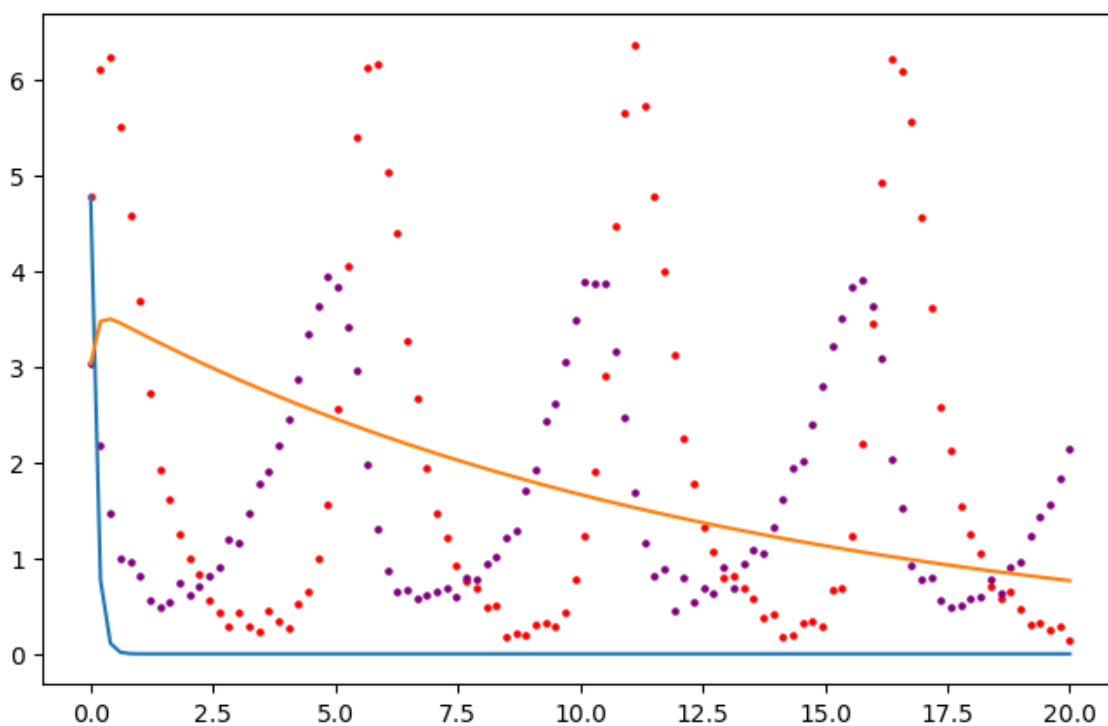
```
1 input_data = data[:,1:3]
2 # t =
3 initial_conditions = [input_data[0][0], input_data[0][1]]
4
5 alpha = np.random.uniform(0.5, 2)
6 beta = np.random.uniform(0.5, 2)
7 delta = np.random.uniform(0.5, 2)
8 gamma = np.random.uniform(0.5, 2)
9 parameters = [alpha, beta, delta, gamma]
10
11 # Using MSeX
12 x_best, scores, num_iterations = hill_climbing(input_data, data[:,0], i
13 print(x_best,scores,num_iterations)
```

```
[2.058470826384955, 3.3348226913782497, 0.33391386324627437, 0.07775331273
010042] [1.7841476511911036, 1.7630633552849264, 1.7463166480090968, 1.741
6628363579707, 1.7024628624472102, 1.6549539797197672, 1.541264871748595]
16
```

Plotting hill climbing results

```
In [11]: 1 # t, x ,y = data
2 initial_conditions = [input_data[0][0], input_data[0][1]]
3 t = data[:,0]
4
5 parameters = x_best
6 # Using MSE
7 x = predator_prey_integration(t,initial_conditions,parameters)
8
9 # Increase the figure size
10 plt.figure(figsize=(8, 5))
11
12 plt.plot(t, x[:,0])
13 plt.plot(t, x[:,1])
14
15 plt.scatter(t, data[:,1], color= 'red', s =5)
16 plt.scatter(t, data[:,2], color= 'purple', s=5)
17
18 plt.figure(figsize=(10, 8))
```

Out[11]: <Figure size 1000x800 with 0 Axes>



<Figure size 1000x800 with 0 Axes>

Running simulation for different random walker variance

```
In [ ]: 1 # parameter_list = np.array(parameter_list)
2 # # Create a figure with 3x3 subplotshttp://localhost:8888/notebooks/De
3 # fig, axes = plt.subplots(3, 3, figsize=(12, 12), sharex=True)
4
5 # # Plot histograms on each subplot using for loops with the same color
6 # color = 'blue'
7 # titles = ['variance = 0.1', 'variance=0.25', 'variance=0.5']
8 # x_titles = ['alpha', 'beta', 'delta', 'omega']
9
10 # for i in range(3):
11 #     for j in range(3):
12 #         ax = axes[i, j]
13 #         ax.hist(parameter_list[j][:,i])
14 #         ax.set_xlabel(x_titles[i])
15 #         ax.set_ylabel('Frequency')
16 #         ax.set_title(titles[j])
17
18 # # Adjust layout to prevent overlapping
19 # plt.tight_layout()
20
21 # # Show the plot
22 # plt.show()
23
```

Running multi run for hill climbing

In [12]:

```
1
2 # We save the parameter estimation we will use as ground truth for test
3 parameter_list, reference_param, scores, reference_score, num_iteration
4
5 # Integrating with best guess
6 results = predator_prey_integration(t, initial_conditions, reference_param
7
8 # Increase the figure size
9 plt.figure(figsize=(8, 5))
10
11 plt.plot(data[:,0], results[:,0])
12 plt.plot(data[:,0], results[:,1])
13
14 plt.scatter(t, data[:,1], color= 'red', s =5)
15 plt.scatter(t, data[:,2], color= 'purple', s=5)
16
17 plt.show()
18
```

C:\Users\Aleks\AppData\Local\anaconda3\Lib\site-packages\scipy\integrate\odepack_py.py:248: ODEintWarning: Excess work done on this call (perhaps wrong Dfun type). Run with full_output = 1 to get quantitative information.
warnings.warn(warning_msg, ODEintWarning)

-
KeyboardInterrupt

Traceback (most recent call last)

t)

Cell In[12], line 2

```
1 # We save the parameter estimation we will use as ground truth for
testing
----> 2 parameter_list, reference_param, scores, reference_score, num_iter
ations = multi_run_hill_climbing(data, MSE, nruns = 200)
4 # Integrating with best guess
5 results = predator_prey_integration(t, initial_conditions, reference
_param)
```

Cell In[9], line 47, in multi_run_hill_climbing(data, objective, nruns, nsamples, variance)

```
44 for __ in range(nruns):
46     parameters = [next(alpha), next(beta), next(delta), next(gamma)]
---> 47     params, score, iterations = hill_climbing(data[:,1:3], time, i
nitial_conditions, parameters, objective, max_iterations=nsamples, variance
=variance)
48     parameter_list.append(params)
49     num_iterations.append(iterations)
```

Cell In[7], line 30, in hill_climbing(data, time, initial_conditions, parameters, objective, max_iterations, variance)

```
28 # Calculate the current and next estimations
29 current_est = predator_prey_integration(time, initial_conditions,
x_n)
---> 30 new_estimation = predator_prey_integration(time, initial_conditions,
s, x_n_1)
32 new_score = objective(data, new_estimation)
34 # If the next estimation is better, update the parameters
```

