In [3]:
```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import norm
import pandas as pd
from scipy.integrate import odeint
from scipy.integrate import solve_ivp
import statistics
import random
import scipy.stats
```

Opening predator-prey dataset

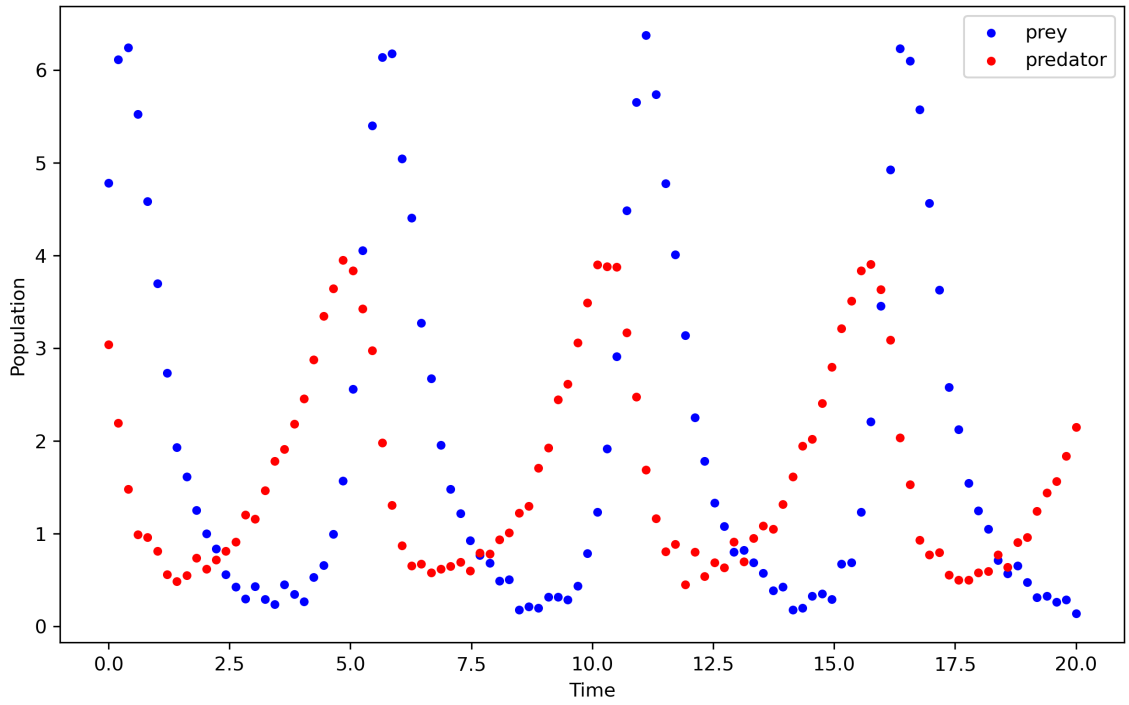
In [4]:
```
df = pd.read_csv('predator-prey-data.csv', index_col=False)
df.head()

```

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]:
```
# Loading data into read-only numpy arrays
data = df[['t','x','y']].values
# data[1], data[2] = data[2].copy(), data[1].copy()
data.flags.writeable = False


# Plotting
plt.figure(dpi =300, figsize=(10, 6))
point_width = 13
plt.scatter(data[:,0], data[:,1], label = 'prey', color = 'blue', s =pc
plt.scatter(data[:,0], data[:,2], label = 'predator', color = 'red', s=
plt.ylabel('Population')
plt.xlabel('Time')
plt.legend()

```

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



Objective functions

Defining volterra equations function

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

```

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


```

Defining objective functions

In [8]:
```
def negative_log_likelihood(actual, predicted, variance=1.0):
    '''Log likelyhood function'''
    # Assuming a normal distribution for simplicity
    log_likelihoods1 =norm.logpdf(actual[0], loc=predicted[0], scale=np
    log_likelihoods2 = norm.logpdf(actual[1], loc=predicted[1], scale=r

    return -np.sum((log_likelihoods1,log_likelihoods2))

def MSE(actual, predicted):
    '''Mean squared error'''
    return np.mean((actual - predicted)**2)
```

Algorithms & Optimisation

Simulated Annealing

In [9]:

```
def random_walk_annealing(parameters): #A random walk designed for anne
    lst = [parameter + np.random.normal(0, 0.5) for parameter in parame
    # Ensure all elements are positive
    while any(x <= 0 for x in lst):
        for indx in range(len(lst)):
            if lst[indx] <= 0:
                lst[indx] = max(0, parameters[indx] + np.random.normal(

    return lst



def simulated_annealing(initial_temp,cooling_constant, data, time, init

    temp = initial_temp #Scaling factor for random movement. We square
    start = parameters #Initial starting parameters
    x_n = start
    scores = [] #A score is just the value of the objective function ev

    current_est = predator_prey_integration(time, initial_conditions, x
    current_score = objective(data, current_est) #The current value of
    scores.append(current_score) #Keeping track of the values of the ob

    #cur = function(x) #The function value of the current x solution
    history = [x_n] #Stores previously searched x values

    for i in range (max_iterations):

        proposal = random_walk_annealing(x_n) #A new proposal for the p
        new_est = predator_prey_integration(time, initial_conditions, p
        new_score = objective(data, new_est) #Calculate new value of ob

        delta = new_score - current_score #Difference in objective func

        #if proposal < 0 or proposal > 1:
           #proposal = x_n # Reject proposal by setting it equal to pre

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

        #if delta < 0:
           #x_n = proposal ##Accept proposal
            #current_score = new_score

        if np.random.rand() < acceptance_probability: #else if it is no
            x_n = proposal #Accept proposal
            current_score = new_score

        scores.append(current_score)
        temp = cooling_constant**i * initial_temp #Cool temperature
        #print(temp)
        history.append(x_n) #Add to history

    return x_n, scores
```

In [8]:

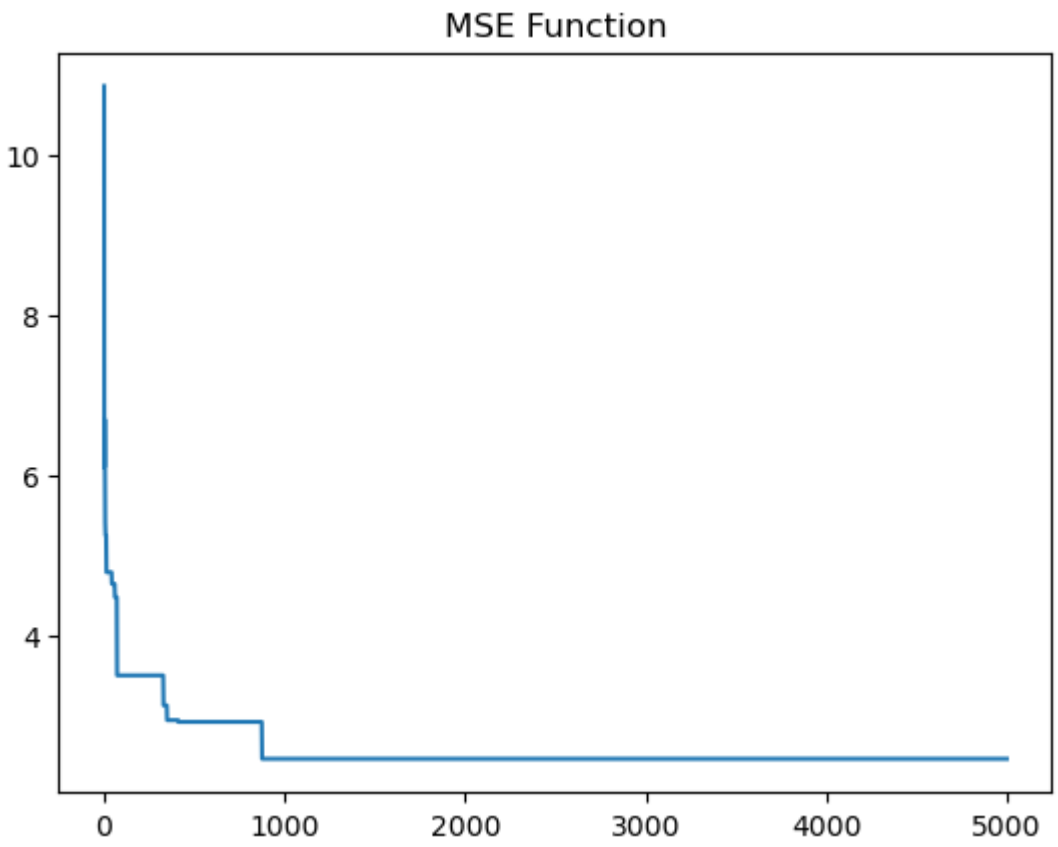
```
input_data = data[:,1:3]
initial_conditions = [input_data[0][0], input_data[0][1]]
t = data[:,0]

#Taking random draw for initial parameters (initial guess)
alpha = np.random.uniform(0,1)
beta = np.random.uniform(0,1)
delta = np.random.uniform(0,1)
gamma = np.random.uniform(0,1)
parameters = [alpha, beta, delta, gamma]
initial_temp = 20
cooling_constant = 0.10

# Using MSE
x_best, scores = simulated_annealing(initial_temp,cooling_constant, inp
```

```
C:\Users\kayad\AppData\Local\Temp\ipykernel_27100\3853094899.py:38: Runtim
eWarning: overflow encountered in exp
  acceptance_probability = min(np.exp(-(delta/temp)),1)#Calculate acceptan
ce probability
C:\Users\kayad\AppData\Local\Temp\ipykernel_27100\3853094899.py:38: Runtim
eWarning: overflow encountered in scalar divide
  acceptance_probability = min(np.exp(-(delta/temp)),1)#Calculate acceptan
ce probability
C:\Users\kayad\AppData\Local\Temp\ipykernel_27100\3853094899.py:38: Runtim
eWarning: divide by zero encountered in scalar divide
  acceptance_probability = min(np.exp(-(delta/temp)),1)#Calculate acceptan
ce probability
```

Solution discovery over iterations

In [9]:
```
xvalues = [x for x in range(0,len(scores))]



plt.plot(xvalues,scores)
plt.title("MSE Function")

print("LOWEST MSE: " + str(scores[-1]))
```

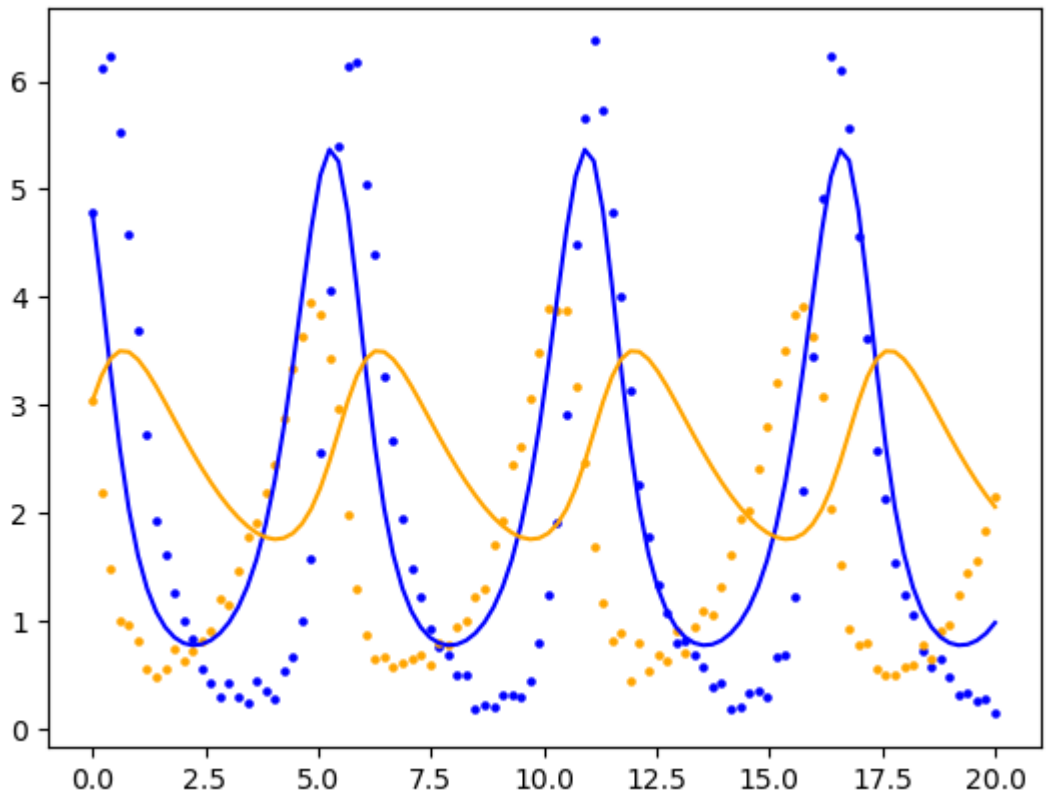
```
LOWEST MSE: 2.4498445634217485
```



Curve fit

In [10]:
```
# t, x ,y = data
initial_conditions = [input_data[0][0], input_data[0][1]]
t = data[:,0]
scaling = 2

parameters = x_best

# Using MSE
x = predator_prey_integration(t,initial_conditions,parameters)

#plt.figure(dpi =300, figsize=(6, 5))
point_width = 13
plt.plot(t, x[:,0],color = "b")
plt.plot(t, x[:,1],color = "orange")

plt.scatter(t, data[:,1], color= 'blue', s = 5)
plt.scatter(t, data[:,2], color= 'orange', s = 5)

mse_prey = MSE(data[:,1],x[:,0]) #MSE for fitted curve
mse_predator = MSE(data[:,2],x[:,1])
mse_total = mse_prey + mse_predator
print("Mean Square Error: " +str(mse_total))
```

```
Mean Square Error: 4.899689126843497
```



Calculation of mean and variance

In [10]:
```
### Distribution of parameters for multiple runs

def multiple_runs_annealing(initial_temp,cooling_constant,input_data,t,

    mse_total_list = []


    for i in range(n_runs):

        x_best, scores = simulated_annealing(initial_temp,cooling_const

        x = predator_prey_integration(t,initial_conditions,x_best)
        mse_prey = MSE(data[:,1],x[:,0])
        mse_predator = MSE(data[:,2],x[:,1])
        mse_total = mse_prey + mse_predator

        mse_total_list.append(mse_total) #Add total MSE for this simula

    return mse_total_list

```

In [12]:
```
input_data = data[:,1:3]
initial_conditions = [input_data[0][0], input_data[0][1]]
t = data[:,0]
#Taking random draw for initial parameters (initial guess)
alpha = np.random.uniform(0,1)
beta = np.random.uniform(0,1)
delta = np.random.uniform(0,1)
gamma = np.random.uniform(0,1)
parameters = [alpha, beta, delta, gamma]
parameters = [alpha, beta, delta, gamma]

initial_temp = 20
cooling_constant = 0.10

mse_total_list = multiple_runs_annealing(initial_temp,cooling_constant,

#print(mse_total_list)
```

```
C:\Users\kayad\AppData\Local\Temp\ipykernel_27100\3853094899.py:38: Runtim
eWarning: overflow encountered in exp
  acceptance_probability = min(np.exp(-(delta/temp)),1)#Calculate acceptan
ce probability
C:\Users\kayad\AppData\Local\Temp\ipykernel_27100\3853094899.py:38: Runtim
eWarning: overflow encountered in scalar divide
  acceptance_probability = min(np.exp(-(delta/temp)),1)#Calculate acceptan
ce probability
C:\Users\kayad\AppData\Local\Temp\ipykernel_27100\3853094899.py:38: Runtim
eWarning: divide by zero encountered in scalar divide
  acceptance_probability = min(np.exp(-(delta/temp)),1)#Calculate acceptan
ce probability
C:\Users\kayad\anaconda3\Lib\site-packages\scipy\integrate\_odepack_py.py:
248: ODEintWarning: Excess work done on this call (perhaps wrong Dfun typ
e). 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]:

```
input_data = data[:,1:3]
initial_conditions = [input_data[0][0], input_data[0][1]]
t = data[:,0]
#Taking random draw for initial parameters (initial guess)
alpha = np.random.uniform(0,1)
beta = np.random.uniform(0,1)
delta = np.random.uniform(0,1)
gamma = np.random.uniform(0,1)
parameters = [alpha, beta, delta, gamma]
parameters = [alpha, beta, delta, gamma]

initial_temp = 20

cooling_constants = np.arange(0.10,1,0.10)

mean_and_sd_per_simulation = []

for constant in cooling_constants:
    print(constant)
    mse_total_list = multiple_runs_annealing(initial_temp,constant,inpu
    mean_mse_annealing = np.mean(mse_total_list)
    std_mse_annealing = np.std(mse_total_list)
    mean_and_sd_per_simulation.append([(mean_mse_annealing,std_mse_anne


```

