Error in the Fish Logistic Model

Aaron Lee (2111646), Aidan Phelan (2036213), Noah McMahon (2164515)

December 8, 2023

1 Abstract

Our group wanted to analyze the accuracy of the logistic model against real-life fish populations over time spanning across 257 different fish populations, along with five other models: a linear, 4th degree polynomial, cubic spline, logistic-linear, and logistic-sin fit. We used the root mean squared error (RMSE) to determine the accuracy of each fit and found that all six fits were relatively accurate. Specifically, we found that the cubic spline to be the most accurate, the linear curve to be the least accurate accurate. Furthermore, the logistic model was the second-worst fit among the six.

2 Introduction

We learned about the logistic model and how it pertains to fish populations in class. We also looked at fitting cubic and other functions to data.

The logistic function that we used in class uses a few assumptions in order to make those calculations easier. One of the assumptions would be that the fish population starts or ends at zero. This is not true in reality, as most data measuring fish populations do not start at the inception of a fish population. For example, the American Plaice Southern Gulf of St. Lawrence fish species has their population start at 8×10^9 and decreases to 1×10^1, which is radically different than the previous assumption as the population decreases as opposed to logistically increasing.

Another assumption that we made would be assuming that the data does not waver from the logistic model. This is immediately proven false after looking at even one of the actual fish population datasets, as most of the fish populations fluctuate up and down instead of directly following the logistic model.

No longer using these assumptions, we now consider how accurate the logistic model is, as well as if there are better fits to real fish data, such as cubic, polynomial, and linear fits.

3 Data Sources

The data source that we used in our project is Our World in Data Fish Stocks, which is linked in the appendices. This dataset was collected by Our World in Data, which was founded by Max Roser. Our World in Data's mission is to make the knowledge on large problems easily accessible and understandable, usually in interactive graphs and plots.

This fish population dataset contains a variety of fields, ranging from total catch to biomass to entity to year, as well as all of the fields as a ratio to constants such as maximum sustained yield. In this project, we are only using the entity, year, and total number columns. The entity column displays the fish population name, the year column shows the time for each fish population, and the total number column displays the total population of a certain fish species at a certain given time from the year column.

4 Methods

Before even starting the fitting process, we need to first normalize our data. If not, the extremely large fish population values would force the code to time out. To do this, we applied the function $\frac{x_i - x_{min}}{x_{max} - x_{min}}$ to every

data point on both axis.

The six types of fits we attempted were the logistic, logistic-linear, logistic-sin, linear, 4th degree polynomial, and lastly a cubic spline fit. The following equations were used to fit the data:

$$\frac{K-b}{1+\exp(-r(x-a))}+b \qquad \text{Logistic}$$

$$\frac{d(x-1)+K-(cx+b)}{1+\exp(-r(x-a))}+(cx+b) \qquad \text{Logistic-linear}$$

$$\frac{K+a_1 sin(f_1 x)-b-a_2 sin(f_2 x)}{1+\exp(-r(x-a))}+b+a_2 sin(f_2 x) \qquad \text{Logistic-sin}$$

$$mx+b \qquad \text{linear (slope intercept)}$$

$$b_4x^4+b_3x^3+b_2x^2+b_1x+b_0 \qquad \text{4th degree polynomial}$$

For characterizing the logistic fits and for picking parameter boundaries this Desmos calculator was used: `https://www.desmos.com/calculator/mkg6xrlpjg`
The fits were computed with numpy and scipy fitting packages, and all the logistic fits were given boundaries on the parameters to assist the function in coming to a good solution. For any of the logistic fits where the iterative nonlinear least squares solution failed with our given boundaries, we fall back on a simple linear fit to ensure a result. The logistic curve can take a near linear shape depending on the parameters, so it's still a reasonable result even if it lacks the familiar logistic shape. We then applied each fit to every species of fish (that had population data as not all of them had the variable "total number" filled out).

Next, to determine the accuracy of each of these fits to the 257 applicable species, we calculated the root mean squared error (RMSE) for each species such that $RMSE = \sqrt{\frac{1}{257}\sum_1^{257}(b_i-\hat{b_i})^2}$. Finally, we took the average RMSE across all the species. Thus, we then would have a singular value to determine fit accuracy. We then repeat this process for each of the fit equations. The R squared value is a more popular (and informative) regression statistic, but since we are working with nonlinear regression equations it does not properly apply to all of our models. For that reason a RMSE on normalized data was chosen as a way to effectively compare the error between all the fit equations, linear or nonlinear.

5 Results

Upon fitting all six fits (logistic, logistic-linear, logistic-sin, linear, 4th degree polynomial, and cubic spline) to all 257 applicable species, we determined the average RMSE for type of fit. The values of these are as follows (rounded to thousandths place):

Root Mean Squared Error	
Logistic:	0.202
Logistic-Linear:	0.183
Logistic-Sin:	0.198
Linear:	0.220
4th Polynomial:	0.186
Cubic Spline:	0.166
Mean Average Error	
Logistic:	0.141
Logistic-Linear:	0.126
Logistic-Sin:	0.139
Linear:	0.162
4th Polynomial:	0.132
Cubic Spline:	0.114

The graphs of the first twelve species for each fit is located in the appendices below.