Cell In[5], line 12, in predator_prey_integration(time, initial_conditions, parameters)

```
10 alpha, beta, delta, gamma = parameters
11 #odeint is now used as part of this function which returns the # o
f infected in the model
---> 12 results = odeint(predator_prey_odes, initial_conditions, time, args
=(alpha, beta, delta, gamma))
13 predator_values, prey_values = results[:,0], results[:,1]
14 return np.array([predator_values, prey_values]).T
```

File ~\AppData\Local\anaconda3\Lib\site-packages\scipy\integrate\odepack_py.py:242, in odeint(func, y0, t, args, Dfun, col_deriv, full_output, ml, mu, rtol, atol, tcrit, h0, hmax, hmin, ixpr, mxstep, mxhnil, mxordn, mxords, printmessg, tfirst)

```
240 t = copy(t)
241 y0 = copy(y0)
--> 242 output = _odepack.odeint(func, y0, t, args, Dfun, col_deriv, ml, m
u,
243                               full_output, rtol, atol, tcrit, h0, hmax,
hmin,
244                               ixpr, mxstep, mxhnil, mxordn, mxords,
245                               int(bool(tfirst)))
246 if output[-1] < 0:
247     warning_msg = _msgs[output[-1]] + " Run with full_output = 1 t
o get quantitative information."
```

KeyboardInterrupt:

Kaya's code section: Points removal

```
In [13]: 1 def point_removal(time, input_data, points_removed, Focus = 'both'):
2         '''removes points randomly'''
3
4         #We set the seed for removing points
5         random.seed(123)
6
7         prey = input_data.T[0].copy()
8         predator = input_data.T[1].copy()
9
10        # initialize set up for removing points randomly given the bounds
11        removal_options = np.arange(0, len(time))
12
13        # choose points to be removed randomly
14        if points_removed > len(removal_options):
15            points_removed = len(removal_options)
16            print('WARNING: Maximum number of points that can be removed has been reached')
17        removed_points_indices = random.choices(removal_options, k = points_removed)
18
19        # remove points based on choices for points to be removed
20        if Focus == 'both':
21            for i in removed_points_indices:
22                prey[i] = None
23                predator[i] = None
24
25        elif Focus == 'prey':
26            for i in removed_points_indices:
27                prey[i] = None
28
29        elif Focus == 'predator':
30            for i in removed_points_indices:
31                predator[i] = None
32
33        return np.array([time, prey, predator]).T
34
35
```



```

In [14]: 1 def extrema_removal(time, input_data, points_removed, Focus = 'both'):
2         '''Removes points in extrema'''
3         prey = input_data.T[0].copy()
4         predator = input_data.T[1].copy()
5
6         # Calculate mean and variance to set regions for data
7         mean_prey_population, mean_predator_population = np.mean(prey), np.
8         variance_prey, variance_predator = statistics.variance(prey), stati
9
10        # set upper bound and lower bound for point removals
11        ub_prey, lb_prey = mean_prey_population + 1.645*variance_prey/len(t
12        ub_predator, lb_predator = mean_predator_population + 1.645*varianc
13
14        # initialize set up for removing points randomly given the bounds
15        prey_options = []
16        predator_options = []
17        # enumerate through list of stored points
18        for index, prey_count in enumerate(prey):
19            # check if they are in specified region
20            if prey_count > ub_prey or prey_count < lb_prey:
21                prey_options.append([index, prey_count, predator[index]])
22        for index, predator_count in enumerate(predator):
23            if predator_count > ub_predator or predator_count < lb_predator
24                predator_options.append([index, prey[index], predator_count
25
26        # remove points from list depending on which focus is set
27        removal_options = []
28        if Focus == 'both':
29            removal_options = removal_options + prey_options + predator_opt
30        elif Focus == 'prey':
31            removal_options = removal_options + prey_options
32        elif Focus == 'predator':
33            removal_options = removal_options + predator_options
34        else:
35            print('Error: Removal option not known. Try either both, prey,
36
37        # choose points to be removed randomly
38        if points_removed > len(removal_options):
39            points_removed = len(removal_options)
40            print('WARNING: Maximum number of points that can be removed ha
41        removed_points_indices = random.choices(np.array(removal_options).T
42
43        # turn the list into integers so we can remove them based on the in
44        integer_array = []
45        for counter in range(len(removed_points_indices)):
46            integer_array.append(int(removed_points_indices[counter]))
47
48        # update the lists based on points we wanted to remove
49        if Focus == 'both':
50            for i in integer_array:
51                prey[i] = None
52                predator[i] = None
53
54        elif Focus == 'prey':
55            for i in integer_array:
56                prey[i] = None
57
58        elif Focus == 'predator':
59            for i in integer_array:
60                predator[i] = None
61

```



```
62     return np.array(time), np.array(preyn), np.array(predatorn)
63
64 # extrema_removal(t, input_data, 5, Focus = 'both')
```



```

In [15]: 1 def midpoint_removal(time, input_data, points_removed, Focus = 'both'):
2         '''Removes points close to the mean'''
3         prey = input_data.T[0].copy()
4         predator = input_data.T[1].copy()
5
6         # Calculate mean and variance to set regions for data
7         mean_prey_population, mean_predator_population = np.mean(prey), np.
8         variance_prey, variance_predator = statistics.variance(prey), stati
9
10        # set upper bound and lower bound for point removals
11        ub_prey, lb_prey = mean_prey_population + 1.645*variance_prey/len(t
12        ub_predator, lb_predator = mean_predator_population + 1.645*variance
13
14        # initialize set up for removing points randomly given the bounds
15        prey_options = []
16        predator_options = []
17        # enumerate through list of stored points
18        for index, prey_count in enumerate(prey):
19            # check if they are in specified region
20            if prey_count <= ub_prey or prey_count >= lb_prey:
21                prey_options.append([index, prey_count, predator[index]])
22        for index, predator_count in enumerate(predator):
23            if predator_count <= ub_predator or predator_count >= lb_predat
24                predator_options.append([index, prey[index], predator_count
25
26        # remove points from list depending on which focus is set
27        removal_options = []
28        if Focus == 'both':
29            removal_options = removal_options + prey_options + predator_opt
30        elif Focus == 'prey':
31            removal_options = removal_options + prey_options
32        elif Focus == 'predator':
33            removal_options = removal_options + predator_options
34        else:
35            print('Error: Removal option not known. Try either both, prey,
36
37        # choose points to be removed randomly
38        if points_removed > len(removal_options):
39            points_removed = len(removal_options)
40            print('WARNING: Maximum number of points that can be removed ha
41        removed_points_indices = random.choices(np.array(removal_options).T
42
43        # turn the list into integers so we can remove them based on the in
44        integer_array = []
45        for counter in range(len(removed_points_indices)):
46            integer_array.append(int(removed_points_indices[counter]))
47
48        # update the lists based on points we wanted to remove
49        if Focus == 'both':
50            for i in integer_array:
51                prey[i] = None
52                predator[i] = None
53
54        elif Focus == 'prey':
55            for i in integer_array:
56                prey[i] = None
57
58        elif Focus == 'predator':
59            for i in integer_array:
60                predator[i] = None
61

```

```
62     return np.array([time, prey, predator]).T
63
```

In []:

```
1  # 1. Run multi run for different size datasets save best parameters
2  # 2. Calculate MSE for each run for best parameters
3  # 2. Do this for 2x, one only for predator, other for prey
4  # 4. Plot error relative to best solution of y axis
5  # 5. On x axis should be relative number points
6
```

Hypothesis testing random removal points (Aleks section)