```
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 acceptan
ce probability
C:\Users\Aleks\AppData\Local\anaconda3\Lib\site-packages\scipy\integrate\_
odepack_py.py:248: ODEintWarning: Excess work done on this call (perhaps w
rong 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 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
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
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_iterations=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 objective function based on new function values
     33     delta = new_score - current_score #Difference in objective function 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 increasing "
    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, mu,
    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
t)
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]:

```
def random_walk(parameters):
    lst = [parameter + np.random.normal(0, 1) for parameter in paramete
    # Ensure all elements are positive
    while any(x <= 0 for x in lst):
        for indx in range(len(lst)):
            if lst[indx] <= 0:
                lst[indx] = max(0, parameters[indx] + np.random.normal(

    return lst

def hill_climbing(initial_temp,cooling_constant, data, time, initial_co
    '''Tries to find the best solution using random walker'''
    # Initialize starting parameter state
    scores = []
    x_n = parameters

    current_est = predator_prey_integration(time, initial_conditions, x
    current_score = objective(data, current_est)
    scores.append(current_score)

    for _ in range(max_iterations):
        # Generate a random walk for parameters
        x_n_1 = random_walk(x_n)

        # Calculate the current and next estimations
        current_est = predator_prey_integration(time, initial_condition
        new_estimation = predator_prey_integration(time, initial_condit
        new_score = objective(data, new_estimation)

        # If the next estimation is better, update the parameters
        if new_score < current_score:
            current_score = new_score
            x_n = x_n_1
            scores.append(current_score)

    return [x_n, scores]

```

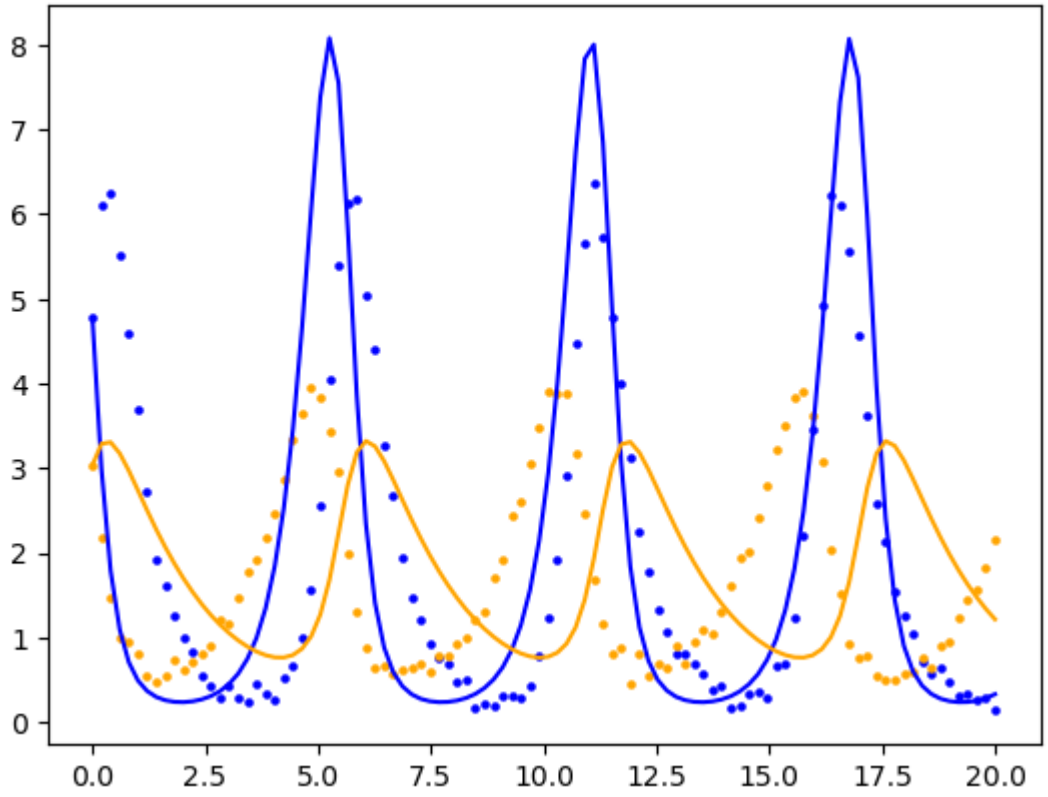
In [35]:

```
input_data = data[:,1:3]
initial_conditions = [input_data[0][0], input_data[0][1]]
t = data[:,0]
#Taking random draw for initial parameters (initial guess)
alpha = np.random.uniform(0,1)
beta = np.random.uniform(0,1)
delta = np.random.uniform(0,1)
gamma = np.random.uniform(0,1)
parameters = [alpha, beta, delta, gamma]

# Using MSE
x_best = hill_climbing(initial_temp,cooling_constant, input_data, t, ir

```

Curve fit

In [17]:
```
# t, x ,y = data
initial_conditions = [input_data[0][0], input_data[0][1]]
t = data[:,0]

parameters = x_best

# Using MSE
x = predator_prey_integration(t,initial_conditions,parameters)

point_width = 13
plt.plot(t, x[:,0],color = "b")
plt.plot(t, x[:,1],color = "orange")

plt.scatter(t, data[:,1], color= 'blue', s = 5)
plt.scatter(t, data[:,2], color= 'orange', s = 5)

mse_prey = MSE(data[:,1],x[:,0]) #MSE for fitted curve
mse_predator = MSE(data[:,2],x[:,1])
mse_total = mse_prey + mse_predator
print("Mean Square Error: " +str(mse_total))

```

```
Mean Square Error: 6.016318958977161
```



In [36]:
```
def multiple_runs_hill_climbing(initial_temp,cooling_constant, input_da

    mse_total_list = []

    for i in range(n_runs):

        x_best = hill_climbing(initial_temp,cooling_constant, input_dat

        x = predator_prey_integration(t,initial_conditions,x_best)

        mse_prey = MSE(data[:,1],x[:,0])
        mse_predator = MSE(data[:,2],x[:,1])
        mse_total = mse_prey + mse_predator

        mse_total_list.append(mse_total) #Add total MSE for this simula

    return mse_total_list

```

In [37]:
```
input_data = data[:,1:3]
initial_conditions = [input_data[0][0], input_data[0][1]]
t = data[:,0]
#Taking random draw for initial parameters (initial guess)
alpha = np.random.uniform(0,1)
beta = np.random.uniform(0,1)
delta = np.random.uniform(0,1)
gamma = np.random.uniform(0,1)
parameters = [alpha, beta, delta, gamma]


mse_total_list = multiple_runs_hill_climbing(initial_temp,cooling_const
```

In [20]:
```
mean_mse_hill_climbing = np.mean(mse_total_list)
std_mse_hill_climbing = np.std(mse_total_list)

print("Average MSE = " + str(mean_mse_hill_climbing))
print("Standard deviation of MSE = " + str(std_mse_hill_climbing))
```

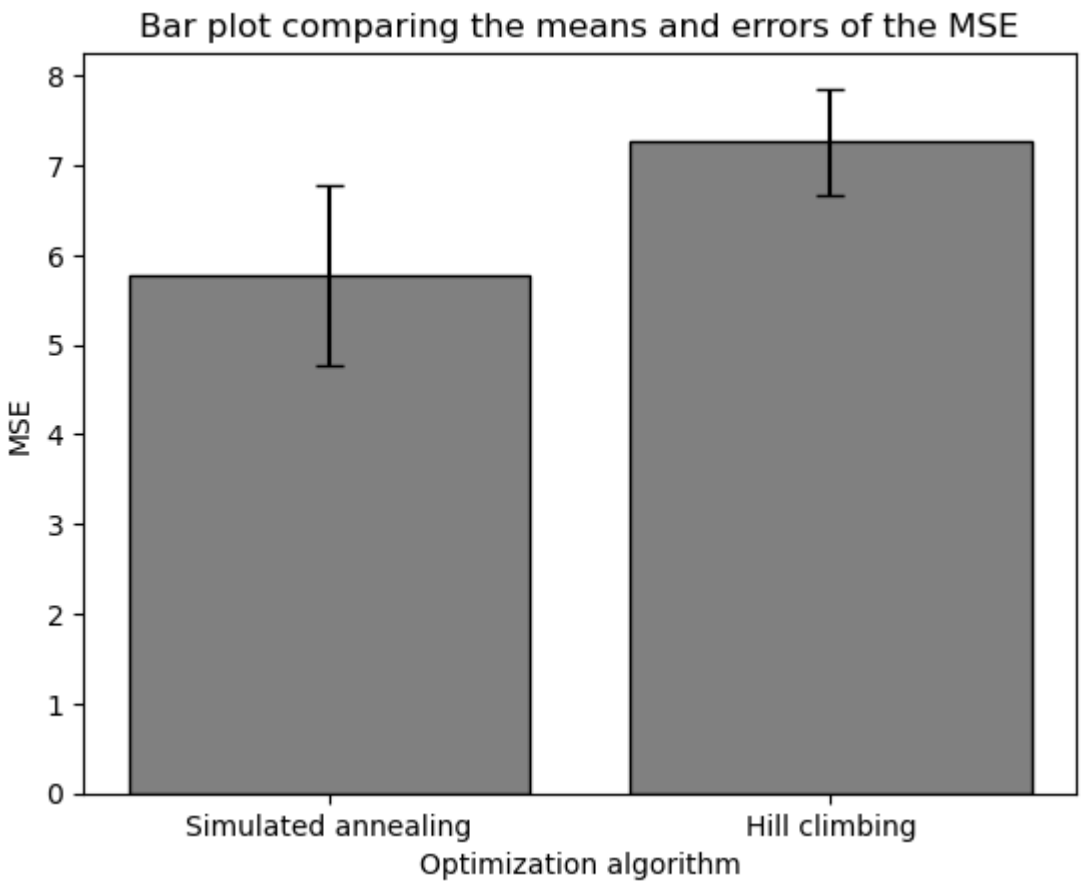
```
Average MSE = 7.258042040055488
Standard deviation of MSE = 0.5962174191554426
```

Comparison of Optimisation Algorithms

MSE objecive function

In [21]:
```
x = ["Simulated annealing", "Hill climbing"]
mse_list = [mean_mse_annealing,mean_mse_hill_climbing] #Stores the mear
std_list = [std_mse_annealing,std_mse_hill_climbing]

plt.bar(x, mse_list, yerr = std_list,color = "grey",ec = "black", ecolo
plt.title("Bar plot comparing the means and errors of the MSE")
plt.ylabel("MSE")
plt.xlabel("Optimization algorithm")
```

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



In [48]:
```

good= data[:,1:3]
bad = np.zeros_like(good)

print(negative_log_likelihood(x,good))
print()
print(negative_log_likelihood(x,bad))
```

Reverse engineering with removal of points targeted

In [18]:

```
def MSE2(actual, predicted):
    '''Mean squared error'''
    x1 = actual[:, 0]
    y1 = actual[:, 1]

    #Getting useful indexes
    indx_x = np.where(~np.isnan(x1))
    indx_y = np.where(~np.isnan(y1))

    x2, y2 = predicted[:, 0], predicted[:, 1]
    err1 = (x1[indx_x] - x2[indx_x])**2
    err2 = (y1[indx_y] - y2[indx_y])**2
    err = np.concatenate([err1, err2])

    return np.nanmean([err])

def MAE(actual, predicted):
    '''Mean absolute error'''
    return np.mean(np.abs(actual - predicted))

def MAE2(actual, predicted):
    x1 = actual[0]
    y1 = actual[1]

    #Getting useful indexes
    indx_x = np.where(~np.isnan(x1))
    indx_y = np.where(~np.isnan(y1))

    x2, y2 = predicted[:, 0], predicted[:, 1]
    err1 = abs(x1[indx_x] - x2[indx_x])
    err2 = abs(y1[indx_y] - y2[indx_y])
    err = np.concatenate([err1, err2])
    err = np.concatenate([err1, err2])

    return np.nanmean([err])


def point_removal(time, input_data, points_removed, Focus = 0):
    prey = input_data.T[0].copy()
    predator = input_data.T[1].copy()

    # initialize set up for removing points randomly given the bounds
    removal_options = np.arange(0,len(time))

    # choose points to be removed randomly
    if points_removed > len(removal_options):
        points_removed = len(removal_options)
        print('WARNING: Maximum number of points that can be removed ha
    removed_points_indices = random.choices(removal_options, k = points

    # remove points based on choices for points to be removed
    if Focus == 0:
        for i in removed_points_indices:
            prey[i] = None
            predator[i] = None

    elif Focus == 1:
        for i in removed_points_indices:
            prey[i] = None

    elif Focus == 2:
```

```
        for i in removed_points_indices:
            predator[i] = None

    return np.array(time), np.array(prey), np.array(predator)

```

In [19]:

```
def extrema_removal(time, input_data, points_removed, Focus = 0):
    prey = input_data.T[0].copy()
    predator = input_data.T[1].copy()

    # Calculate mean and variance to set regions for data
    mean_prey_population, mean_predator_population = np.mean(prey), np.
    variance_prey, variance_predator = statistics.variance(prey), stati

    # set upper bound and lower bound for point removals
    ub_prey, lb_prey = mean_prey_population + 0.45 * variance_prey, mea
    ub_predator, lb_predator = mean_predator_population + 0.9 *variance

    # initialize set up for removing points randomly given the bounds
    prey_options = []
    predator_options = []
    # enumerate through list of stored points
    for index, prey_count in enumerate(prey):
        # check if they are in specified region
        if prey_count >= ub_prey or prey_count <= lb_prey:
            prey_options.append([index, prey_count, predator[index]])
    for index, predator_count in enumerate(predator):
        if predator_count >= ub_predator or predator_count <= lb_predat
            predator_options.append([index, prey[index], predator_count

    # remove points from list depending on which focus is set
    removal_options = []
    if Focus == 0:
        removal_options = removal_options + prey_options + predator_opt
    elif Focus == 1:
        removal_options = removal_options + prey_options
    elif Focus == 2:
        removal_options = removal_options + predator_options
    else:
        print('Error: Removal option not known. Try either both, prey,

    # choose points to be removed randomly
    if points_removed > len(removal_options):
        points_removed = len(removal_options)
        print('WARNING: Maximum number of points that can be removed ha
    removed_points_indices = random.choices(np.array(removal_options).T

    # turn the list into integers so we can remove them based on the in
    integer_array = []
    for counter in range(len(removed_points_indices)):
        integer_array.append(int(removed_points_indices[counter]))

    # update the lists based on points we wanted to remove
    if Focus == 0:
        for i in integer_array:
            prey[i] = None
            predator[i] = None

    elif Focus == 1:
        for i in integer_array:
            prey[i] = None

    elif Focus == 2:
        for i in integer_array:
            predator[i] = None

    return np.array(time), np.array(prey), np.array(predator)
```

```
62
63
```

In [314]:

```
def midpoint_removal(time, input_data, points_removed, Focus = 0):
    prey = input_data.T[0].copy()
    predator = input_data.T[1].copy()

    # Calculate mean and variance to set regions for data
    mean_prey_population, mean_predator_population = np.mean(prey), np.
    variance_prey, variance_predator = statistics.variance(prey), stati

    # set upper bound and lower bound for point removals
    ub_prey, lb_prey = mean_prey_population + 0.45 * variance_prey, mea
    ub_predator, lb_predator = mean_predator_population + 0.9 *variance

    # initialize set up for removing points randomly given the bounds
    prey_options = []
    predator_options = []
    # enumerate through list of stored points
    for index, prey_count in enumerate(prey):
        # check if they are in specified region
        if prey_count <= ub_prey and prey_count >= lb_prey:
            prey_options.append([index, prey_count, predator[index]])
    for index, predator_count in enumerate(predator):
        if predator_count <= ub_predator and predator_count >= lb_preda
            predator_options.append([index, prey[index], predator_count

    # remove points from list depending on which focus is set
    removal_options = []
    if Focus == 0:
        removal_options = removal_options + prey_options + predator_opt
    elif Focus == 1:
        removal_options = removal_options + prey_options
    elif Focus == 2:
        removal_options = removal_options + predator_options
    else:
        print('Error: Removal option not known. Try either both, prey,

    # choose points to be removed randomly
    if points_removed > len(removal_options):
        points_removed = len(removal_options)
        print('WARNING: Maximum number of points that can be removed ha
    removed_points_indices = random.choices(np.array(removal_options).T

    # turn the list into integers so we can remove them based on the in
    integer_array = []
    for counter in range(len(removed_points_indices)):
        integer_array.append(int(removed_points_indices[counter]))

    # update the lists based on points we wanted to remove
    if Focus == 0:
        for i in integer_array:
            prey[i] = None
            predator[i] = None

    elif Focus == 1:
        for i in integer_array:
            prey[i] = None

    elif Focus == 2:
        for i in integer_array:
            predator[i] = None

    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]))
```