6 Discussion

As we can see from the results above, all six of the fits are relatively accurate, with the most erroneous average fit having a value of 0.220. Going through the fits, the ranking of each fit is fairly intuitive: the linear fit will have the largest error since it can only be a straight line while the cubic spline will have the smallest error due to not only it being essentially a piece-wise function so it's not just restricted to one curve, but also it's relatively high degree of 3. For these two fits, it is easy to see their accuracy: the data is rarely linear so the linear fit won't work while the cubic spline has a large amount of flexibility so it has a much easier time fitting to all different kinds of data. You can see that in the graphs located in the appendices.

When it comes to the 3 logistic fits, their accuracy is mediocre in comparison to the cubic spline. Looking at their logistic RMSE values, the logistic linear fit seems to be the most accurate of the 3 while just the normal logistic curve is the worst. You can see why this is from the graphs: the tails of the normal logistic curves are horizontal lines while the tails for the logistic-linear curve are lines with a slope. This makes intuitive sense since fish populations tend to not stay the same for long periods of time.

While all of the fit types are relatively accurate, we've come to find that the logistic model is not the best model to most accurately fit fish population data when there are no assumptions. As we saw, their are other more accurate models such as the cubic spline.

Furthermore, we found that the normal logistic model is not as accurate as other variations of it: as seen in our results above, the logistic-linear curve is noticeably more accurate.

7 Contributions

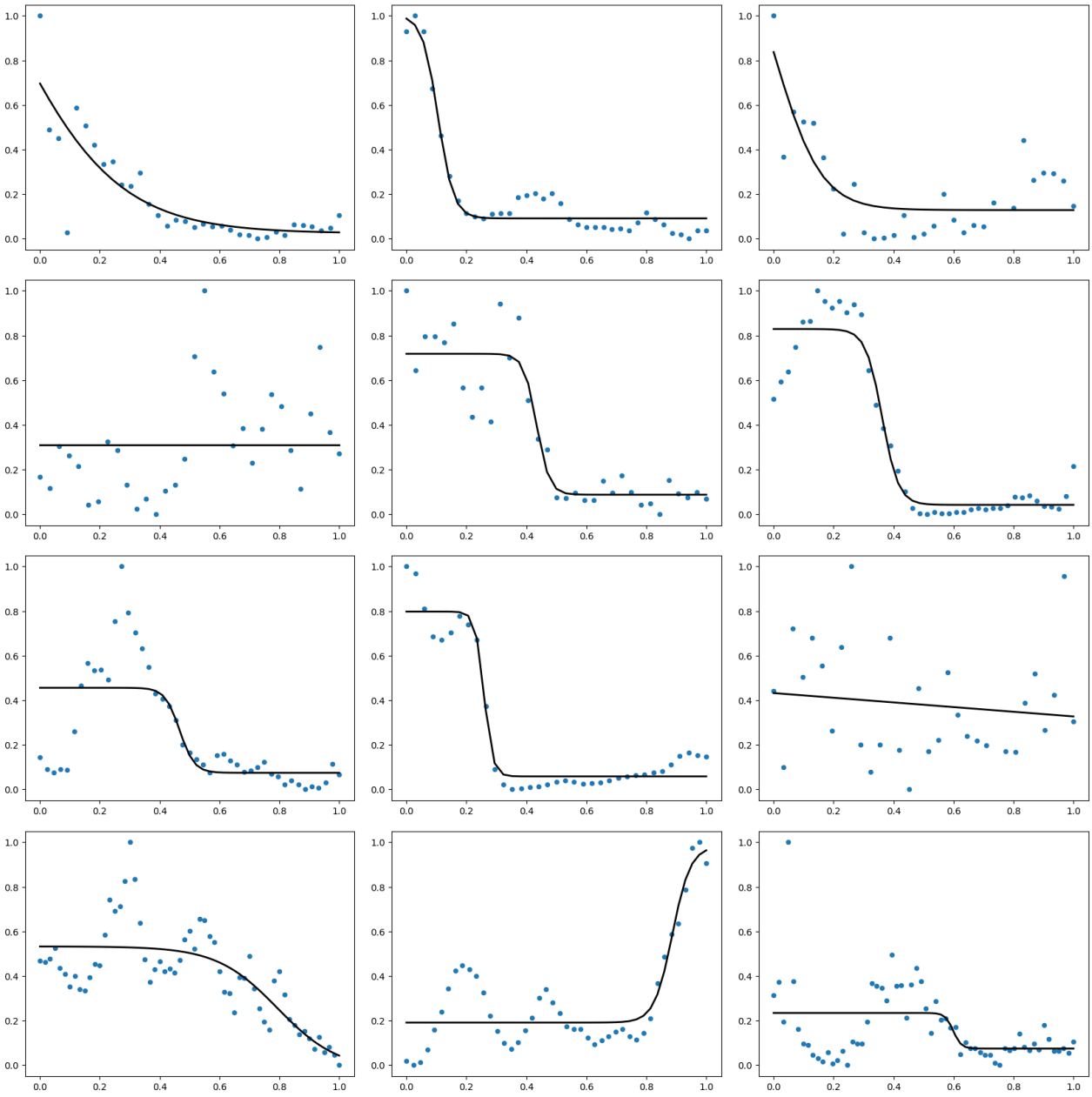
Aidan wrote up all the code and calculations. Noah came up with the idea for the project, suggesting the fits and looking at population data. Aaron found the data source and helped identify and filter which columns to use in order to accurately analyze this question. All three of us collaborated in writing up the report, interpreting the data, and drawing conclusions about the results.