Duplicating code for the functions I use in case they are different

```

In [ ]: 1 def random_walk(parameters, variance = 0.5):
2         lst = [parameter + np.random.normal(0, 1) for parameter in parameters]
3         # Ensure all elements are positive
4         while any(x <= 0 for x in lst):
5             for indx in range(len(lst)):
6                 if lst[indx] <= 0:
7                     while lst[indx] < 0:
8                         lst[indx] = parameters[indx] + np.random.normal(0,
9                     variance)
10        return lst
11
12 def hill_climbing(data, time, initial_conditions, parameters, objective):
13     '''Tries to find the best solution using random walker'''
14     # Initialize starting parameter state
15     scores = []
16     x_n = parameters
17     all_scores = []
18
19     current_est = predator_prey_integration(time, initial_conditions, parameters)
20     current_score = objective(data, current_est)
21     scores.append(current_score)
22     number_iterations = 1
23
24     for k in range(max_iterations):
25         # Generate a random walk for parameters
26         x_n_1 = random_walk(x_n, variance)
27
28         # Calculate the current and next estimations
29         current_est = predator_prey_integration(time, initial_conditions, x_n_1)
30         new_estimation = predator_prey_integration(time, initial_conditions, x_n_1)
31
32         new_score = objective(data, new_estimation)
33
34         # If the next estimation is better, update the parameters
35         if new_score < current_score:
36             number_iterations = k
37             current_score = new_score
38             x_n = x_n_1
39             scores.append(current_score)
40
41     return x_n, scores, number_iterations
42
43 def simulated_annealing(initial_temp, cooling_constant, data, time, initial_conditions, parameters, objective):
44     temp = initial_temp #Scaling factor for random movement. We square it
45     start = parameters #Initial starting parameters
46     x_n = start
47     scores = [] #A score is just the value of the objective function evaluated at the current parameters
48
49     current_est = predator_prey_integration(time, initial_conditions, x_n)
50     current_score = objective(data, current_est) #The current value of the objective function
51     scores.append(current_score) #Keeping track of the values of the objective function
52
53     #cur = function(x) #The function value of the current x solution
54     history = [x_n] #Stores previously searched x values
55
56     for i in range(max_iterations):
57         proposal = random_walk(x_n) #A new proposal for the parameters
58         new_est = predator_prey_integration(time, initial_conditions, proposal)
59         new_score = objective(data, new_est) #Calculate new value of the objective function
60
61         #Acceptance probability
62         prob = min(1, new_score / current_score)
63         rand = np.random.random()
64         if rand < prob:
65             x_n = proposal
66             current_est = new_est
67             current_score = new_score
68             scores.append(new_score)
69
70     return x_n, scores, number_iterations

```

```

62     delta = new_score - current_score #Difference in objective function
63
64     #if proposal < 0 or proposal > 1:
65         #proposal = x_n # Reject proposal by setting it equal to previous
66
67     acceptance_probability = min(np.exp(-(delta/temp)),1)#Calculate acceptance probability
68
69     #if delta < 0:
70         #x_n = proposal ##Accept proposal
71         #current_score = new_score
72
73     if np.random.rand() < acceptance_probability: #else if it is not accepted
74         x_n = proposal #Accept proposal
75         current_score = new_score
76
77     scores.append(current_score)
78     temp = cooling_constant**i * initial_temp #Cool temperature
79     #print(temp)
80     history.append(x_n) #Add to history
81
82     return x_n, scores
83
84
85 def uniform_draw_g(lower_bound, upper_bound):
86     while True:
87         yield np.random.uniform(lower_bound, upper_bound)
88
89 def multiple_runs_annealing(initial_temp,cooling_constant,input_data,t_max):
90
91     mse_total_list = []
92     all_all_best = []
93
94     for i in range(n_runs):
95
96         x_best, scores = simulated_annealing(initial_temp,cooling_constant,input_data,t_max)
97         all_all_best.append(x_best)
98
99         x = predator_prey_integration(t,initial_conditions,x_best)
100         mse_prey = MSE(data[:,1],x[:,0])
101         mse_predator = MSE(data[:,2],x[:,1])
102         mse_total = mse_prey + mse_predator
103
104         mse_total_list.append(mse_total) #Add total MSE for this simulation
105
106     return np.array(all_all_best) , mse_total_list
107
108
109
110 def multi_run_hill_climbing(data, objective, nruns = 50, nsamples=100,
111     initial_conditions = data[0][1:3]
112     time = data[:,0]
113
114     # Defining generators for variables
115     alpha = uniform_draw_g(0,1)
116     beta = uniform_draw_g(0,1)
117     delta = uniform_draw_g(0,1)
118     gamma = uniform_draw_g(0,1)
119
120     # Lists for storing values
121     parameter_list = []
122     best = []

```

```

123 best_score = float('inf')
124 best_param = None
125 num_iterations = []
126
127 # Running simulation for
128 for __ in range(nruns):
129
130     parameters = [next(alpha), next(beta), next(delta), next(gamma
131     params, score, iterations = hill_climbing(data[:,1:3], time, i
132     parameter_list.append(params)
133     num_iterations.append(iterations)
134     scores.append(score)
135
136     #Saving best parameter combination
137     if score[-1] < best_score:
138         best_score = score[-1]
139         best_param = params
140
141     parameter_list = np.array(parameter_list)
142
143     return parameter_list, best_param, scores, best_score, num_iterati
144
145
146 def point_removal(time, input_data, points_removed, Focus = 'both'):
147     '''removes points randomly'''
148
149     #We set the seed for removing points
150     random.seed(123)
151
152     prey = input_data.T[0].copy()
153     predator = input_data.T[1].copy()
154
155     # initialize set up for removing points randomly given the bounds
156     removal_options = np.arange(0,len(time))
157
158     # choose points to be removed randomly
159     if points_removed > len(removal_options):
160         points_removed = len(removal_options)
161         print('WARNING: Maximum number of points that can be removed ha
162     removed_points_indices = random.choices(removal_options, k = point
163
164     # remove points based on choices for points to be removed
165     if Focus == 'both':
166         for i in removed_points_indices:
167             prey[i] = None
168             predator[i] = None
169
170     elif Focus == 'prey':
171         for i in removed_points_indices:
172             prey[i] = None
173
174     elif Focus == 'predator':
175         for i in removed_points_indices:
176             predator[i] = None
177
178     return np.array([time, prey, predator]).T

```


Getting distribution of averages of best guesses for hill climbing (reference dataset)

```
In [22]: 1 # We get the reference distribution for testing
2
3 # Timing your code
4 start_time = time.time()
5
6 # Reference distribution of averages
7 ref_average1 = []
8 for k in range(50):
9     parameter_list, best_param, scores, best_score, num_iterations = mu
10     # Appending average
11     ref_average1.append(np.mean(parameter_list, axis=0))
12
13 ref_average1 = np.array(ref_average1)
14
15 end_time = time.time()
16
17 # Calculating and printing the total time
18 total_time = end_time - start_time
19 print(f"Total time taken: {total_time} seconds")
20
21 # param_distribution, reference_param, scores, reference_score, num_ite
```