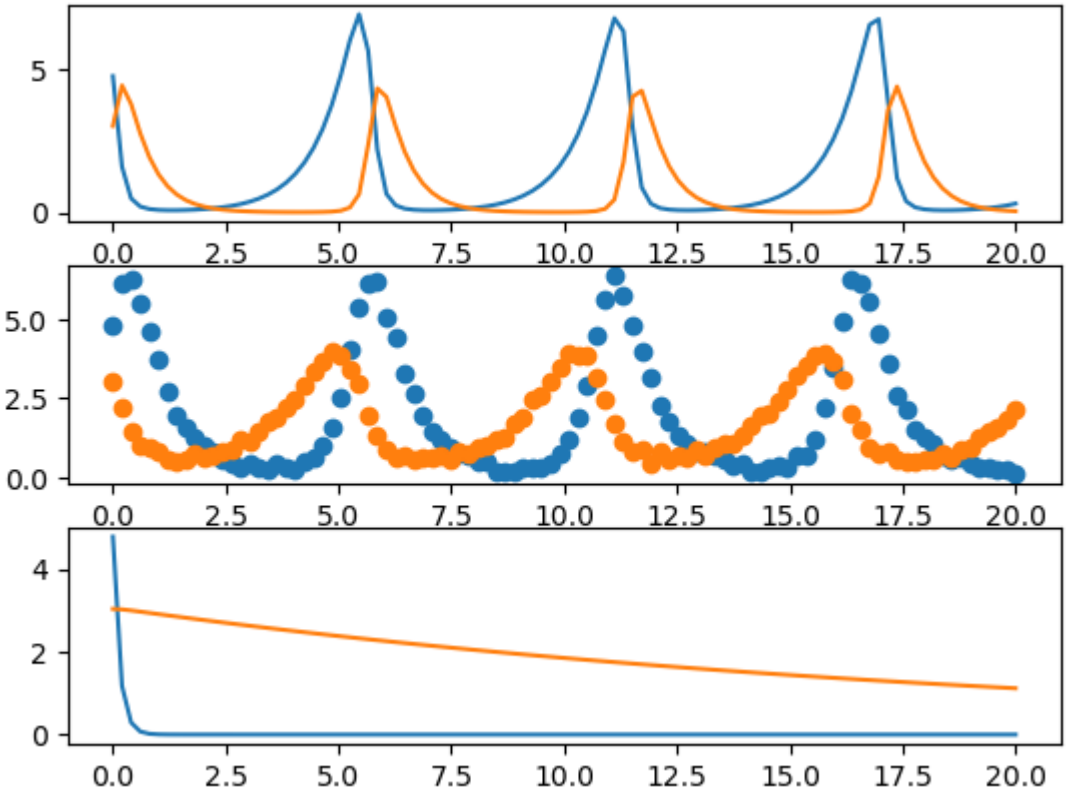
In [16]:

```
def inferance_removal(initial_temp,cooling_constant, objective, algorit
    # define input data based on number of points we want to remove
    new_data_set = removal_type(time, input_data, points_removed, Focus
    time_new = new_data_set[0]
    prey_new = new_data_set[1]
    predator_new = new_data_set[2]

    # prepare data to match input requirements of algorithm (so far onl
    input_data_new = []
    for i in range(len(time_new)):
        input_data_tulple = [prey_new[i], predator_new[i]]
        input_data_new.append(input_data_tulple)

    # apply algorithm to find best fitted parameters given new data set
    x_best, scores = algorithm(initial_temp,cooling_constant, np.array(

    return x_best, scores

# parameters = [0.5, 0.6, 0.7, 0.8]
# x = inferance_removal(initial_temp,cooling_constant, MSE2, simulated_
# print(x[0])
```

In [268]:

```
initial_conditions = [input_data[0][0], input_data[0][1]]
t = data[:,0]
time = t
input_data = data[:,1:3]

best = inferance_removal(initial_temp,cooling_constant, MSE2, simulated
removed = inferance_removal(initial_temp,cooling_constant, MSE2, simula

best_params_tests = best[0]
reduced_params_test = removed[0]

res1 = predator_prey_integration(time,initial_conditions,best_params_te
res2 = predator_prey_integration(time,initial_conditions,reduced_params

MSE_ = MSE2(res1, input_data)
MSE = MSE2(res2, input_data)

print(MSE_, MSE)

plt.figure()
plt.subplot(311)
plt.plot(data[:,0], res1[:,0])
plt.plot(data[:,0], res1[:,1])

plt.subplot(312)
plt.scatter(time, data[:,1])
plt.scatter(time, data[:, 2])

plt.subplot(313)
plt.plot(data[:,0], res2[:,0])
plt.plot(data[:,0], res2[:,1])
plt.show()
plt.close
```

```
C:\Users\kayad\AppData\Local\Temp\ipykernel_27100\3853094899.py:38: Runtim
eWarning: overflow encountered in exp
  acceptance_probability = min(np.exp(-(delta/temp)),1)#Calculate acceptan
ce probability
C:\Users\kayad\AppData\Local\Temp\ipykernel_27100\3853094899.py:38: Runtim
eWarning: overflow encountered in scalar divide
  acceptance_probability = min(np.exp(-(delta/temp)),1)#Calculate acceptan
ce probability
C:\Users\kayad\AppData\Local\Temp\ipykernel_27100\3853094899.py:38: Runtim
eWarning: divide by zero encountered in scalar divide
  acceptance_probability = min(np.exp(-(delta/temp)),1)#Calculate acceptan
ce probability

WARNING: Maximum number of points that can be removed has been exceeded. A
ll points possible within the given bound have now been removed.
3.7074011653569787 4.557827773152935
```



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

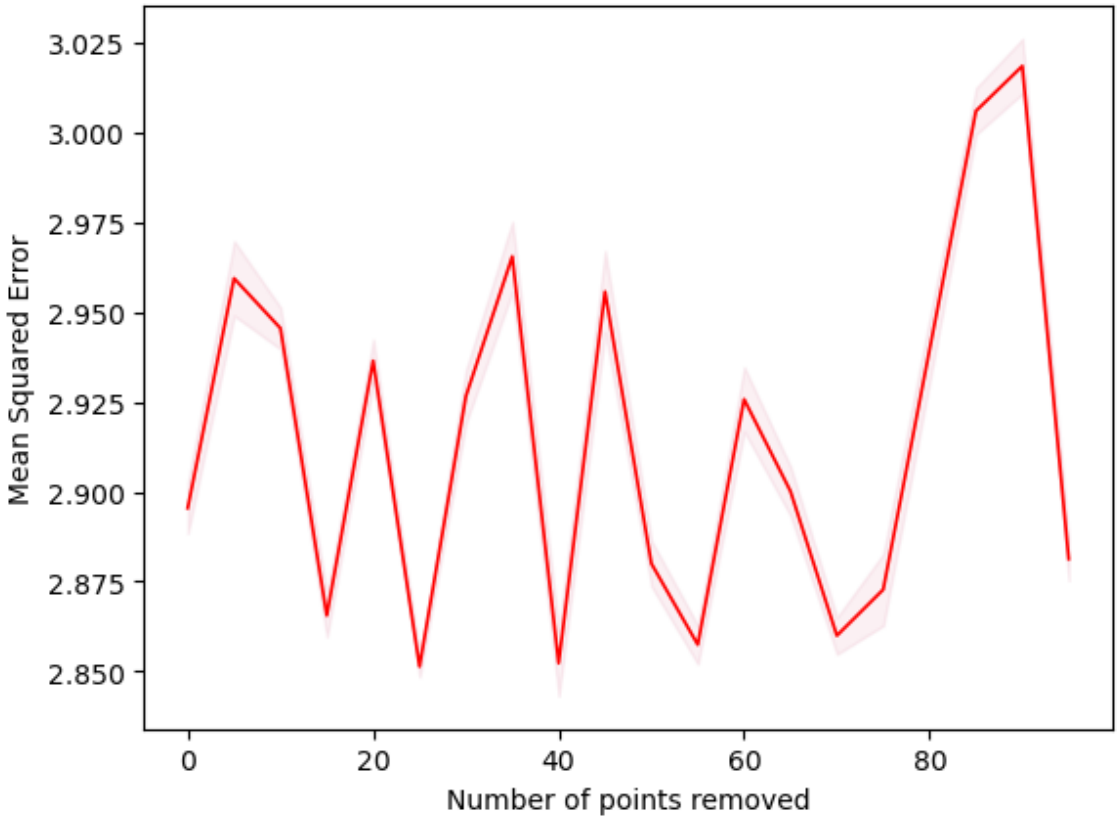
In [378]:

```
def multirun_SA_point_removal(iterations, simulations, initial_temperat
    # initialize objective score sufficiently high
    Lowest_MSE = [100000]
    best_param_overall = []
    # start simulated annealing from varying initial parameters
    for simulation_count in range(simulations):
        init_conditions = original_data[0]
        initial_parameter_guess = [np.random.uniform(0,1), np.random.ur
        # iterate several times for each initial condition
        for iteration_count in range(iterations):
            # find parameters best fitted
            best_found_parameters = inferance_removal(initial_temperatu
            # integrate
            Reversed_curve = predator_prey_integration(timestep,init_cc
            # compute objective function
            MSE = objective(original_data, Reversed_curve)
            # check if current objective lower than initial ones and if
            if MSE < Lowest_MSE[-1]:
                best_param_overall.append(best_found_parameters[0])
                Lowest_MSE.append(MSE)
    return best_param_overall[-1], Lowest_MSE[-1]

```

In [425]:

```
# Points removed with aggressive cooling for prey population, simulated
simulations = 2
iterations = 1
init_temp = 20
cooling = 0.1 # cooling strategy more aggressive
t = data[:,0]
time = t
inp = data[:,1:3]

all_scores_prey = []
plt.figure(dpi = 300)
for counter in range(25):
    # storage
    paramets_prey = []
    objective_results_prey = []
    # point removal loop (removing points at steps of 2)
    npoints = np.arange(0,100,5)
    for p_removed in npoints:
        best_parameters_found_prey_removal = multirun_SA_point_removal(
        paramets_prey.append(best_parameters_found_prey_removal[0])
        objective_results_prey.append(best_parameters_found_prey_remova
    all_scores_prey.append(objective_results_prey)
means_scores_prey = []
variance_scores_prey = []
for index in range(len(npoints)):
    current_score_count = np.array(all_scores_prey)[:,index]
    means_scores_prey.append(np.mean(current_score_count))
    variance_scores_prey.append(statistics.variance(current_score_count
ci = 1.96 * np.array(variance_scores_prey)/25
fig, ax = plt.subplots()
ax.plot(npoints,means_scores_prey, linewidth = '1.2', color = 'red')
ax.fill_between(npoints, (np.array(means_scores_prey)-ci), (np.array(me
#plt.plot(npoints, means_scores_prey, linewidth = '1.2', color = 'red')
plt.xlabel('Number of points removed')
plt.ylabel('Mean Squared Error')
plt.show()
plt.close()
```

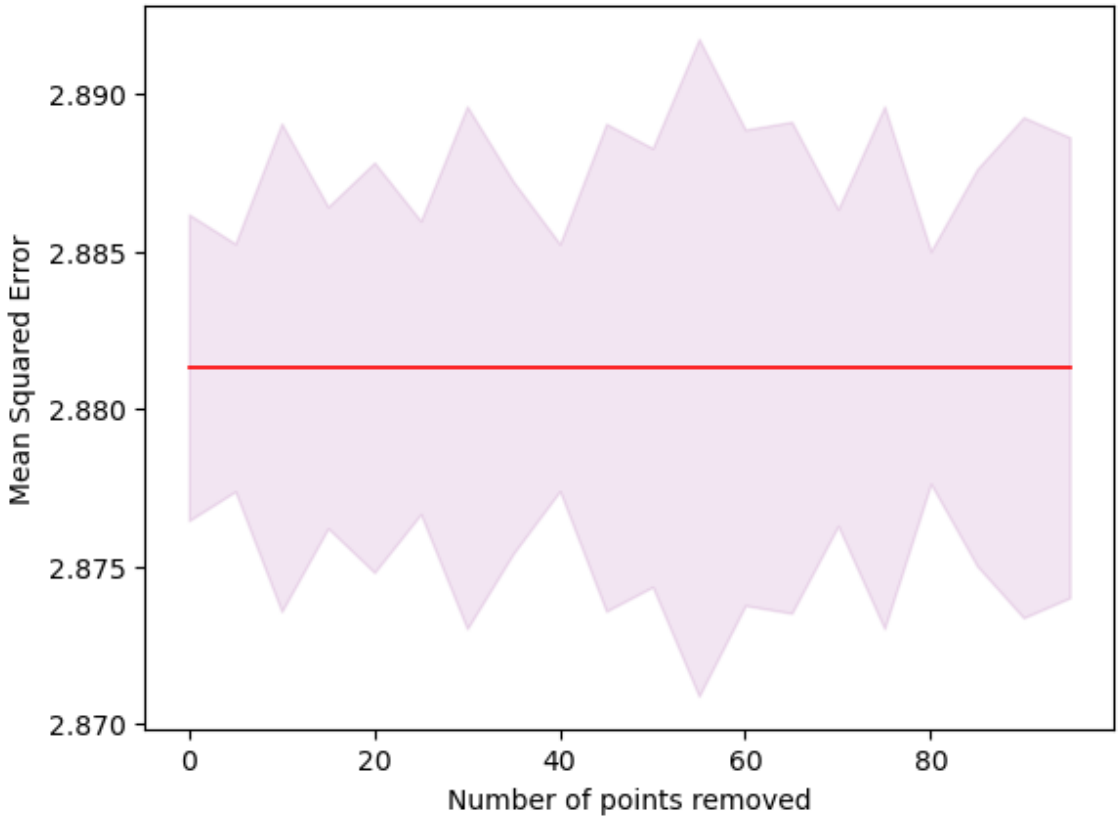
```
C:\Users\kayad\AppData\Local\Temp\ipykernel_27100\3853094899.py:38: Runtim
eWarning: overflow encountered in exp
  acceptance_probability = min(np.exp(-(delta/temp)),1)#Calculate acceptan
ce probability
C:\Users\kayad\AppData\Local\Temp\ipykernel_27100\3853094899.py:38: Runtim
eWarning: overflow encountered in scalar divide
  acceptance_probability = min(np.exp(-(delta/temp)),1)#Calculate acceptan
ce probability
C:\Users\kayad\AppData\Local\Temp\ipykernel_27100\3853094899.py:38: Runtim
eWarning: divide by zero encountered in scalar divide
  acceptance_probability = min(np.exp(-(delta/temp)),1)#Calculate acceptan
ce probability

<Figure size 1920x1440 with 0 Axes>
```


3.025
3.000
2.975
2.950
2.925
2.900
2.875
2.850
Mean Squared Error
0
20
40
60
80
Number of points removed

In [429]:

```
# Points removed with aggressive cooling for prey population, simulated
simulations = 2
iterations = 1
init_temp = 20
cooling = 0.1 # cooling strategy more aggressive
t = data[:,0]
time = t
inp = data[:,1:3]

all_scores_predprey = []
plt.figure(dpi = 300)
for counter in range(25):
    # storage
    paramets_predprey = []
    objective_results_predprey = []
    # point removal loop (removing points at steps of 2)
    npoints = np.arange(0,100,5)
    for p_removed in npoints:
        best_parameters_found_predprey_removal = multirun_SA_point_remo
        paramets_predprey.append(best_parameters_found_predprey_removal
        objective_results_predprey.append(best_parameters_found_predpre
    all_scores_predprey.append(objective_results_predprey)
means_scores_predprey = []
variance_scores_predprey = []
for index in range(len(npoints)):
    current_score_count_predprey = np.array(all_scores_predprey)[:,inde
    means_scores_predprey.append(np.mean(current_score_count))
    variance_scores_predprey.append(statistics.variance(current_score_c
ci_predprey = 1.96 * np.array(variance_scores_predprey)/25
plt.figure(dpi = 300)
fig, ax = plt.subplots()
ax.plot(npoints,means_scores_predprey, linewidth = '1.2', color = 'red'
ax.fill_between(npoints, (np.array(means_scores_predprey)-ci_predprey),
#plt.plot(npoints, means_scores_prey, linewidth = '1.2', color = 'red')
plt.xlabel('Number of points removed')
plt.ylabel('Mean Squared Error')
plt.show()
plt.close()
```