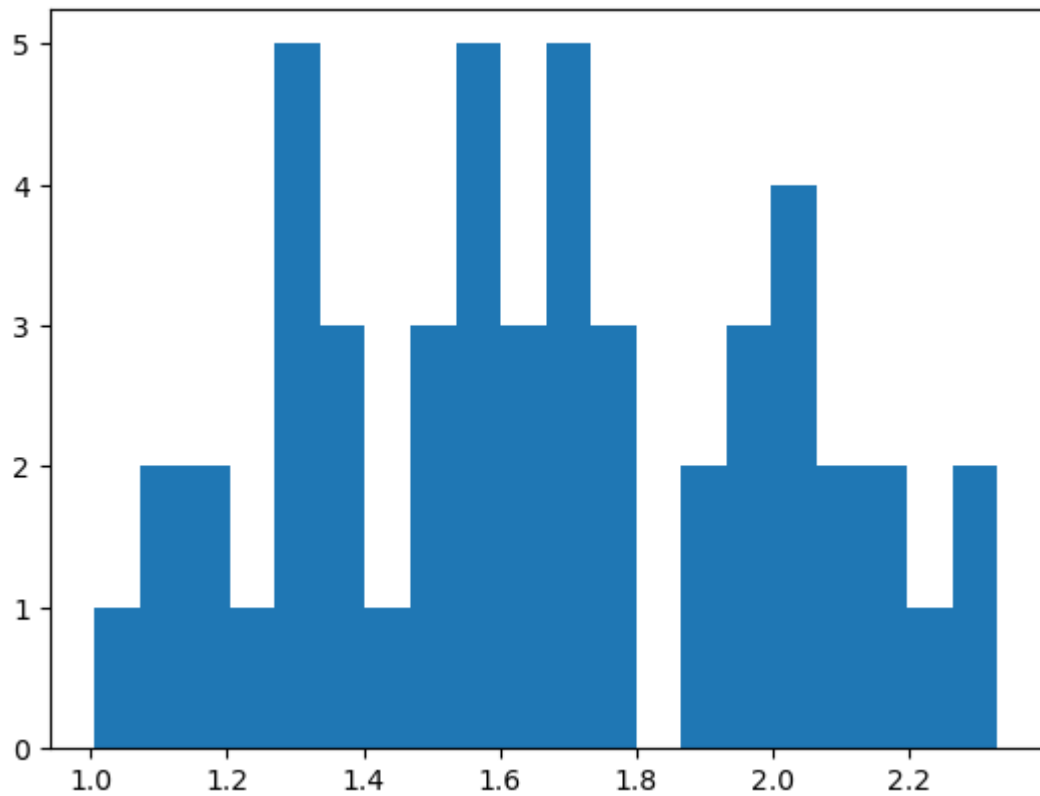
Total time taken: 439.93707609176636 seconds

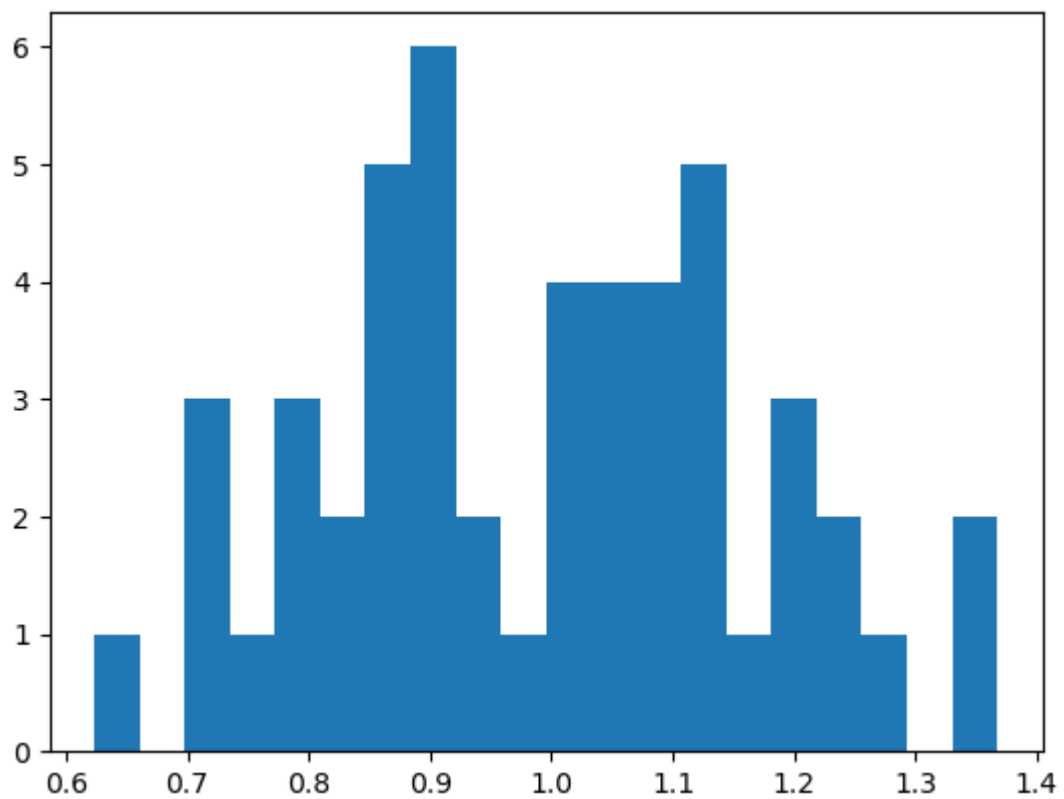
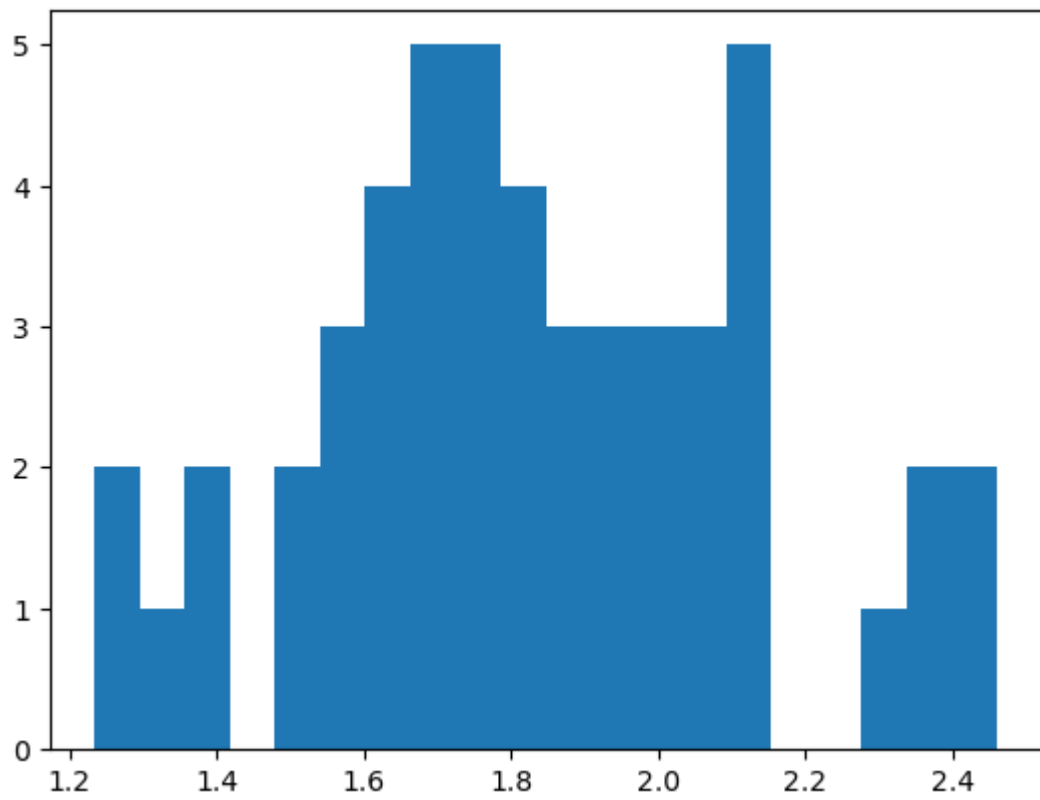
```
In [ ]: 1
```

These histograms plots are optional (I dont think im adding them to the report)

In [24]:

```
1 # num_iterations
2 # print(ref_average)
3 # for k in range(3):
4 #     plt.hist(ref_average1[:,k], bins = 20)
5 #     plt.show()
6
7 # print(np.var(ref_average1,axis=0))
```





[0.1170307 0.08449044 0.02953349 0.11417836]

```
In [25]: 1 # # print(param_distribution)
2 # plt.axvline(x=reference_param[0], color='r', linestyle='--', label='V
3 # plt.hist(param_distribution[:,0], bins =30)
```

```
-----
NameError                                Traceback (most recent call las
t)
Cell In[25], line 2
      1 # print(param_distribution)
----> 2 plt.axvline(x=reference_param[0], color='r', linestyle='--', label
      = 'Vertical Line at x=2.5')
      3 plt.hist(param_distribution[:,0], bins =30)

NameError: name 'reference_param' is not defined
```

Running welch test between reference distribution of averages and incomplete time series (hill climbing)

```
In [27]: 1 def random_point_ttests(data,ref_distribution, points_removed, focus_ch
2         '''runs welch test for multi run of hill climbing for every paramet
3         param_distribution = ref_distribution
4         # focus_choices = ['prey', 'predator', 'both']
5         # focus_choices = ['prey', 'predator', 'both']
6         scores = [[],[]]
7         p_values = {'prey': [], 'predator': [], 'both': []}
8
9         for indx, choice in enumerate(focus_choices):
10            print(choice)
11            for npoints in points_removed:
12                print(npoints)
13                # print(f"Points removed: {npoints}")
14                limited_data = point_removal(data[:,0], data[:,1:3], npoint
15                #Getting distribution of averages
16                average_distribution = []
17                for k in range(30):
18                    parameter_list, best_param, scores, best_score, num_ite
19                    #Appending average
20                    average_distribution.append(np.mean(parameter_list, axi
21
22                average_distribution = np.array(average_distribution)
23                t_stat1, p_value1 = scipy.stats.ttest_ind(ref_distribution[
24                t_stat2, p_value2 = scipy.stats.ttest_ind(ref_distribution[
25                t_stat3, p_value3 = scipy.stats.ttest_ind(ref_distribution[
26                t_stat3, p_value4 = scipy.stats.ttest_ind(ref_distribution[
27                p_values[choice].append([p_value1, p_value2, p_value3, p_va
28
29            return p_values
30
31
```

```
In [28]: 1 start_time = time.time()
2 p_values_hill_climbing = random_point_ttests(data, ref_average1, np.arange(
3
4 end_time = time.time()
5
6 # Calculating and printing the total time
7 total_time = end_time - start_time
8 print(f"Total time taken: {total_time} seconds")
```

prey

3
6
9
12
15

C:\Users\Aleks\AppData\Local\anaconda3\Lib\site-packages\scipy\integrate\odepack_py.py:248: ODEintWarning: Excess accuracy requested (tolerances too small). Run with full_output = 1 to get quantitative information.
warnings.warn(warning_msg, ODEintWarning)

predator

3
6
9
12
15

C:\Users\Aleks\AppData\Local\anaconda3\Lib\site-packages\scipy\integrate\odepack_py.py:248: ODEintWarning: Illegal input detected (internal error). Run with full_output = 1 to get quantitative information.
warnings.warn(warning_msg, ODEintWarning)
C:\Users\Aleks\AppData\Local\anaconda3\Lib\site-packages\scipy\integrate\odepack_py.py:248: ODEintWarning: Run terminated (internal error). Run with full_output = 1 to get quantitative information.
warnings.warn(warning_msg, ODEintWarning)

both

3
6
9
12
15

Total time taken: 1060.7844800949097 seconds

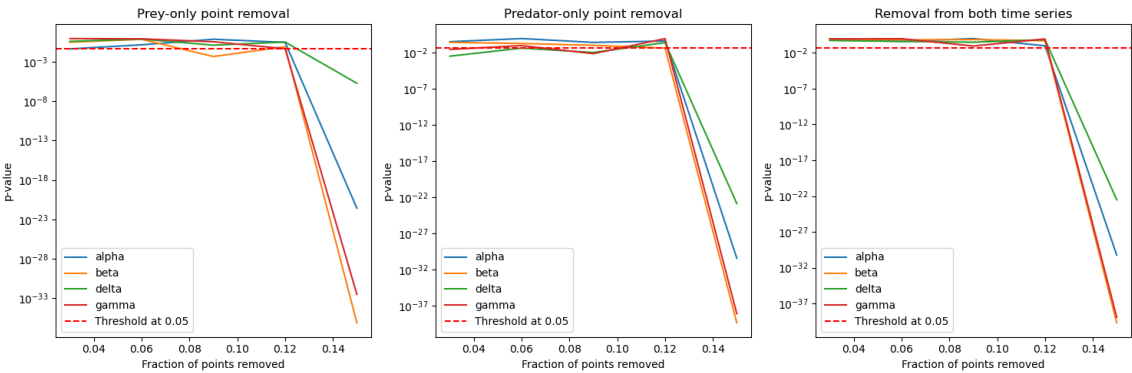
Visualizing p-values from welch test for hill climbing

In [43]:

```

1  # Fraction points removed
2  fraction_points = np.arange(3,16,3) / 100
3  p_values_pre = np.array(p_values_hill_climbing['prey'])
4  p_values_predator = np.array(p_values_hill_climbing['predator'])
5  p_values_both = np.array(p_values_hill_climbing['both'])
6
7
8  # Creating subplots
9  fig, axes = plt.subplots(1, 3, figsize=(15, 5))
10
11 # Plotting p-values when only prey points are removed
12 axes[0].plot(fraction_points, p_values_pre[:, 0], label='alpha')
13 axes[0].plot(fraction_points, p_values_pre[:, 1], label='beta')
14 axes[0].plot(fraction_points, p_values_pre[:, 2], label='delta')
15 axes[0].plot(fraction_points, p_values_pre[:, 3], label='gamma')
16 axes[0].axhline(y=0.05, color='r', linestyle='--', label='Threshold at
17 axes[0].set_ylabel('p-value')
18 axes[0].set_xlabel('Fraction of points removed')
19 axes[0].set_yscale('log')
20 axes[0].set_title('Prey-only point removal') # Add title to the first
21 axes[0].legend()
22
23 # Plotting p-values when only predator points are removed
24 axes[1].plot(fraction_points, p_values_predator[:, 0], label='alpha')
25 axes[1].plot(fraction_points, p_values_predator[:, 1], label='beta')
26 axes[1].plot(fraction_points, p_values_predator[:, 2], label='delta')
27 axes[1].plot(fraction_points, p_values_predator[:, 3], label='gamma')
28 axes[1].axhline(y=0.05, color='r', linestyle='--', label='Threshold at
29 axes[1].set_ylabel('p-value')
30 axes[1].set_xlabel('Fraction of points removed')
31 axes[1].set_yscale('log')
32 axes[1].set_title('Predator-only point removal')
33 axes[1].legend()
34
35 # Plotting p-values when both prey and predator points are removed
36 axes[2].plot(fraction_points, p_values_both[:, 0], label='alpha')
37 axes[2].plot(fraction_points, p_values_both[:, 1], label='beta')
38 axes[2].plot(fraction_points, p_values_both[:, 2], label='delta')
39 axes[2].plot(fraction_points, p_values_both[:, 3], label='gamma')
40 axes[2].axhline(y=0.05, color='r', linestyle='--', label='Threshold at
41 axes[2].set_ylabel('p-value')
42 axes[2].set_xlabel('Fraction of points removed')
43 axes[2].set_yscale('log')
44 axes[2].set_title('Removal from both time series')
45 axes[2].legend()
46
47 # Adjusting layout
48 plt.tight_layout()
49 plt.savefig('welch_tests_hill_climbing', dpi = 300)
50 plt.show()
51