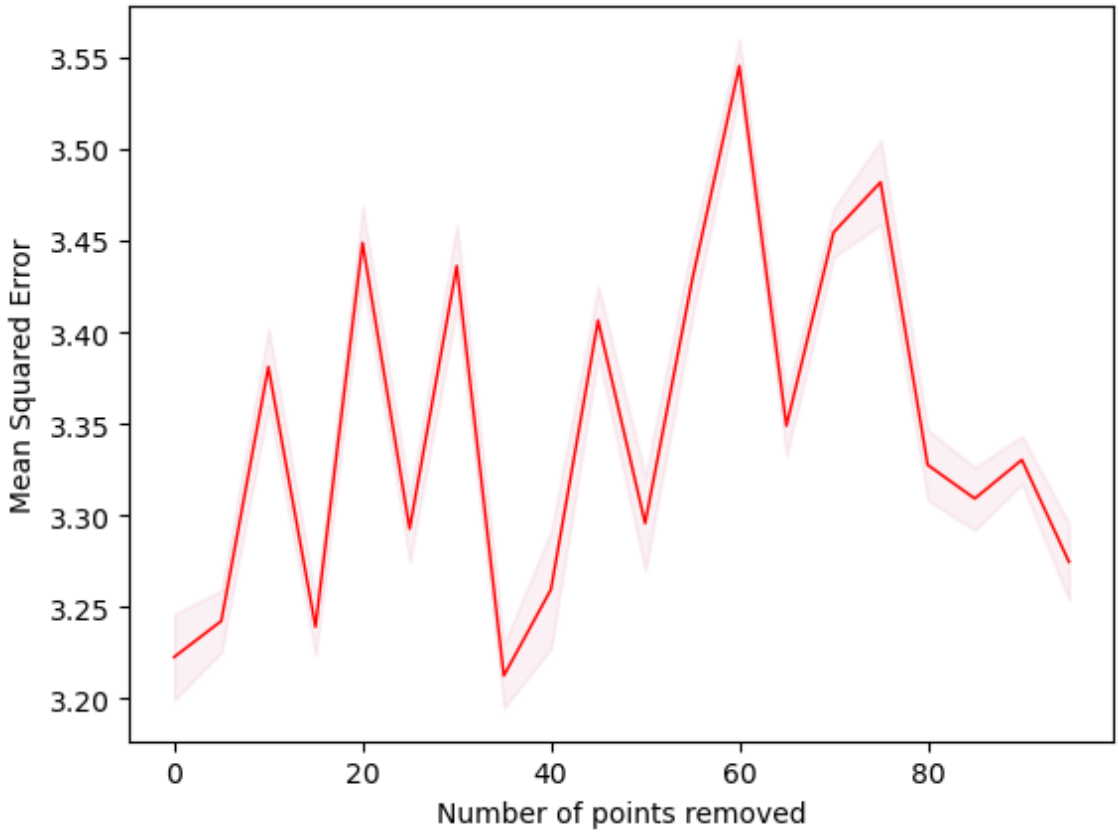
```
C:\Users\kayad\AppData\Local\Temp\ipykernel_27100\3853094899.py:38: Runtim
eWarning: overflow encountered in exp
  acceptance_probability = min(np.exp(-(delta/temp)),1)#Calculate acceptan
ce probability
C:\Users\kayad\AppData\Local\Temp\ipykernel_27100\3853094899.py:38: Runtim
eWarning: overflow encountered in scalar divide
  acceptance_probability = min(np.exp(-(delta/temp)),1)#Calculate acceptan
ce probability
C:\Users\kayad\AppData\Local\Temp\ipykernel_27100\3853094899.py:38: Runtim
eWarning: divide by zero encountered in scalar divide
  acceptance_probability = min(np.exp(-(delta/temp)),1)#Calculate acceptan
ce probability

<Figure size 1920x1440 with 0 Axes>

<Figure size 1920x1440 with 0 Axes>
```


2.890
2.885
2.880
2.875
2.870
Mean Squared Error
0
20
40
60
80
Number of points removed

In [424]:

```
# Points removed with slower cooling simulated annealing and slower coo
simulations = 2
iterations = 1
init_temp = 20
cooling = 0.9
t = data[:,0]
time = t
inp = data[:,1:3]

all_scores_predator = []
plt.figure(dpi = 300)
for counter in range(25):
    # storage
    paramets_predator = []
    objective_results_predator = []
    # point removal loop (removing points at steps of 2)
    npoints = np.arange(0,100,5)
    for p_removed in npoints:
        best_parameters_found_predator_removal = multirun_SA_point_remo
        paramets_predator.append(best_parameters_found_predator_removal
        objective_results_predator.append(best_parameters_found_predato
    all_scores_predator.append(objective_results_predator)
    #plt.plot(npoints ,objective_results_prey, linewidth = '0.6', color
means_scores_predator = []
variance_scores_predator = []
for index in range(len(npoints)):
    current_score_count_predator = np.array(all_scores_predator)[:,inde
    means_scores_predator.append(np.mean(current_score_count_predator))
    variance_scores_predator.append(statistics.variance(current_score_c
ci_predator = 1.96 * np.array(variance_scores_predator)/25
fig, ax = plt.subplots()
ax.plot(npoints,means_scores_predator, linewidth = '1', color = 'red')
ax.fill_between(npoints, (np.array(means_scores_predator)-ci_predator),
#plt.plot(npoints, means_scores_prey, linewidth = '1.2', color = 'red')
plt.xlabel('Number of points removed')
plt.ylabel('Mean Squared Error')
plt.show()
plt.close()
```

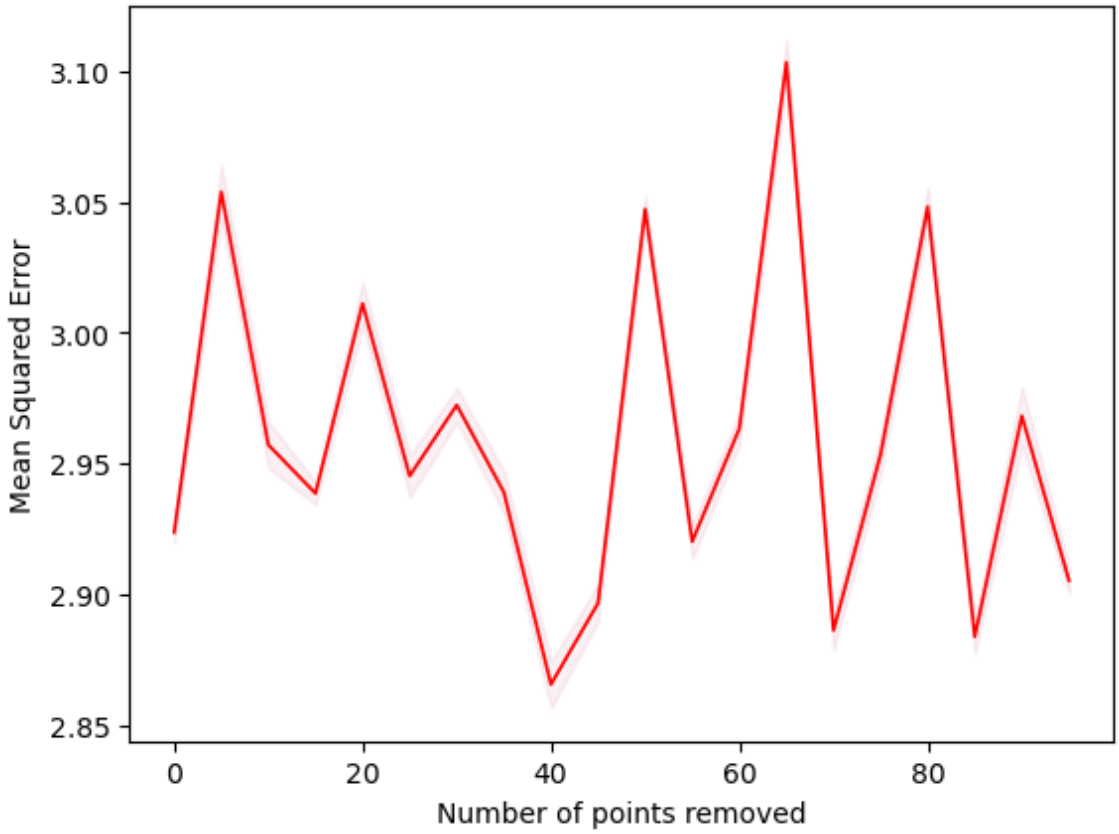
```
C:\Users\kayad\AppData\Local\Temp\ipykernel_27100\3853094899.py:38: Runtim
eWarning: overflow encountered in exp
  acceptance_probability = min(np.exp(-(delta/temp)),1)#Calculate acceptan
ce probability
C:\Users\kayad\AppData\Local\Temp\ipykernel_27100\528580857.py:11: Runtime
Warning: overflow encountered in square
  err1 = (x1[indx_x] - x2[indx_x])**2

<Figure size 1920x1440 with 0 Axes>
```


3.55
3.50
3.45
3.40
3.35
3.30
3.25
3.20
Mean Squared Error
0
20
40
60
80
Number of points removed

In [423]:

```
# Points removed with aggressive cooling
simulations = 2
iterations = 1
init_temp = 20
cooling = 0.1 # cooling strategy more aggressive
t = data[:,0]
time = t
inp = data[:,1:3]

all_scores = []
plt.figure(dpi = 300)
for counter in range(25):
    # storage
    paramets_all = []
    objective_results_all = []
    # point removal loop (removing points at steps of 2)
    npoints = np.arange(0,100,5)
    for p_removed in npoints:
        best_parameters_found_all_removal = multirun_SA_point_removal(i
        paramets_all.append(best_parameters_found_all_removal[0])
        objective_results_all.append(best_parameters_found_all_removal[
    all_scores.append(objective_results_all)
    #plt.plot(npoints ,objective_results_prey, linewidth = '0.6', color
means_scores_all = []
variance_scores_all = []
for index in range(len(npoints)):
    current_score_count_all = np.array(all_scores)[:,index]
    means_scores_all.append(np.mean(current_score_count_all))
    variance_scores_all.append(statistics.variance(current_score_count_
ci_all = 1.96 * np.array(variance_scores_all)/25
fig, ax = plt.subplots()
ax.plot(npoints,means_scores_all, linewidth = '1.2', color = 'red')
ax.fill_between(npoints, (np.array(means_scores_all)-ci_all), (np.array
#plt.plot(npoints, means_scores_prey, linewidth = '1.2', color = 'red')
plt.xlabel('Number of points removed')
plt.ylabel('Mean Squared Error')
plt.show()
plt.close()
```

```
C:\Users\kayad\AppData\Local\Temp\ipykernel_27100\3853094899.py:38: Runtim
eWarning: overflow encountered in exp
  acceptance_probability = min(np.exp(-(delta/temp)),1)#Calculate acceptan
ce probability
C:\Users\kayad\AppData\Local\Temp\ipykernel_27100\3853094899.py:38: Runtim
eWarning: overflow encountered in scalar divide
  acceptance_probability = min(np.exp(-(delta/temp)),1)#Calculate acceptan
ce probability
C:\Users\kayad\AppData\Local\Temp\ipykernel_27100\3853094899.py:38: Runtim
eWarning: divide by zero encountered in scalar divide
  acceptance_probability = min(np.exp(-(delta/temp)),1)#Calculate acceptan
ce probability

<Figure size 1920x1440 with 0 Axes>
```

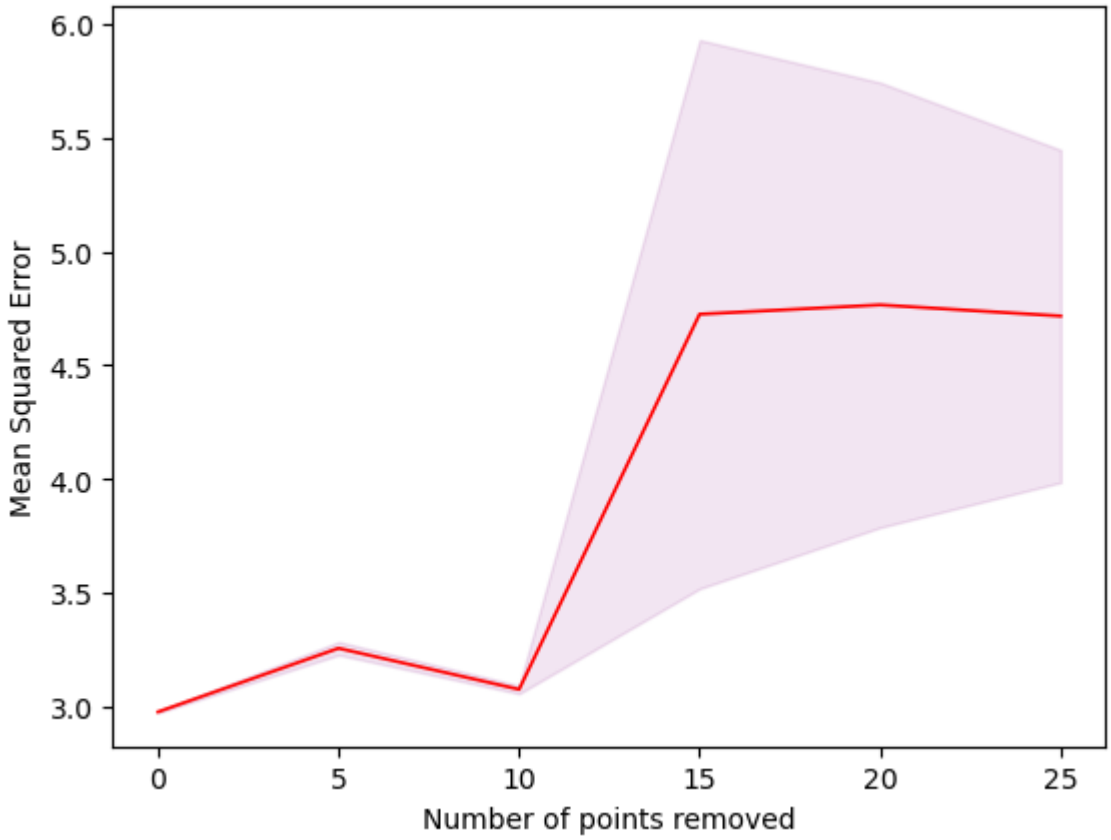

3.10
3.05
3.00
2.95
2.90
2.85
Mean Squared Error
0
20
40
60
80
Number of points removed

In [418]:

```
# Point extrema removed with aggressive cooling MSE2 function prey popu
simulations = 2
iterations = 1
init_temp = 20
cooling = 0.1 # cooling strategy more aggressive
t = data[:,0]
time = t
inp = data[:,1:3]

all_scores_prey = []
plt.figure(dpi = 300)
for counter in range(25):
    # storage
    paramets_prey = []
    objective_results_prey = []
    # point removal loop (removing points at steps of 2)
    npoints = np.arange(0,30,5)
    for p_removed in npoints:
        best_parameters_found_prey_removal = multirun_SA_point_removal(
        paramets_prey.append(best_parameters_found_prey_removal[0])
        objective_results_prey.append(best_parameters_found_prey_remova
    all_scores_prey.append(objective_results_prey)
    #plt.plot(npoints ,objective_results_prey, linewidth = '0.6', color
means_scores_prey = []
variance_scores_prey = []
for index in range(len(npoints)):
    current_score_count = np.array(all_scores_prey)[:,index]
    means_scores_prey.append(np.mean(current_score_count))
    variance_scores_prey.append(statistics.variance(current_score_count
ci = 1.96 * np.array(variance_scores_prey)/25
fig, ax = plt.subplots()
ax.plot(npoints,means_scores_prey, linewidth = '1.2', color = 'red')
ax.fill_between(npoints, (np.array(means_scores_prey)-ci), (np.array(me
#plt.plot(npoints, means_scores_prey, linewidth = '1.2', color = 'red')
plt.xlabel('Number of points removed')
plt.ylabel('Mean Squared Error')
plt.show()
plt.close()
```

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

```
<Figure size 1920x1440 with 0 Axes>
```

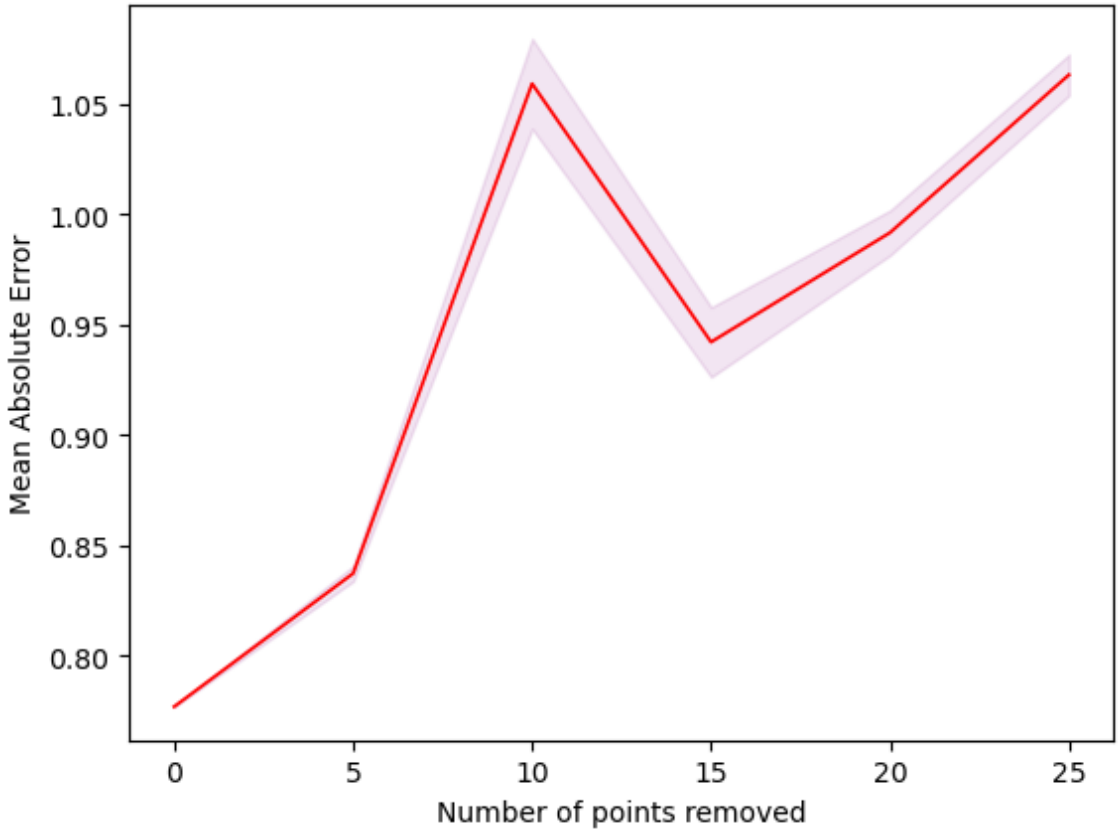


In [419]:

```
# Point extrema removed with aggressive cooling, prey population and MA
simulations = 2
iterations = 1
init_temp = 20
cooling = 0.1 # cooling strategy more aggressive
t = data[:,0]
time = t
inp = data[:,1:3]

all_scores_prey_MAE = []
for counter in range(25):
    # storage
    paramets_prey_MAE = []
    objective_results_prey_MAE = []
    # point removal loop (removing points at steps of 2)
    npoints = np.arange(0,30,5)
    for p_removed in npoints:
        best_parameters_found_prey_removal_MAE = multirun_SA_point_remo
        paramets_prey_MAE.append(best_parameters_found_prey_removal_MAE
        objective_results_prey_MAE.append(best_parameters_found_prey_re
    all_scores_prey_MAE.append(objective_results_prey_MAE)

means_scores_prey_MAE = []
variance_scores_prey_MAE = []
for index in range(len(npoints)):
    current_score_count_MAE = np.array(all_scores_prey_MAE)[:,index]
    means_scores_prey_MAE.append(np.mean(current_score_count_MAE))
    variance_scores_prey_MAE.append(statistics.variance(current_score_c
ci_MAE = 1.96 * np.array(variance_scores_prey_MAE)/25

plt.figure(dpi = 300)
fig, ax = plt.subplots()
ax.plot(npoints,means_scores_prey_MAE, linewidth = '1.2', color = 'red'
ax.fill_between(npoints, (np.array(means_scores_prey_MAE)-ci_MAE), (np.
plt.xlabel('Number of points removed')
plt.ylabel('Mean Absolute Error')
plt.show()
plt.close()
```

```
C:\Users\kayad\AppData\Local\Temp\ipykernel_27100\3853094899.py:38: Runtim
eWarning: overflow encountered in exp
  acceptance_probability = min(np.exp(-(delta/temp)),1)#Calculate acceptan
ce probability
C:\Users\kayad\AppData\Local\Temp\ipykernel_27100\3853094899.py:38: Runtim
eWarning: overflow encountered in scalar divide
  acceptance_probability = min(np.exp(-(delta/temp)),1)#Calculate acceptan
ce probability
C:\Users\kayad\AppData\Local\Temp\ipykernel_27100\3853094899.py:38: Runtim
eWarning: divide by zero encountered in scalar divide
  acceptance_probability = min(np.exp(-(delta/temp)),1)#Calculate acceptan
ce probability
C:\Users\kayad\AppData\Local\Temp\ipykernel_27100\3853094899.py:38: Runtim
eWarning: invalid value encountered in scalar divide
  acceptance_probability = min(np.exp(-(delta/temp)),1)#Calculate acceptan
ce probability
C:\Users\kayad\AppData\Local\Temp\ipykernel_27100\528580857.py:35: Runtime
Warning: Mean of empty slice
  return np.nanmean([err])
```

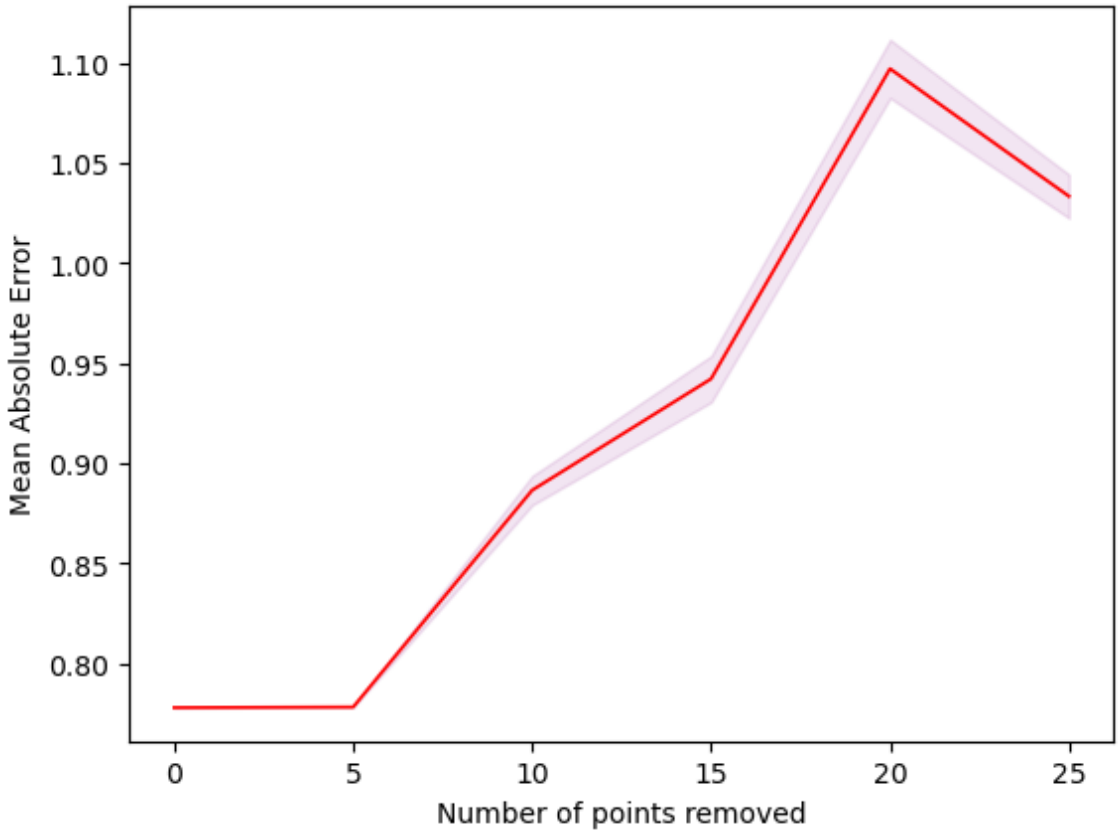
```
<Figure size 1920x1440 with 0 Axes>
```



In [420]:

```
# Point extrema removed with not aggressive cooling, prey population an
simulations = 2
iterations = 1
init_temp = 20
cooling = 0.9 # cooling strategy more aggressive
t = data[:,0]
time = t
inp = data[:,1:3]

all_scores_prey_MAE = []
for counter in range(25):
    # storage
    paramets_prey_MAE = []
    objective_results_prey_MAE = []
    # point removal loop (removing points at steps of 2)
    npoints = np.arange(0,30,5)
    for p_removed in npoints:
        best_parameters_found_prey_removal_MAE = multirun_SA_point_remo
        paramets_prey_MAE.append(best_parameters_found_prey_removal_MAE
        objective_results_prey_MAE.append(best_parameters_found_prey_re
    all_scores_prey_MAE.append(objective_results_prey_MAE)

means_scores_prey_MAE = []
variance_scores_prey_MAE = []
for index in range(len(npoints)):
    current_score_count_MAE = np.array(all_scores_prey_MAE)[:,index]
    means_scores_prey_MAE.append(np.mean(current_score_count_MAE))
    variance_scores_prey_MAE.append(statistics.variance(current_score_c
ci_MAE = 1.96 * np.array(variance_scores_prey_MAE)/25

plt.figure(dpi = 300)
fig, ax = plt.subplots()
ax.plot(npoints,means_scores_prey_MAE, linewidth = '1.2', color = 'red'
ax.fill_between(npoints, (np.array(means_scores_prey_MAE)-ci_MAE), (np.
plt.xlabel('Number of points removed')
plt.ylabel('Mean Absolute Error')
plt.show()
plt.close()
```

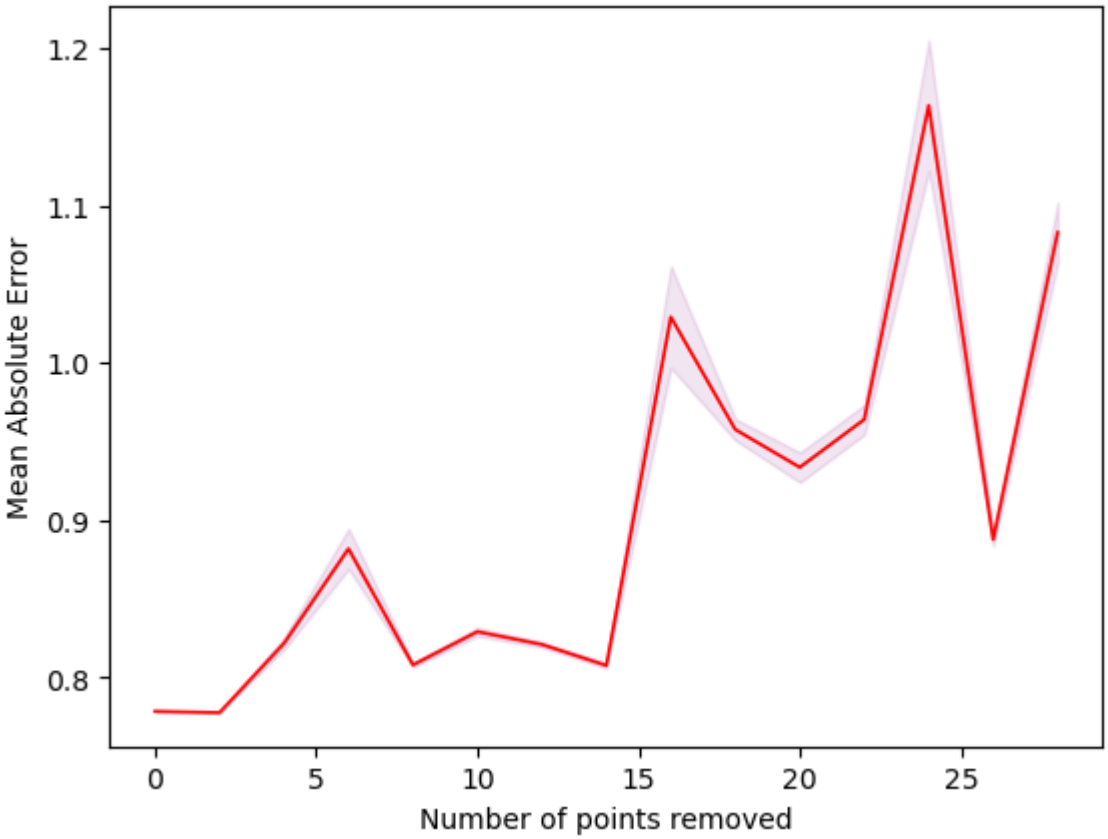
```
C:\Users\kayad\AppData\Local\Temp\ipykernel_27100\3853094899.py:38: Runtim
eWarning: overflow encountered in exp
  acceptance_probability = min(np.exp(-(delta/temp)),1)#Calculate acceptan
ce probability
C:\Users\kayad\AppData\Local\Temp\ipykernel_27100\1517521670.py:4: Runtime
Warning: overflow encountered in scalar multiply
  dxdt = (alpha * x) - (beta * x * y)  # Prey ODE
C:\Users\kayad\AppData\Local\Temp\ipykernel_27100\1517521670.py:5: Runtime
Warning: overflow encountered in scalar multiply
  dydt = (delta * x * y) - (gamma * y)  # Predator ODE
C:\Users\kayad\AppData\Local\Temp\ipykernel_27100\528580857.py:35: Runtime
Warning: Mean of empty slice
  return np.nanmean([err])

<Figure size 1920x1440 with 0 Axes>
```


1.10
1.05
1.00
0.95
0.90
0.85
0.80
Mean Absolute Error
0
5
10
15
20
25
Number of points removed

In [428]:

```
# Point extrema removed with not aggressive cooling, prey population an
simulations = 2
iterations = 1
init_temp = 20
cooling = 0.9 # cooling strategy more aggressive
t = data[:,0]
time = t
inp = data[:,1:3]

all_scores_prey_MAE = []
for counter in range(25):
    # storage
    paramets_prey_MAE = []
    objective_results_prey_MAE = []
    # point removal loop (removing points at steps of 2)
    npoints = np.arange(0,30,2)
    for p_removed in npoints:
        best_parameters_found_prey_removal_MAE = multirun_SA_point_remo
        paramets_prey_MAE.append(best_parameters_found_prey_removal_MAE
        objective_results_prey_MAE.append(best_parameters_found_prey_re
    all_scores_prey_MAE.append(objective_results_prey_MAE)

means_scores_prey_MAE = []
variance_scores_prey_MAE = []
for index in range(len(npoints)):
    current_score_count_MAE = np.array(all_scores_prey_MAE)[:,index]
    means_scores_prey_MAE.append(np.mean(current_score_count_MAE))
    variance_scores_prey_MAE.append(statistics.variance(current_score_c
ci_MAE = 1.96 * np.array(variance_scores_prey_MAE)/25

plt.figure(dpi = 300)
fig, ax = plt.subplots()
ax.plot(npoints,means_scores_prey_MAE, linewidth = '1.2', color = 'red'
ax.fill_between(npoints, (np.array(means_scores_prey_MAE)-ci_MAE), (np.
plt.xlabel('Number of points removed')
plt.ylabel('Mean Absolute Error')
plt.show()
plt.close()
```

```
C:\Users\kayad\AppData\Local\Temp\ipykernel_27100\3853094899.py:38: Runtim
eWarning: overflow encountered in exp
  acceptance_probability = min(np.exp(-(delta/temp)),1)#Calculate acceptan
ce probability
C:\Users\kayad\AppData\Local\Temp\ipykernel_27100\528580857.py:35: Runtime
Warning: Mean of empty slice
  return np.nanmean([err])
C:\Users\kayad\AppData\Local\Temp\ipykernel_27100\1517521670.py:4: Runtime
Warning: overflow encountered in scalar multiply
  dxdt = (alpha * x) - (beta * x * y)  # Prey ODE
C:\Users\kayad\AppData\Local\Temp\ipykernel_27100\1517521670.py:5: Runtime
Warning: overflow encountered in scalar multiply
  dydt = (delta * x * y) - (gamma * y)  # Predator ODE

<Figure size 1920x1440 with 0 Axes>
```