```



Getting distribution of averages of best guesses for simulated annealing (reference dataset)

In [40]:

```

1  # We get the reference distribution for testing
2
3  # Timing your code
4  start_time = time.time()
5
6  initial_temp = 20
7  cooling_constant = 0.10
8
9  #Taking random draw for initial parameters (initial guess)
10 alpha = np.random.uniform(0,1)
11 beta = np.random.uniform(0,1)
12 delta = np.random.uniform(0,1)
13 gamma = np.random.uniform(0,1)
14 parameters = [alpha, beta, delta, gamma]
15 parameters = [alpha, beta, delta, gamma]
16
17 # Reference distribution of averages
18 ref_average2 = []
19 for k in range(50):
20     # parameter_list, best_param, scores, best_score, num_iterations =
21     parameter_list, scores = multiple_runs_annealing(initial_temp, cooling_constant, parameters)
22     # Appending average
23     ref_average2.append(np.mean(parameter_list, axis=0))
24
25 ref_average2 = np.array(ref_average2)
26 end_time = time.time()
27
28 # Calculating and printing the total time
29 total_time = end_time - start_time
30 print(f"Total time taken: {total_time} seconds")

```

C:\Users\Aleks\AppData\Local\Temp\ipykernel_764\951072704.py:25: RuntimeWarning: overflow encountered in exp

acceptance_probability = min(np.exp(-(delta/temp)),1)#Calculate acceptance probability

C:\Users\Aleks\AppData\Local\Temp\ipykernel_764\951072704.py:25: RuntimeWarning: overflow encountered in scalar divide

acceptance_probability = min(np.exp(-(delta/temp)),1)#Calculate acceptance probability

C:\Users\Aleks\AppData\Local\Temp\ipykernel_764\951072704.py:25: RuntimeWarning: divide by zero encountered in scalar divide

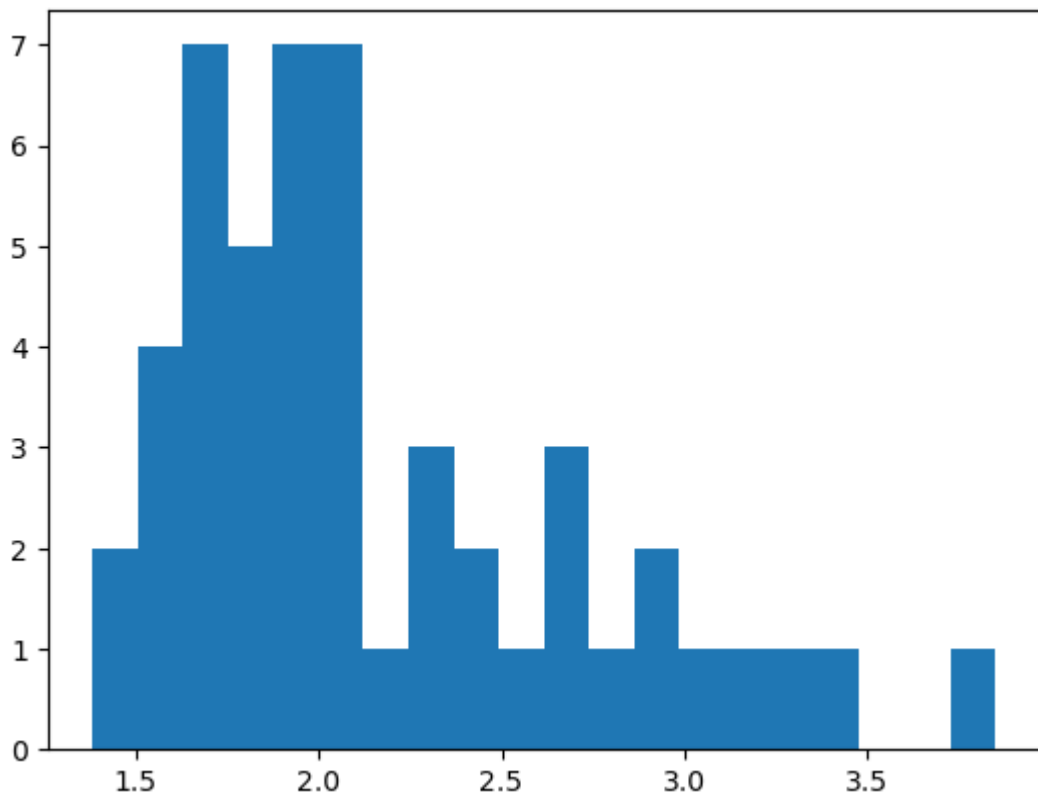
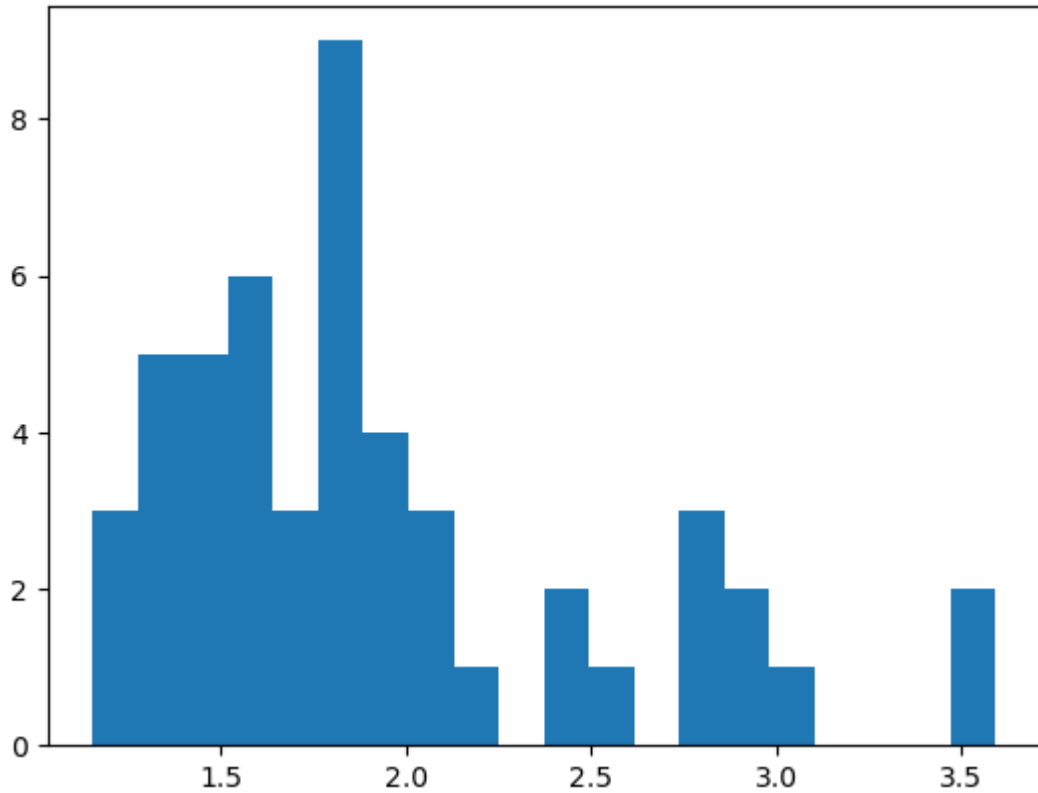
acceptance_probability = min(np.exp(-(delta/temp)),1)#Calculate acceptance probability