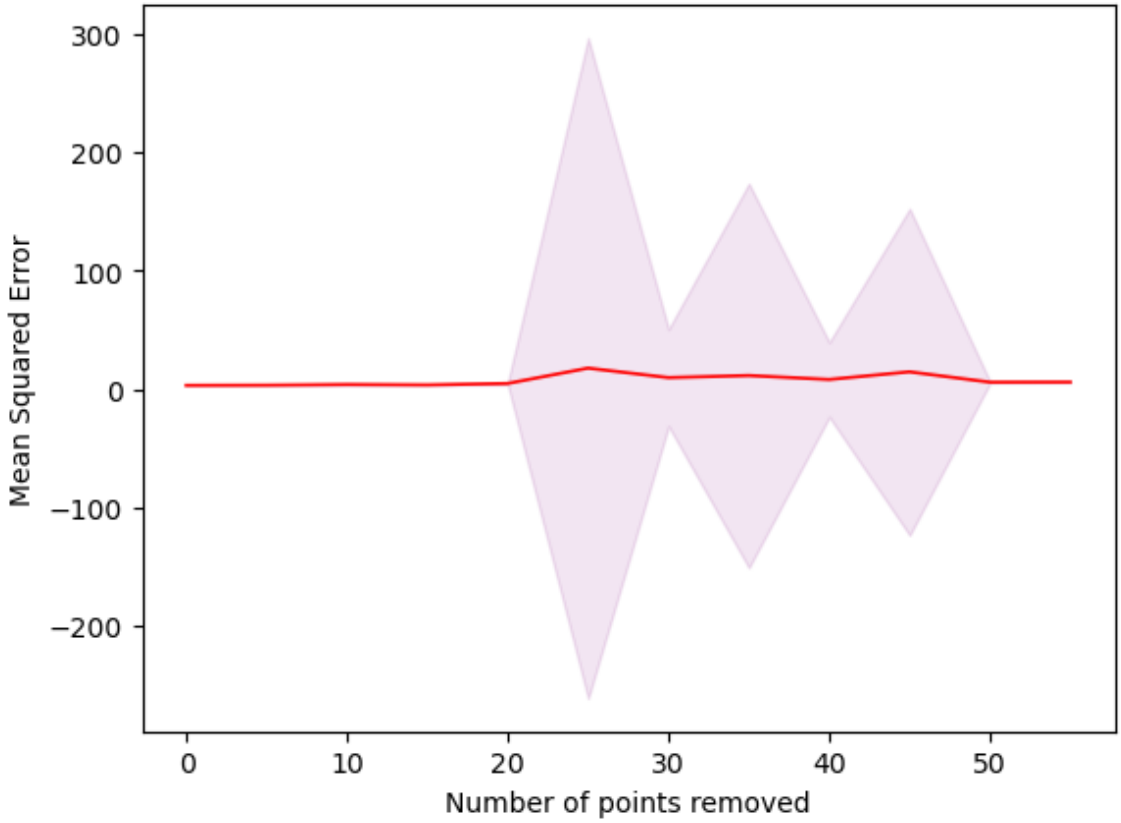

Mean Absolute Error
1.2
1.1
1.0
0.9
0.8
0
5
10
15
20
25
Number of points removed

In [421]:

```
# Point extrema removed with aggressive cooling for both species
simulations = 2
iterations = 1
init_temp = 20
cooling = 0.1 # cooling strategy more aggressive
t = data[:,0]
time = t
inp = data[:,1:3]

all_scores_predprey = []
for counter in range(25):
    # storage
    paramets_predprey = []
    objective_results_predprey = []
    # point removal loop (removing points at steps of 2)
    npoints = np.arange(0,60,5)
    for p_removed in npoints:
        best_parameters_found_predprey_removal = multirun_SA_point_remo
        paramets_predprey.append(best_parameters_found_predprey_removal
        objective_results_predprey.append(best_parameters_found_predpre
    all_scores_predprey.append(objective_results_predprey)
    #plt.plot(npoints ,objective_results_prey, linewidth = '0.6', color
means_scores_predprey = []
variance_scores_predprey = []
for index in range(len(npoints)):
    current_score_count_predprey = np.array(all_scores_predprey)[:,inde
    means_scores_predprey.append(np.mean(current_score_count_predprey))
    variance_scores_predprey.append(statistics.variance(current_score_c
ci_predprey = 1.96 * np.array(variance_scores_predprey)/15
plt.figure(dpi = 300)
fig, ax = plt.subplots()
ax.plot(npoints,means_scores_predprey, linewidth = '1.2', color = 'red'
ax.fill_between(npoints, (np.array(means_scores_predprey)-ci_predprey),
#plt.plot(npoints, means_scores_prey, linewidth = '1.2', color = 'red')
plt.xlabel('Number of points removed')
plt.ylabel('Mean Squared Error')
plt.show()
plt.close()
```

```
C:\Users\kayad\AppData\Local\Temp\ipykernel_27100\3853094899.py:38: Runtim
eWarning: overflow encountered in exp
  acceptance_probability = min(np.exp(-(delta/temp)),1)#Calculate acceptan
ce probability
C:\Users\kayad\AppData\Local\Temp\ipykernel_27100\3853094899.py:38: Runtim
eWarning: overflow encountered in scalar divide
  acceptance_probability = min(np.exp(-(delta/temp)),1)#Calculate acceptan
ce probability
C:\Users\kayad\AppData\Local\Temp\ipykernel_27100\3853094899.py:38: Runtim
eWarning: divide by zero encountered in scalar divide
  acceptance_probability = min(np.exp(-(delta/temp)),1)#Calculate acceptan
ce probability
C:\Users\kayad\AppData\Local\Temp\ipykernel_27100\3853094899.py:38: Runtim
eWarning: invalid value encountered in scalar divide
  acceptance_probability = min(np.exp(-(delta/temp)),1)#Calculate acceptan
ce probability
C:\Users\kayad\AppData\Local\Temp\ipykernel_27100\528580857.py:15: Runtime
Warning: Mean of empty slice
  return np.nanmean([err])
```

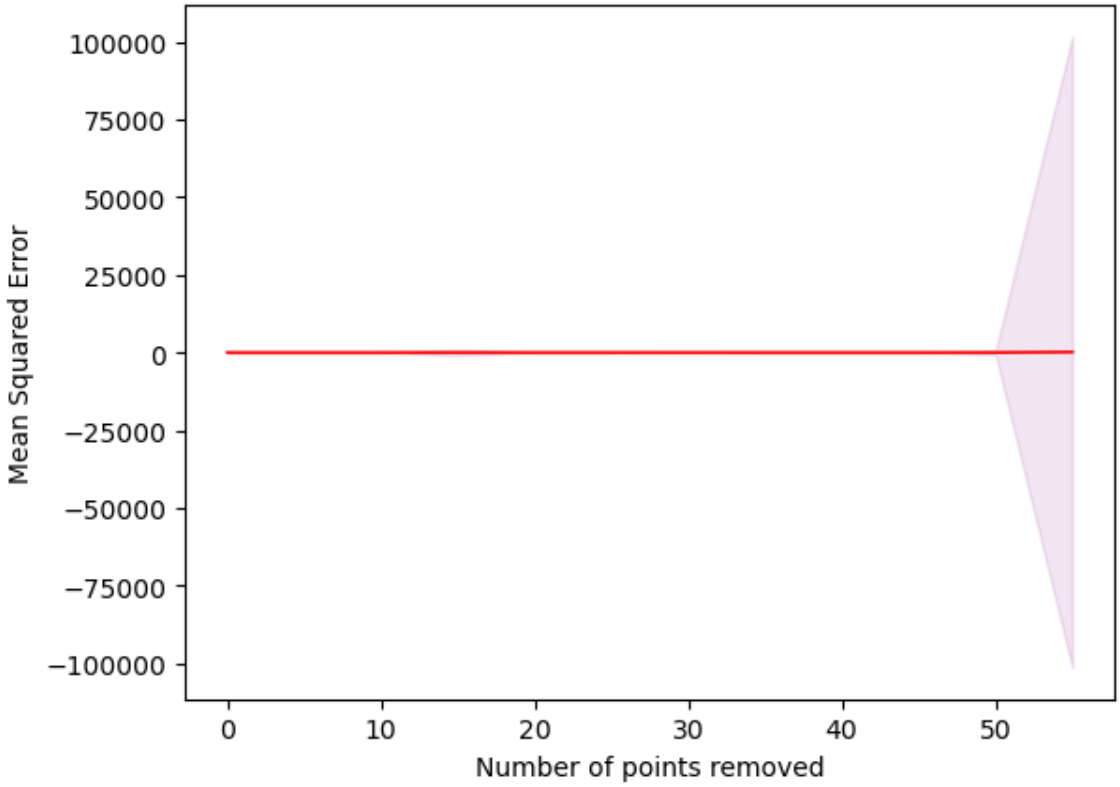
```
<Figure size 1920x1440 with 0 Axes>
```



In [422]:

```
# Point extrema removed with aggressive cooling for both species
simulations = 2
iterations = 1
init_temp = 20
cooling = 0.9 # cooling strategy more aggressive
t = data[:,0]
time = t
inp = data[:,1:3]

all_scores_predprey = []
for counter in range(25):
    # storage
    paramets_predprey = []
    objective_results_predprey = []
    # point removal loop (removing points at steps of 2)
    npoints = np.arange(0,60,5)
    for p_removed in npoints:
        best_parameters_found_predprey_removal = multirun_SA_point_remo
        paramets_predprey.append(best_parameters_found_predprey_removal
        objective_results_predprey.append(best_parameters_found_predpre
    all_scores_predprey.append(objective_results_predprey)
    #plt.plot(npoints ,objective_results_prey, linewidth = '0.6', color
means_scores_predprey = []
variance_scores_predprey = []
for index in range(len(npoints)):
    current_score_count_predprey = np.array(all_scores_predprey)[:,inde
    means_scores_predprey.append(np.mean(current_score_count_predprey))
    variance_scores_predprey.append(statistics.variance(current_score_c
ci_predprey = 1.96 * np.array(variance_scores_predprey)/15
plt.figure(dpi = 300)
fig, ax = plt.subplots()
ax.plot(npoints,means_scores_predprey, linewidth = '1.2', color = 'red'
ax.fill_between(npoints, (np.array(means_scores_predprey)-ci_predprey),
#plt.plot(npoints, means_scores_prey, linewidth = '1.2', color = 'red')
plt.xlabel('Number of points removed')
plt.ylabel('Mean Squared Error')
plt.show()
plt.close()
```

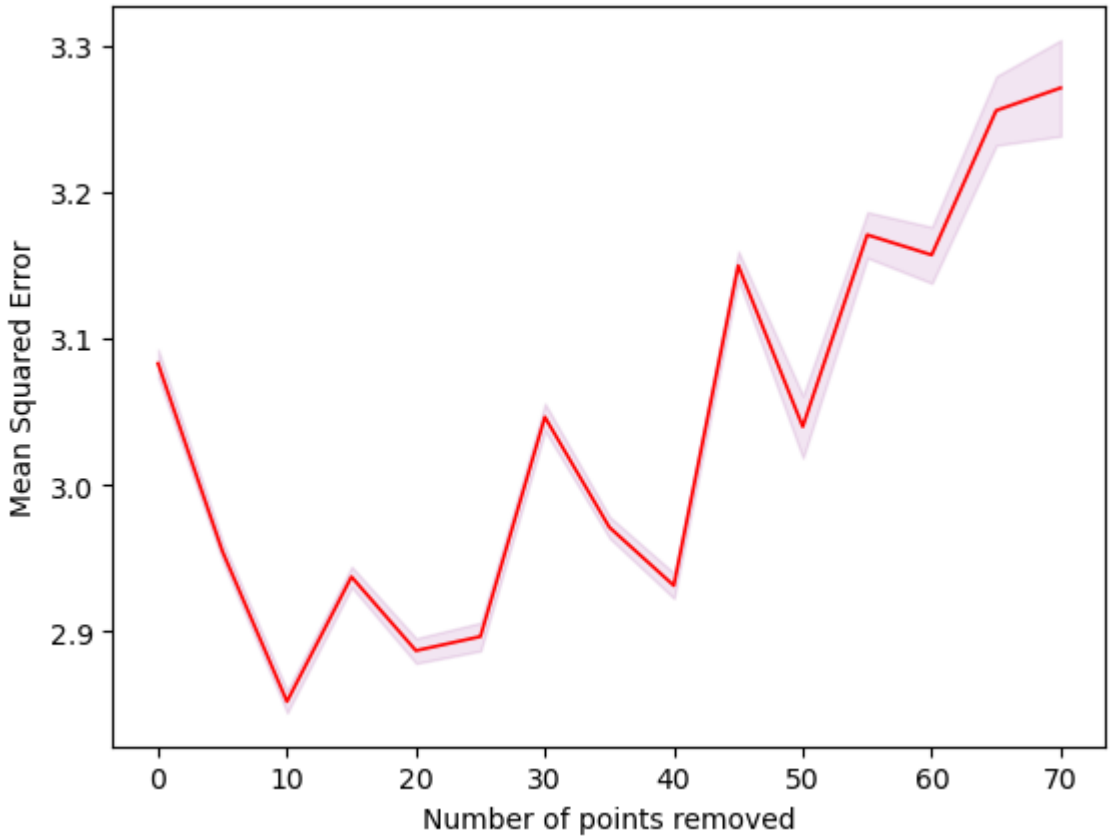
```
C:\Users\kayad\AppData\Local\Temp\ipykernel_27100\3853094899.py:38: Runtim
eWarning: overflow encountered in exp
  acceptance_probability = min(np.exp(-(delta/temp)),1)#Calculate acceptan
ce probability
C:\Users\kayad\AppData\Local\Temp\ipykernel_27100\528580857.py:15: Runtime
Warning: Mean of empty slice
  return np.nanmean([err])
C:\Users\kayad\AppData\Local\Temp\ipykernel_27100\528580857.py:11: Runtime
Warning: overflow encountered in square
  err1 = (x1[indx_x] - x2[indx_x])**2
C:\Users\kayad\AppData\Local\Temp\ipykernel_27100\528580857.py:12: Runtime
Warning: overflow encountered in square
  err2 = (y1[indx_y] - y2[indx_y])**2

<Figure size 1920x1440 with 0 Axes>
```


100000
75000
50000
25000
0
−25000
−50000
−75000
−100000
Mean Squared Error
0
10
20
30
40
50
Number of points removed

In [426]:

```
# Midpoint removal using MSE and aggressive cooling for the prey popula

simulations = 2
iterations = 1
init_temp = 20
cooling = 0.1 # cooling strategy more aggressive
t = data[:,0]
time = t
inp = data[:,1:3]

all_scores_prey_mid = []
for counter in range(25):
    # storage
    paramets_prey_mid = []
    objective_results_prey_mid = []
    # point removal loop (removing points at steps of 2)
    npoints = np.arange(0,75,5)
    for p_removed in npoints:
        best_parameters_found_prey_midremoval = multirun_SA_point_remov
        paramets_prey_mid.append(best_parameters_found_prey_midremoval[
        objective_results_prey_mid.append(best_parameters_found_prey_mi
    all_scores_prey_mid.append(objective_results_prey_mid)
means_scores_prey_mid = []
variance_scores_prey_mid = []
for index in range(len(npoints)):
    current_score_count_prey_mid = np.array(all_scores_prey_mid)[:,inde
    means_scores_prey_mid.append(np.mean(current_score_count_prey_mid))
    variance_scores_prey_mid.append(statistics.variance(current_score_c
ci_prey_mid = 1.96 * np.array(variance_scores_prey_mid)/25
plt.figure(dpi = 300)
fig, ax = plt.subplots()
ax.plot(npoints,means_scores_prey_mid, linewidth = '1.2', color = 'red'
ax.fill_between(npoints, (np.array(means_scores_prey_mid)-ci_prey_mid),
plt.xlabel('Number of points removed')
plt.ylabel('Mean Squared Error')
plt.show()
plt.close()
```

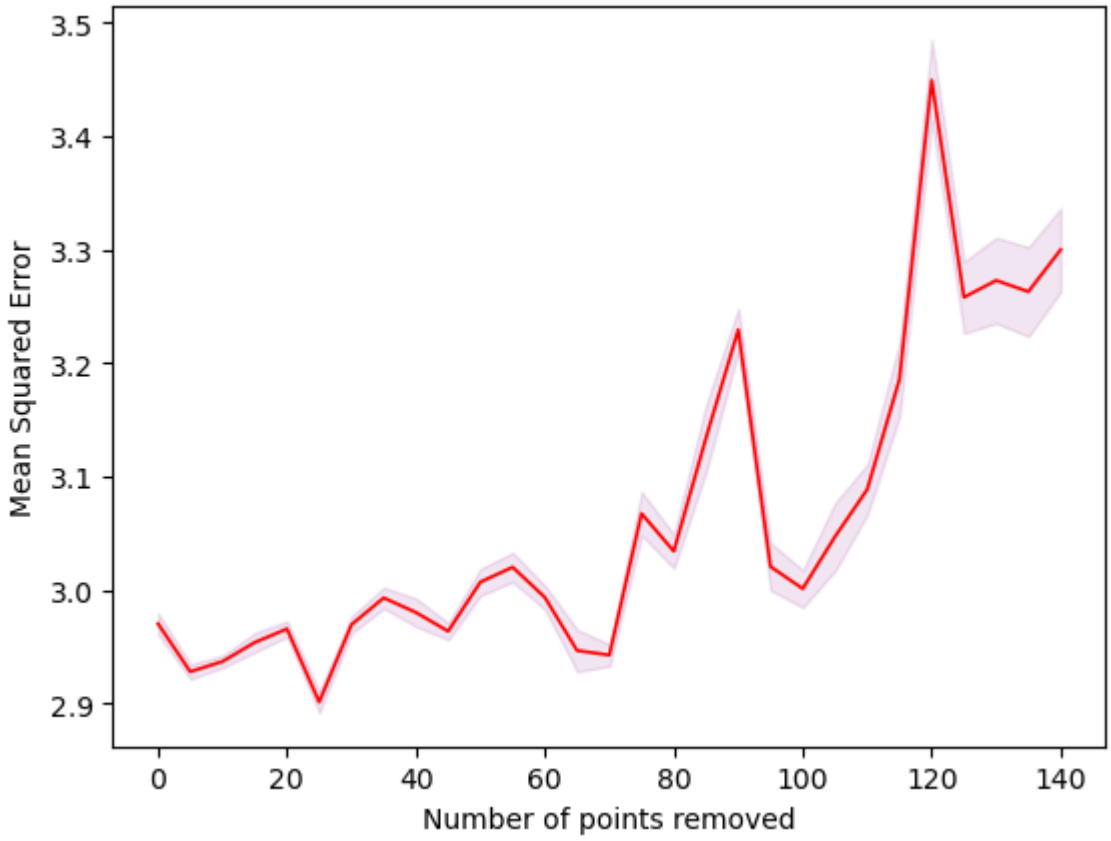
```
C:\Users\kayad\AppData\Local\Temp\ipykernel_27100\3853094899.py:38: Runtim
eWarning: overflow encountered in exp
  acceptance_probability = min(np.exp(-(delta/temp)),1)#Calculate acceptan
ce probability
C:\Users\kayad\AppData\Local\Temp\ipykernel_27100\3853094899.py:38: Runtim
eWarning: overflow encountered in scalar divide
  acceptance_probability = min(np.exp(-(delta/temp)),1)#Calculate acceptan
ce probability
C:\Users\kayad\AppData\Local\Temp\ipykernel_27100\3853094899.py:38: Runtim
eWarning: divide by zero encountered in scalar divide
  acceptance_probability = min(np.exp(-(delta/temp)),1)#Calculate acceptan
ce probability

<Figure size 1920x1440 with 0 Axes>
```


3.3
3.2
3.1
3.0
2.9
Mean Squared Error
0
10
20
30
40
50
60
70
Number of points removed

In [427]:

```
# Midpoint removal using MSE and not aggressive cooling for both popula

simulations = 2
iterations = 1
init_temp = 20
cooling = 0.1 # cooling strategy more aggressive
t = data[:,0]
time = t
inp = data[:,1:3]

all_scores_predprey_mid = []
for counter in range(25):
    # storage
    paramets_predprey_mid = []
    objective_results_predprey_mid = []
    # point removal loop
    npoints = np.arange(0,145,5)
    for p_removed in npoints:
        best_parameters_found_predprey_midremoval = multirun_SA_point_r
        paramets_predprey_mid.append(best_parameters_found_predprey_mid
        objective_results_predprey_mid.append(best_parameters_found_pre
    all_scores_predprey_mid.append(objective_results_predprey_mid)
means_scores_predprey_mid = []
variance_scores_predprey_mid = []
for index in range(len(npoints)):
    current_score_count_predprey_mid = np.array(all_scores_predprey_mid
    means_scores_predprey_mid.append(np.mean(current_score_count_predpr
    variance_scores_predprey_mid.append(statistics.variance(current_sco
ci_predprey_mid = 1.96 * np.array(variance_scores_predprey_mid)/25
plt.figure(dpi = 300)
fig, ax = plt.subplots()
ax.plot(npoints,means_scores_predprey_mid, linewidth = '1.2', color = '
ax.fill_between(npoints, (np.array(means_scores_predprey_mid)-ci_predpr
plt.xlabel('Number of points removed')
plt.ylabel('Mean Squared Error')
plt.show()
plt.close()
```

```
C:\Users\kayad\AppData\Local\Temp\ipykernel_27100\3853094899.py:38: Runtim
eWarning: overflow encountered in exp
  acceptance_probability = min(np.exp(-(delta/temp)),1)#Calculate acceptan
ce probability
C:\Users\kayad\AppData\Local\Temp\ipykernel_27100\3853094899.py:38: Runtim
eWarning: overflow encountered in scalar divide
  acceptance_probability = min(np.exp(-(delta/temp)),1)#Calculate acceptan
ce probability
C:\Users\kayad\AppData\Local\Temp\ipykernel_27100\3853094899.py:38: Runtim
eWarning: divide by zero encountered in scalar divide
  acceptance_probability = min(np.exp(-(delta/temp)),1)#Calculate acceptan
ce probability

<Figure size 1920x1440 with 0 Axes>
```


Mean Squared Error
3.5
3.4
3.3
3.2
3.1
3.0
2.9
0
20
40
60
80
100
120
140
Number of points removed

In [457]:

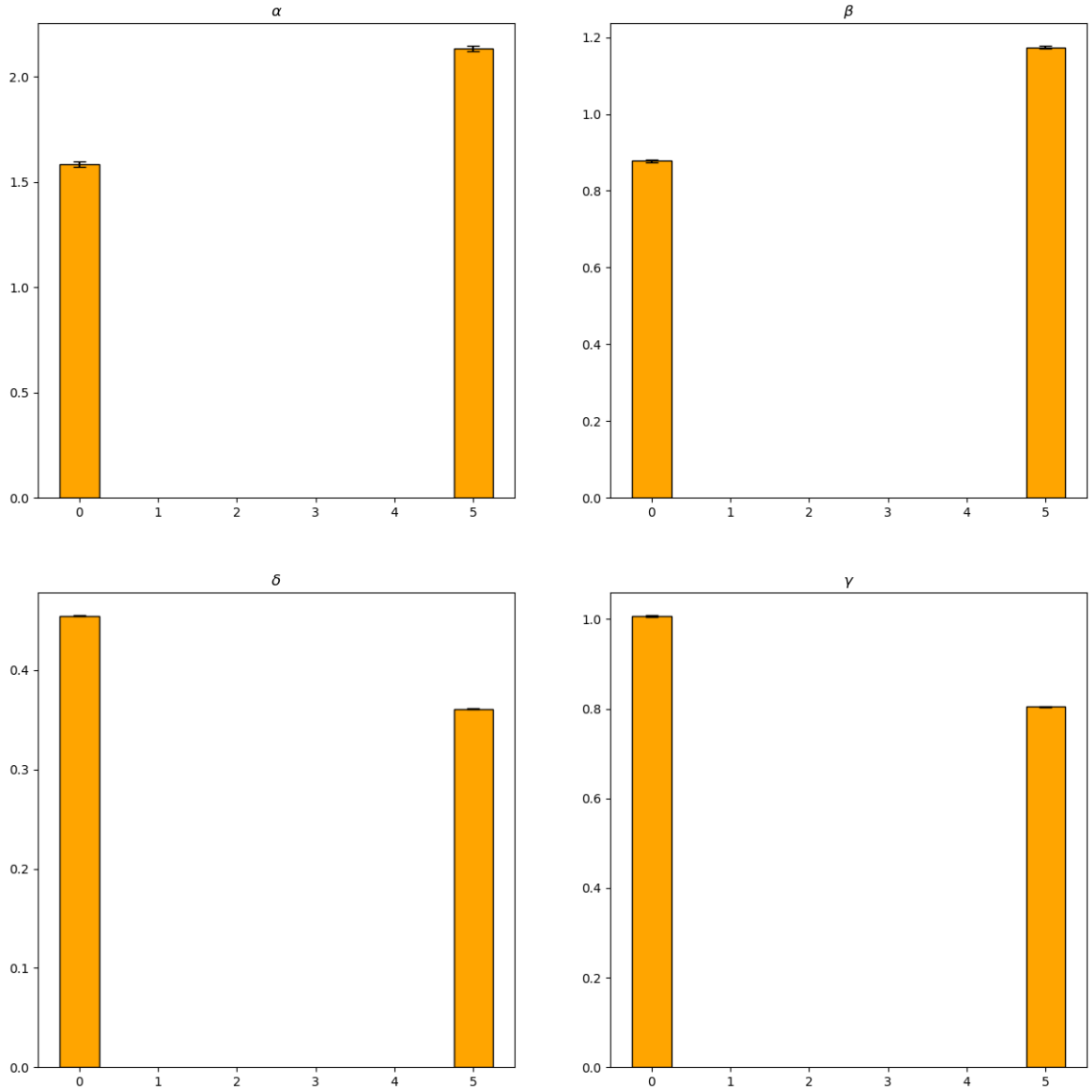
```
# Point extrema removed with aggressive cooling MSE2 function prey popu

simulations = 2
iterations = 1
init_temp = 20
cooling = 0.1 # cooling strategy more aggressive
t = data[:,0]
time = t
inp = data[:,1:3]

all_scores_prey = []
all_paramets_prey = []
for counter in range(25):
    # storage
    paramets_prey = []
    objective_results_prey = []
    # point removal loop
    npoints = np.arange(0,30,5)
    for p_removed in npoints:
        best_parameters_found_prey_removal = multirun_SA_point_removal(
        paramets_prey.append(best_parameters_found_prey_removal[0])
        objective_results_prey.append(best_parameters_found_prey_remova
    all_scores_prey.append(objective_results_prey)
    all_paramets_prey.append(paramets_prey)
    #plt.plot(npoints ,objective_results_prey, linewidth = '0.6', color
means_paramets_prey = []


for index in range(len(npoints)):
    current_paramets_prey = np.array(all_paramets_prey)[:,index]
    means_paramets_prey.append([np.mean(current_paramets_prey[:,0]), np

alpha_all_means = np.array(means_paramets_prey)[:,0]
beta_all_means = np.array(means_paramets_prey)[:,1]
delta_all_means = np.array(means_paramets_prey)[:,2]
gamma_all_means = np.array(means_paramets_prey)[:,3]

variance_alpha_prey = statistics.variance(alpha_all_means)
variance_beta_prey = statistics.variance(beta_all_means)
variance_delta_prey = statistics.variance(delta_all_means)
variance_gamma_prey = statistics.variance(gamma_all_means)

ci_alpha_prey = 1.96 * np.array(variance_alpha_prey)/25
ci_beta_prey = 1.96 * np.array(variance_beta_prey)/25
ci_delta_prey = 1.96 * np.array(variance_delta_prey)/25
ci_gamma_prey = 1.96 *np.array(variance_gamma_prey)/25

print(beta_all_means, npoints)
plt.figure(dpi = 300)
fig, axes = plt.subplots(2, 2, figsize=(15, 15))

bar_width = 0.5
plt.subplot(221)
plt.title(r'$\alpha$')
plt.bar(npoints, alpha_all_means, yerr = ci_alpha_prey,color = "orange"
plt.subplot(222)
plt.title(r'$\beta$')
plt.bar(npoints, beta_all_means, yerr = ci_beta_prey,color = "orange",e
plt.subplot(223)
plt.title(r'$\delta$')
plt.bar(npoints, delta_all_means, yerr = ci_delta_prey,color = "orange"
```

```
plt.subplot(224)
plt.title(r'$\gamma$')
plt.bar(npoints, gamma_all_means, yerr = ci_gamma_prey,color = "orange"
```

```
C:\Users\kayad\AppData\Local\Temp\ipykernel_27100\3853094899.py:38: Runtim
eWarning: overflow encountered in exp
  acceptance_probability = min(np.exp(-(delta/temp)),1)#Calculate acceptan
ce probability
C:\Users\kayad\AppData\Local\Temp\ipykernel_27100\3853094899.py:38: Runtim
eWarning: overflow encountered in scalar divide
  acceptance_probability = min(np.exp(-(delta/temp)),1)#Calculate acceptan
ce probability
C:\Users\kayad\AppData\Local\Temp\ipykernel_27100\3853094899.py:38: Runtim
eWarning: divide by zero encountered in scalar divide
  acceptance_probability = min(np.exp(-(delta/temp)),1)#Calculate acceptan
ce probability

[0.87772397 1.17375944] [0 5]
```

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

<Figure size 1920x1440 with 0 Axes>



In [ ]:

```
# Point extrema removed with aggressive cooling MSE2 function both popu

simulations = 2
iterations = 1
init_temp = 20
cooling = 0.1 # cooling strategy more aggressive
t = data[:,0]
time = t
inp = data[:,1:3]

all_scores_prey = []
all_paramets_prey = []
for counter in range(25):
    # storage
    paramets_prey = []
    objective_results_prey = []
    # point removal loop
    npoints = np.arange(0,60,5)
    for p_removed in npoints:
        best_parameters_found_prey_removal = multirun_SA_point_removal(
        paramets_prey.append(best_parameters_found_prey_removal[0])
        objective_results_prey.append(best_parameters_found_prey_remova
    all_scores_prey.append(objective_results_prey)
    all_paramets_prey.append(paramets_prey)
means_paramets_prey = []


for index in range(len(npoints)):
    current_paramets_prey = np.array(all_paramets_prey)[:,index]
    means_paramets_prey.append([np.mean(current_paramets_prey[:,0]), np

alpha_all_means = np.array(means_paramets_prey)[:,0]
beta_all_means = np.array(means_paramets_prey)[:,1]
delta_all_means = np.array(means_paramets_prey)[:,2]
gamma_all_means = np.array(means_paramets_prey)[:,3]

variance_alpha_prey = statistics.variance(alpha_all_means)
variance_beta_prey = statistics.variance(beta_all_means)
variance_delta_prey = statistics.variance(delta_all_means)
variance_gamma_prey = statistics.variance(gamma_all_means)

ci_alpha_prey = 1.96 * np.array(variance_alpha_prey)/25
ci_beta_prey = 1.96 * np.array(variance_beta_prey)/25
ci_delta_prey = 1.96 * np.array(variance_delta_prey)/25
ci_gamma_prey = 1.96 *np.array(variance_gamma_prey)/25

print(beta_all_means, npoints)
plt.figure(dpi = 300)
fig, axes = plt.subplots(2, 2, figsize=(15, 15))

bar_width = 0.5
plt.subplot(221)
plt.title(r'$\alpha$')
plt.bar(npoints, alpha_all_means, yerr = ci_alpha_prey,color = "orange"
plt.subplot(222)
plt.title(r'$\beta$')
plt.bar(npoints, beta_all_means, yerr = ci_beta_prey,color = "orange",e
plt.subplot(223)
plt.title(r'$\delta$')
plt.bar(npoints, delta_all_means, yerr = ci_delta_prey,color = "orange"
plt.subplot(224)
```

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

In [458]:

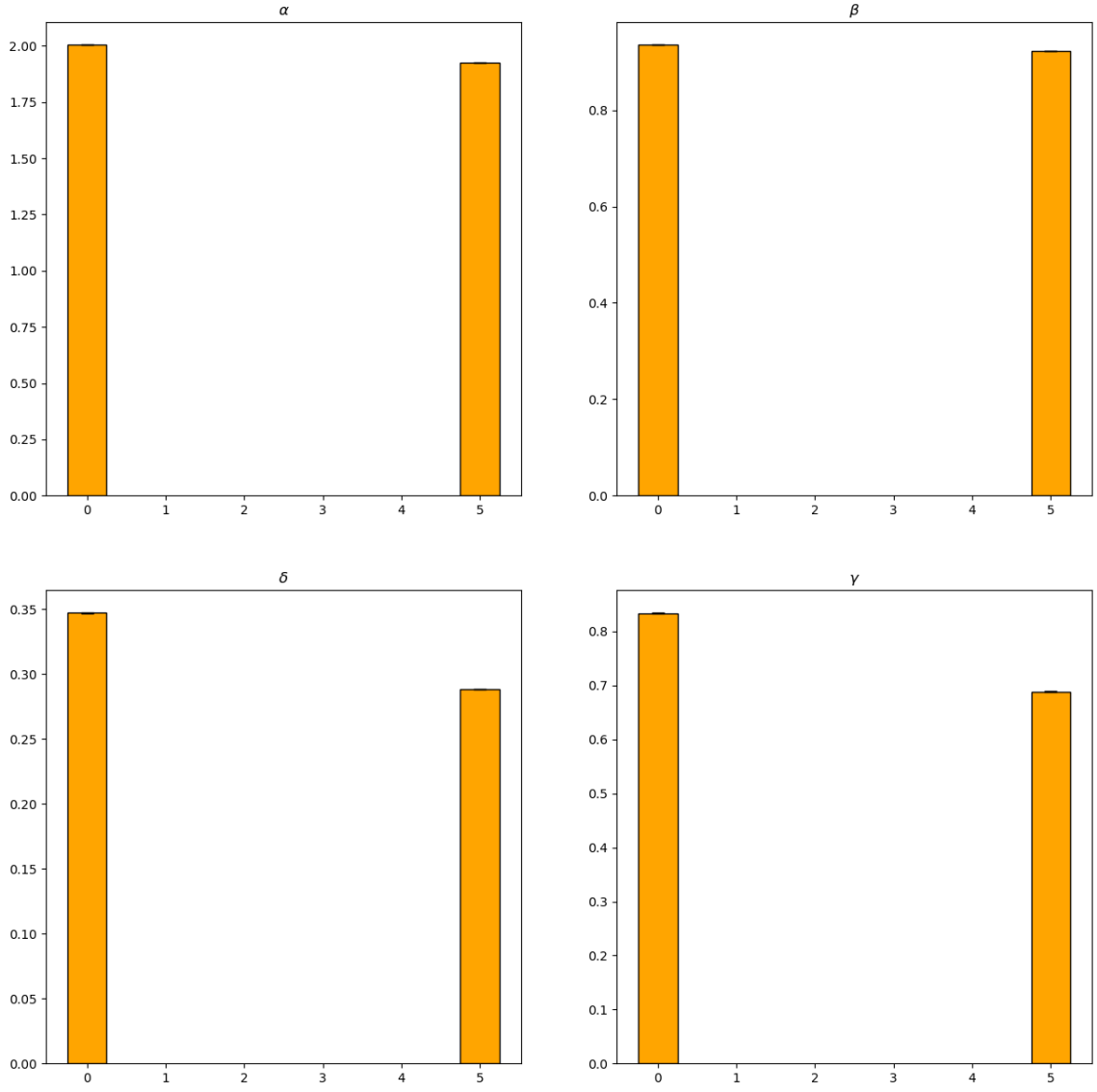
```
# Points removed randomly with aggressive cooling MSE2 function both pc

simulations = 2
iterations = 1
init_temp = 20
cooling = 0.1 # cooling strategy more aggressive
t = data[:,0]
time = t
inp = data[:,1:3]

all_scores = []
all_paramets = []
for counter in range(25):
    # storage
    paramets = []
    objective_results = []
    # point removal loop
    npoints = np.arange(0,100,5)
    for p_removed in npoints:
        best_parameters_found_removal = multirun_SA_point_removal(itera
        paramets.append(best_parameters_found_removal[0])
        objective_results.append(best_parameters_found_removal[1])
    all_scores.append(objective_results)
    all_paramets.append(paramets)
means_paramets = []


for index in range(len(npoints)):
    current_paramets = np.array(all_paramets)[:,index]
    means_paramets.append([np.mean(current_paramets[:,0]), np.mean(curr

alpha_all_means = np.array(means_paramets)[:,0]
beta_all_means = np.array(means_paramets)[:,1]
delta_all_means = np.array(means_paramets)[:,2]
gamma_all_means = np.array(means_paramets)[:,3]

variance_alpha = statistics.variance(alpha_all_means)
variance_beta = statistics.variance(beta_all_means)
variance_delta = statistics.variance(delta_all_means)
variance_gamma = statistics.variance(gamma_all_means)

ci_alpha = 1.96 * np.array(variance_alpha)/25
ci_beta = 1.96 * np.array(variance_beta)/25
ci_delta = 1.96 * np.array(variance_delta)/25
ci_gamma = 1.96 *np.array(variance_gamma)/25

print(beta_all_means, npoints)
plt.figure(dpi = 300)
fig, axes = plt.subplots(2, 2, figsize=(15, 15))

bar_width = 0.5
plt.subplot(221)
plt.title(r'$\alpha$')
plt.bar(npoints, alpha_all_means, yerr = ci_alpha,color = "orange",ec =
plt.subplot(222)
plt.title(r'$\beta$')
plt.bar(npoints, beta_all_means, yerr = ci_beta,color = "orange",ec = '
plt.subplot(223)
plt.title(r'$\delta$')
plt.bar(npoints, delta_all_means, yerr = ci_delta,color = "orange",ec =
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: Runtim
eWarning: overflow encountered in exp
  acceptance_probability = min(np.exp(-(delta/temp)),1)#Calculate acceptan
ce probability
C:\Users\kayad\AppData\Local\Temp\ipykernel_27100\3853094899.py:38: Runtim
eWarning: overflow encountered in scalar divide
  acceptance_probability = min(np.exp(-(delta/temp)),1)#Calculate acceptan
ce probability
C:\Users\kayad\AppData\Local\Temp\ipykernel_27100\3853094899.py:38: Runtim
eWarning: divide by zero encountered in scalar divide
  acceptance_probability = min(np.exp(-(delta/temp)),1)#Calculate acceptan
ce probability

[0.93545137 0.92197714] [0 5]
```