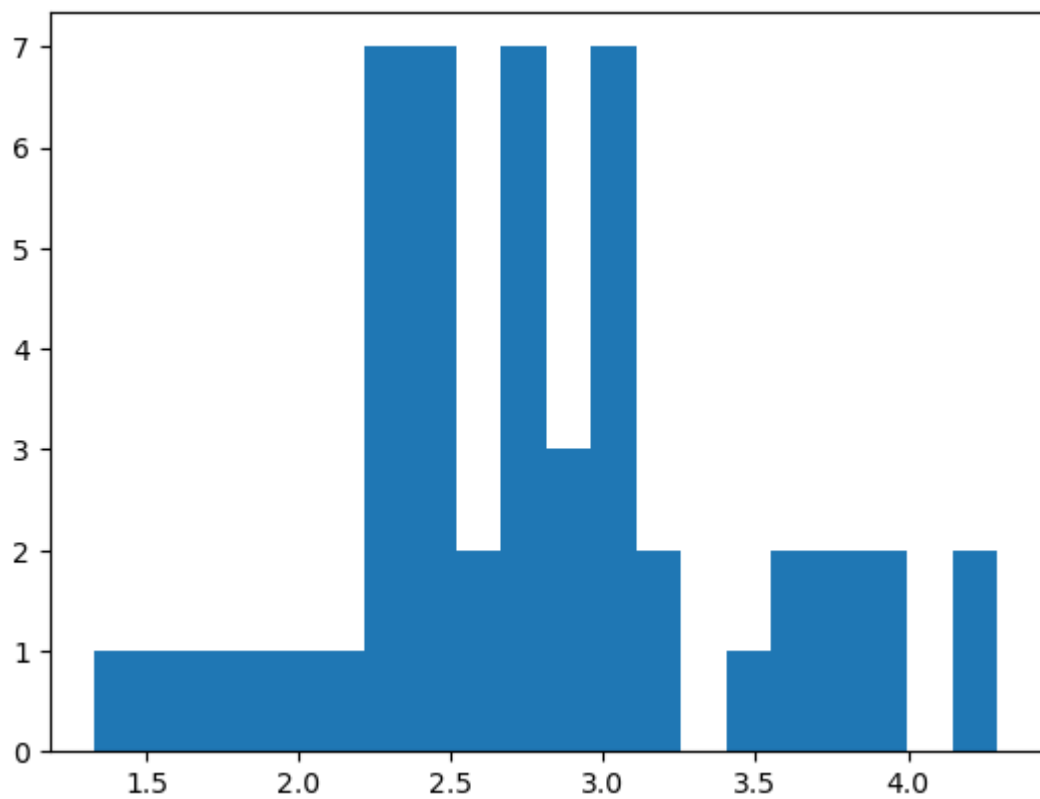
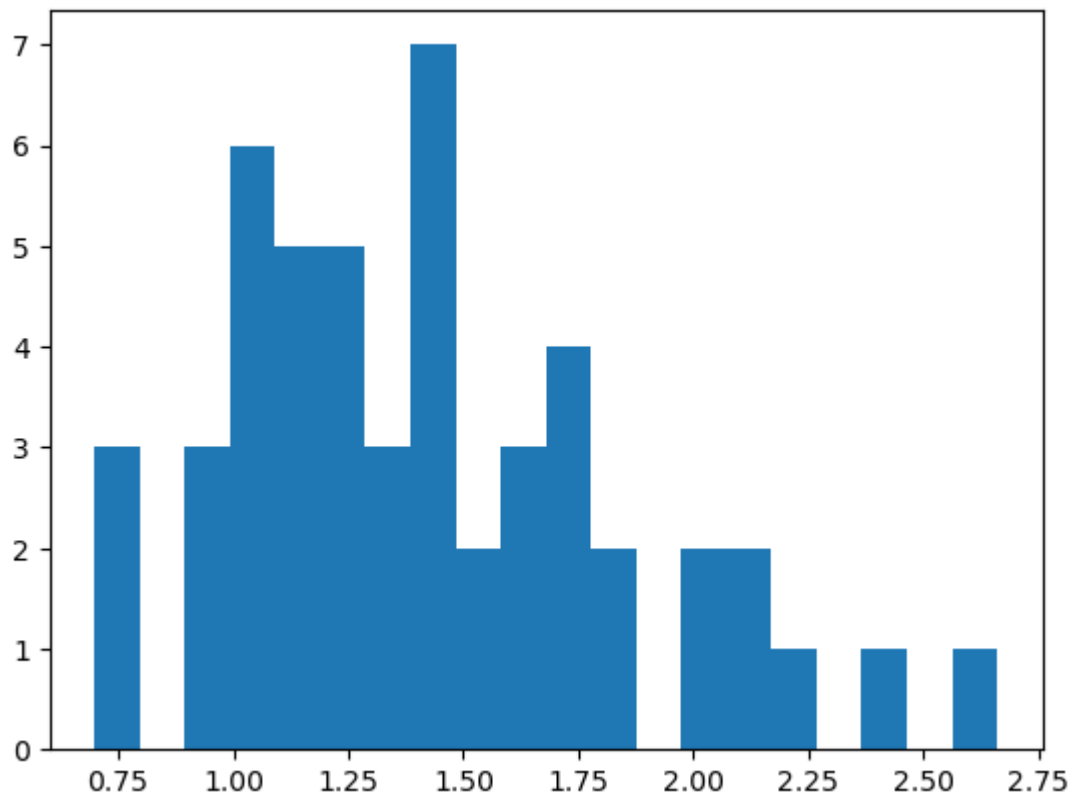
Total time taken: 1028.2772996425629 seconds

In [41]:

```
1 # print(np.var(ref_average1,axis=0))
2 print(np.var(ref_average2,axis=0))
3
4 for k in range(4):
5     plt.hist(ref_average2[:,k], bins = 20)
6     plt.show()
7
```

[0.35481283 0.29947082 0.18320262 0.4135741]





```

In [20]: 1 def random_point_ttests2(data, ref_distribution, points_removed, focus_c
2         '''runs welch test for multi run of simulated annealing for every p
3         param_distribution = ref_distribution
4         # focus_choices = ['prey', 'predator', 'both']
5         # focus_choices = ['prey', 'predator', 'both']
6         scores = [[], []]
7         p_values = {'prey': [], 'predator': [], 'both': []}
8         initial_temp = 20
9         cooling_constant = 0.10
10
11        for indx, choice in enumerate(focus_choices):
12            print(choice)
13            #We Iteratively increase the amount of points we remove
14            for npoints in points_removed:
15                print(f"points: {npoints}")
16                # print(f"Points removed: {npoints}")
17                limited_data = point_removal(data[:,0], data[:,1:3], npoint
18                #Getting distribution of averages
19                average_distribution = []
20                for k in range(30):
21                    parameter_list, best_score = multiple_runs_annealing(ir
22                    #Appending average
23                    average_distribution.append(np.mean(parameter_list, axi
24
25                average_distribution = np.array(average_distribution)
26                t_stat1, p_value1 = scipy.stats.ttest_ind(ref_distribution[
27                t_stat2, p_value2 = scipy.stats.ttest_ind(ref_distribution[
28                t_stat3, p_value3 = scipy.stats.ttest_ind(ref_distribution[
29                t_stat3, p_value4 = scipy.stats.ttest_ind(ref_distribution[
30                p_values[choice].append([p_value1, p_value2, p_value3, p_va
31
32        return p_values

```

```
In [44]: 1 start_time = time.time()
2
3 # Calculating p-values for t-test
4 points_removed_annealing = np.arange(3,16,3)
5 p_values_annealing = random_point_ttests2(data,ref_average2,points_remo
6
7 end_time = time.time()
8 # Calculating and printing the total time
9 total_time = end_time - start_time
10 print(f"Total time taken: {total_time/60} min")
```

```
prey
```

```
points: 3
```

```
C:\Users\Aleks\AppData\Local\Temp\ipykernel_764\951072704.py:25: RuntimeWarning: overflow encountered in exp
```

```
    acceptance_probability = min(np.exp(-(delta/temp)),1)#Calculate acceptance probability
```

```
C:\Users\Aleks\AppData\Local\Temp\ipykernel_764\951072704.py:25: RuntimeWarning: overflow encountered in scalar divide
```

```
    acceptance_probability = min(np.exp(-(delta/temp)),1)#Calculate acceptance probability
```

```
C:\Users\Aleks\AppData\Local\Temp\ipykernel_764\951072704.py:25: RuntimeWarning: divide by zero encountered in scalar divide
```

```
    acceptance_probability = min(np.exp(-(delta/temp)),1)#Calculate acceptance probability
```

```
points: 6
```

```
points: 9
```

```
points: 12
```

```
points: 15
```

```
C:\Users\Aleks\AppData\Local\Temp\ipykernel_764\951072704.py:25: RuntimeWarning: invalid value encountered in scalar divide
```

```
    acceptance_probability = min(np.exp(-(delta/temp)),1)#Calculate acceptance probability
```

```
C:\Users\Aleks\AppData\Local\Temp\ipykernel_764\264107711.py:4: RuntimeWarning: overflow encountered in scalar multiply
```

```
    dxdt = (alpha * x) - (beta * x * y) # Predator ODE
```

```
C:\Users\Aleks\AppData\Local\Temp\ipykernel_764\264107711.py:5: RuntimeWarning: overflow encountered in scalar multiply
```

```
    dydt = (delta * x * y) - (gamma * y) # Predator ODE
```

```
C:\Users\Aleks\AppData\Local\Temp\ipykernel_764\2008888099.py:4: RuntimeWarning: overflow encountered in square
```

```
    return np.mean((actual - predicted)**2)
```

```
C:\Users\Aleks\AppData\Local\Temp\ipykernel_764\2008888099.py:15: RuntimeWarning: overflow encountered in square
```

```
    err1 = (x1[indx_x] - x2[indx_x])**2
```

```
predator
```

```
points: 3
```

```
points: 6
```

```
points: 9
```

```
points: 12
```

```
points: 15
```

```
C:\Users\Aleks\AppData\Local\Temp\ipykernel_764\2008888099.py:16: RuntimeWarning: overflow encountered in square
```

```
    err2 = (y1[indx_y] - y2[indx_y])**2
```

both

points: 3

points: 6

points: 9

points: 12

points: 15

C:\Users\Aleks\AppData\Local\Temp\ipykernel_764\264107711.py:4: RuntimeWarning: invalid value encountered in scalar subtract

$dxdt = (\alpha * x) - (\beta * x * y)$ # Predator ODE

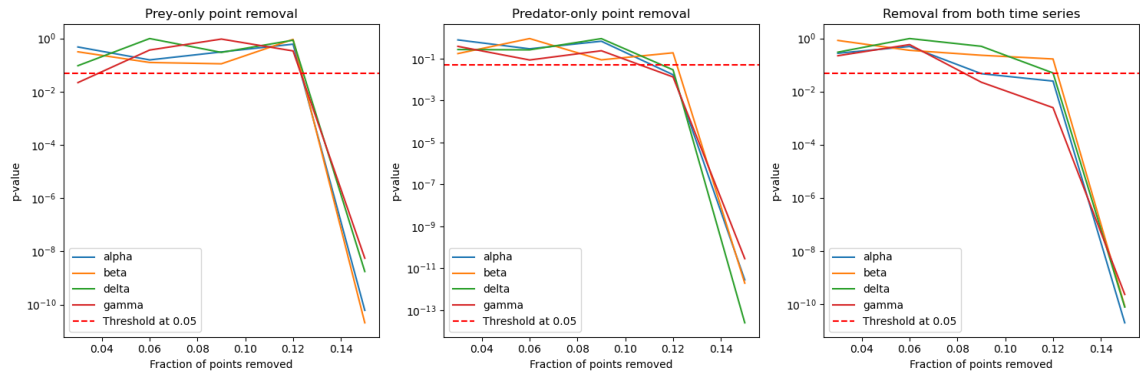
Total time taken: 63.41379015445709 min

In [46]:

```

1  # Fraction points removed
2
3  # Fraction points removed
4
5  fraction_points = np.arange(3,16,3) / 100
6  p_values_pre = np.array(p_values_annealing['prey'])
7  p_values_predator = np.array(p_values_annealing['predator'])
8  p_values_both = np.array(p_values_annealing['both'])
9
10 # Creating subplots
11 fig, axes = plt.subplots(1, 3, figsize=(15, 5))
12
13 # Plotting p-values when only prey points are removed
14 axes[0].plot(fraction_points, p_values_pre[:, 0], label='alpha')
15 axes[0].plot(fraction_points, p_values_pre[:, 1], label='beta')
16 axes[0].plot(fraction_points, p_values_pre[:, 2], label='delta')
17 axes[0].plot(fraction_points, p_values_pre[:, 3], label='gamma')
18 axes[0].axhline(y=0.05, color='r', linestyle='--', label='Threshold at
19 axes[0].set_ylabel('p-value')
20 axes[0].set_xlabel('Fraction of points removed')
21 axes[0].set_yscale('log')
22 axes[0].set_title('Prey-only point removal') # Add title to the first
23 axes[0].legend()
24
25 # Plotting p-values when only predator points are removed
26 axes[1].plot(fraction_points, p_values_predator[:, 0], label='alpha')
27 axes[1].plot(fraction_points, p_values_predator[:, 1], label='beta')
28 axes[1].plot(fraction_points, p_values_predator[:, 2], label='delta')
29 axes[1].plot(fraction_points, p_values_predator[:, 3], label='gamma')
30 axes[1].axhline(y=0.05, color='r', linestyle='--', label='Threshold at
31 axes[1].set_ylabel('p-value')
32 axes[1].set_xlabel('Fraction of points removed')
33 axes[1].set_yscale('log')
34 axes[1].set_title('Predator-only point removal')
35 axes[1].legend()
36
37 # Plotting p-values when both prey and predator points are removed
38 axes[2].plot(fraction_points, p_values_both[:, 0], label='alpha')
39 axes[2].plot(fraction_points, p_values_both[:, 1], label='beta')
40 axes[2].plot(fraction_points, p_values_both[:, 2], label='delta')
41 axes[2].plot(fraction_points, p_values_both[:, 3], label='gamma')
42 axes[2].axhline(y=0.05, color='r', linestyle='--', label='Threshold at
43 axes[2].set_ylabel('p-value')
44 axes[2].set_xlabel('Fraction of points removed')
45 axes[2].set_yscale('log')
46 axes[2].set_title('Removal from both time series')
47 axes[2].legend()
48
49 plt.savefig('welch_test_annealing', dpi=300)
50
51 # Adjusting layout
52 plt.tight_layout()
53 plt.show()

```



In []:

1