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

<Figure size 1920x1440 with 0 Axes>



In [ ]:

```
# Random point removed with aggressive cooling MSE2 function prey popul

simulations = 2
iterations = 1
init_temp = 20
cooling = 0.1 # cooling strategy more aggressive
t = data[:,0]
time = t
inp = data[:,1:3]

all_scores_prey = []
all_paramets_prey = []
for counter in range(25):
    # storage
    paramets_prey = []
    objective_results_prey = []
    # point removal loop
    npoints = np.arange(0,100,5)
    for p_removed in npoints:
        best_parameters_found_prey_removal = multirun_SA_point_removal(
        paramets_prey.append(best_parameters_found_prey_removal[0])
        objective_results_prey.append(best_parameters_found_prey_remova
    all_scores_prey.append(objective_results_prey)
    all_paramets_prey.append(paramets_prey)
means_paramets_prey = []


for index in range(len(npoints)):
    current_paramets_prey = np.array(all_paramets_prey)[:,index]
    means_paramets_prey.append([np.mean(current_paramets_prey[:,0]), np

alpha_all_means = np.array(means_paramets_prey)[:,0]
beta_all_means = np.array(means_paramets_prey)[:,1]
delta_all_means = np.array(means_paramets_prey)[:,2]
gamma_all_means = np.array(means_paramets_prey)[:,3]

variance_alpha_prey = statistics.variance(alpha_all_means)
variance_beta_prey = statistics.variance(beta_all_means)
variance_delta_prey = statistics.variance(delta_all_means)
variance_gamma_prey = statistics.variance(gamma_all_means)

ci_alpha_prey = 1.96 * np.array(variance_alpha_prey)/25
ci_beta_prey = 1.96 * np.array(variance_beta_prey)/25
ci_delta_prey = 1.96 * np.array(variance_delta_prey)/25
ci_gamma_prey = 1.96 *np.array(variance_gamma_prey)/25

print(beta_all_means, npoints)
plt.figure(dpi = 300)
fig, axes = plt.subplots(2, 2, figsize=(15, 15))

bar_width = 0.5
plt.subplot(221)
plt.title(r'$\alpha$')
plt.bar(npoints, alpha_all_means, yerr = ci_alpha_prey,color = "orange"
plt.subplot(222)
plt.title(r'$\beta$')
plt.bar(npoints, beta_all_means, yerr = ci_beta_prey,color = "orange",e
plt.subplot(223)
plt.title(r'$\delta$')
plt.bar(npoints, delta_all_means, yerr = ci_delta_prey,color = "orange"
plt.subplot(224)
```

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

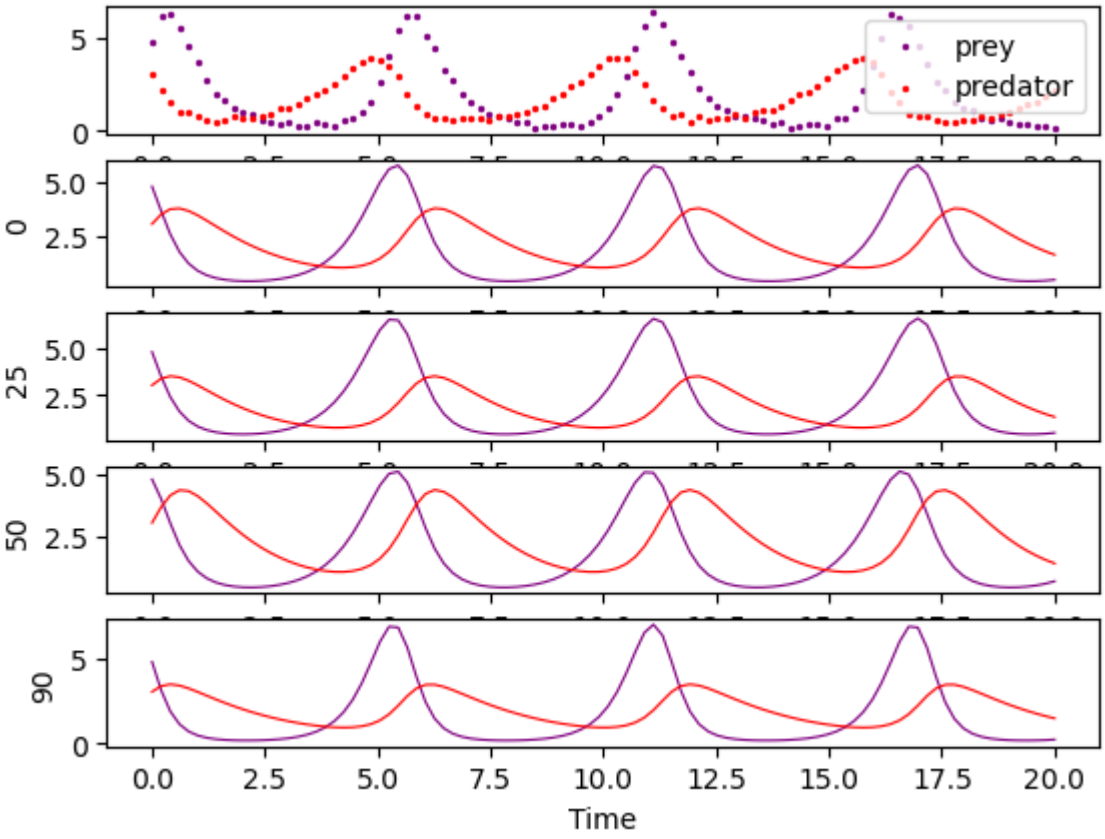
In [14]:

```
def MSE2(actual, predicted):
    '''Mean squared error'''
    x1 = actual[:, 0]
    y1 = actual[:, 1]

    #Getting useful indexes
    indx_x = np.where(~np.isnan(x1))
    indx_y = np.where(~np.isnan(y1))

    x2, y2 = predicted[:, 0], predicted[:, 1]
    err1 = (x1[indx_x] - x2[indx_x])**2
    err2 = (y1[indx_y] - y2[indx_y])**2
    err = np.concatenate([err1, err2])

    return np.nanmean(err)
```

In [20]:

```
simulations = 2
iterations = 1
init_temp = 20
cooling = 0.1 # cooling strategy more aggressive
t = data[:,0]
time = t
inp = data[:,1:3]
cooling_constant = 0.10
parameters =  [np.random.uniform(0,1), np.random.uniform(0,1), np.randc

Result_full = inferance_removal(initial_temp,cooling_constant, MSE2, si
Result_quarter = inferance_removal(initial_temp,cooling_constant, MSE2,
Result_half = inferance_removal(initial_temp,cooling_constant, MSE2, si
Result_90 = inferance_removal(initial_temp,cooling_constant, MSE2, simu

Integration_full = predator_prey_integration(time,inp[0],Result_full[0]
Integration_quarter = predator_prey_integration(time,inp[0],Result_quar
Integration_half = predator_prey_integration(time,inp[0],Result_half[0]
Integration_90 = predator_prey_integration(time,inp[0],Result_90[0])

plt.plot(dpi = 300, figsize = (20, 5))
plt.subplot(511)
plt.scatter(time, inp.T[0], color = 'purple', s = 2, label = 'prey')
plt.scatter(time, inp.T[1], color = 'red', s = 2, label = 'predator')
plt.legend()
plt.subplot(512)
plt.plot(time, Integration_full[:,0], color = 'purple', label = 'prey',
plt.plot(time, Integration_full[:,1], color = 'red', label = 'predator'
plt.ylabel('0')
plt.subplot(513)
plt.plot(time, Integration_quarter[:,0], color = 'purple', label = 'pre
plt.plot(time, Integration_quarter[:,1], color = 'red', label = 'predat
plt.ylabel('25')
plt.subplot(514)
plt.plot(time, Integration_half[:,0], color = 'purple', label = 'prey',
plt.plot(time, Integration_half[:,1], color = 'red', label = 'predator'
plt.ylabel('50')
plt.subplot(515)
plt.plot(time, Integration_90[:,0], color = 'purple', label = 'prey', l
plt.plot(time, Integration_90[:,1], color = 'red', label = 'predator',
plt.ylabel('90')
plt.xlabel('Time')
plt.show()
plt.close()

```

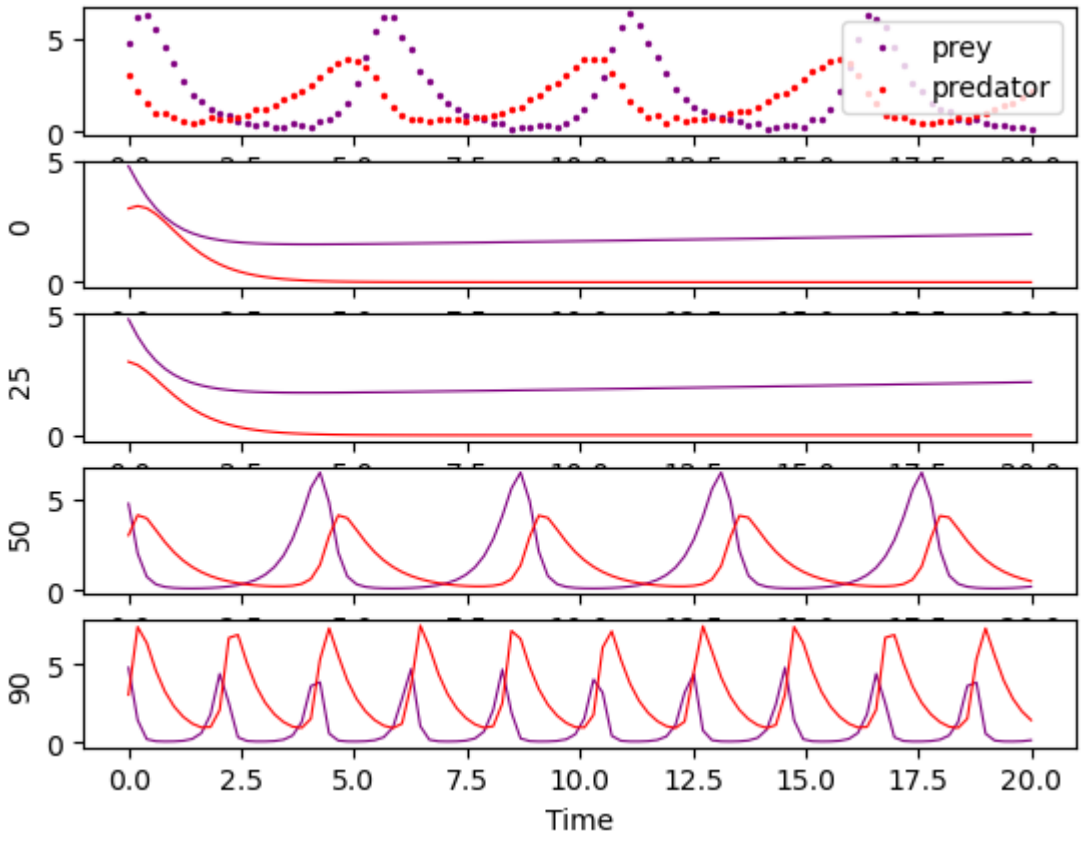
```
C:\Users\Aleks\AppData\Local\Temp\ipykernel_1540\1335953606.py:38: Runtime
Warning: overflow encountered in exp
  acceptance_probability = min(np.exp(-(delta/temp)),1)#Calculate acceptan
ce probability
C:\Users\Aleks\AppData\Local\Temp\ipykernel_1540\1335953606.py:38: Runtime
Warning: overflow encountered 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]:

```
simulations = 2
iterations = 1
init_temp = 20
cooling = 0.1 # cooling strategy more aggressive
t = data[:,0]
time = t
inp = data[:,1:3]
cooling_constant = 0.10
parameters =  [np.random.uniform(0,1), np.random.uniform(0,1), np.rando

Result_full = inferance_removal(initial_temp,cooling_constant, MSE2, si
Result_quarter = inferance_removal(initial_temp,cooling_constant, MSE2,
Result_half = inferance_removal(initial_temp,cooling_constant, MSE2, si
Result_90 = inferance_removal(initial_temp,cooling_constant, MSE2, simu

Integration_full = predator_prey_integration(time,inp[0],Result_full[0]
Integration_quarter = predator_prey_integration(time,inp[0],Result_quar
Integration_half = predator_prey_integration(time,inp[0],Result_half[0]
Integration_90 = predator_prey_integration(time,inp[0],Result_90[0])

plt.plot(dpi = 300, figsize = (20, 5))
plt.subplot(511)
plt.scatter(time, inp.T[0], color = 'purple', s = 2, label = 'prey')
plt.scatter(time, inp.T[1], color = 'red', s = 2, label = 'predator')
plt.legend()
plt.subplot(512)
plt.plot(time, Integration_full[:,0], color = 'purple', label = 'prey',
plt.plot(time, Integration_full[:,1], color = 'red', label = 'predator'
plt.ylabel('0')
plt.subplot(513)
plt.plot(time, Integration_quarter[:,0], color = 'purple', label = 'pre
plt.plot(time, Integration_quarter[:,1], color = 'red', label = 'predat
plt.ylabel('25')
plt.subplot(514)
plt.plot(time, Integration_half[:,0], color = 'purple', label = 'prey',
plt.plot(time, Integration_half[:,1], color = 'red', label = 'predator'
plt.ylabel('50')
plt.subplot(515)
plt.plot(time, Integration_90[:,0], color = 'purple', label = 'prey', l
plt.plot(time, Integration_90[:,1], color = 'red', label = 'predator',
plt.ylabel('90')
plt.xlabel('Time')
plt.show()
plt.close()

```

```
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: Matplotl
ibDeprecationWarning: 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
```