8 Appendices

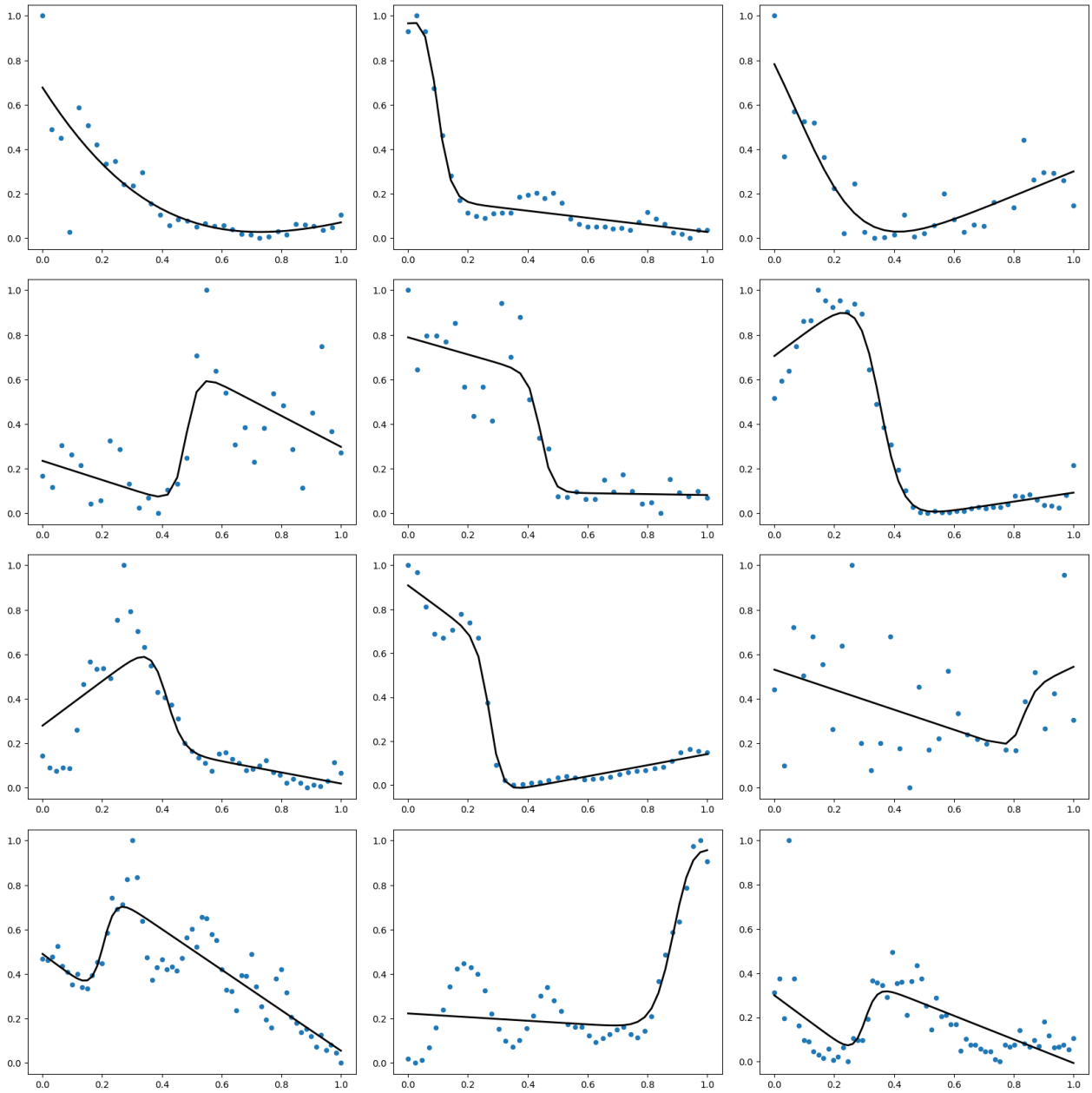
Our World in Data Fish Stocks Data Explorer (csv file can be downloaded from this source)
`https://ourworldindata.org/explorers/fish-stocks`

Google Collab python code used to compute fits, errors, and plots.
`https://colab.research.google.com/drive/1yJSY2yA2KK6JI7y7drbqlOR4a4CMPbTt?usp=sharing`

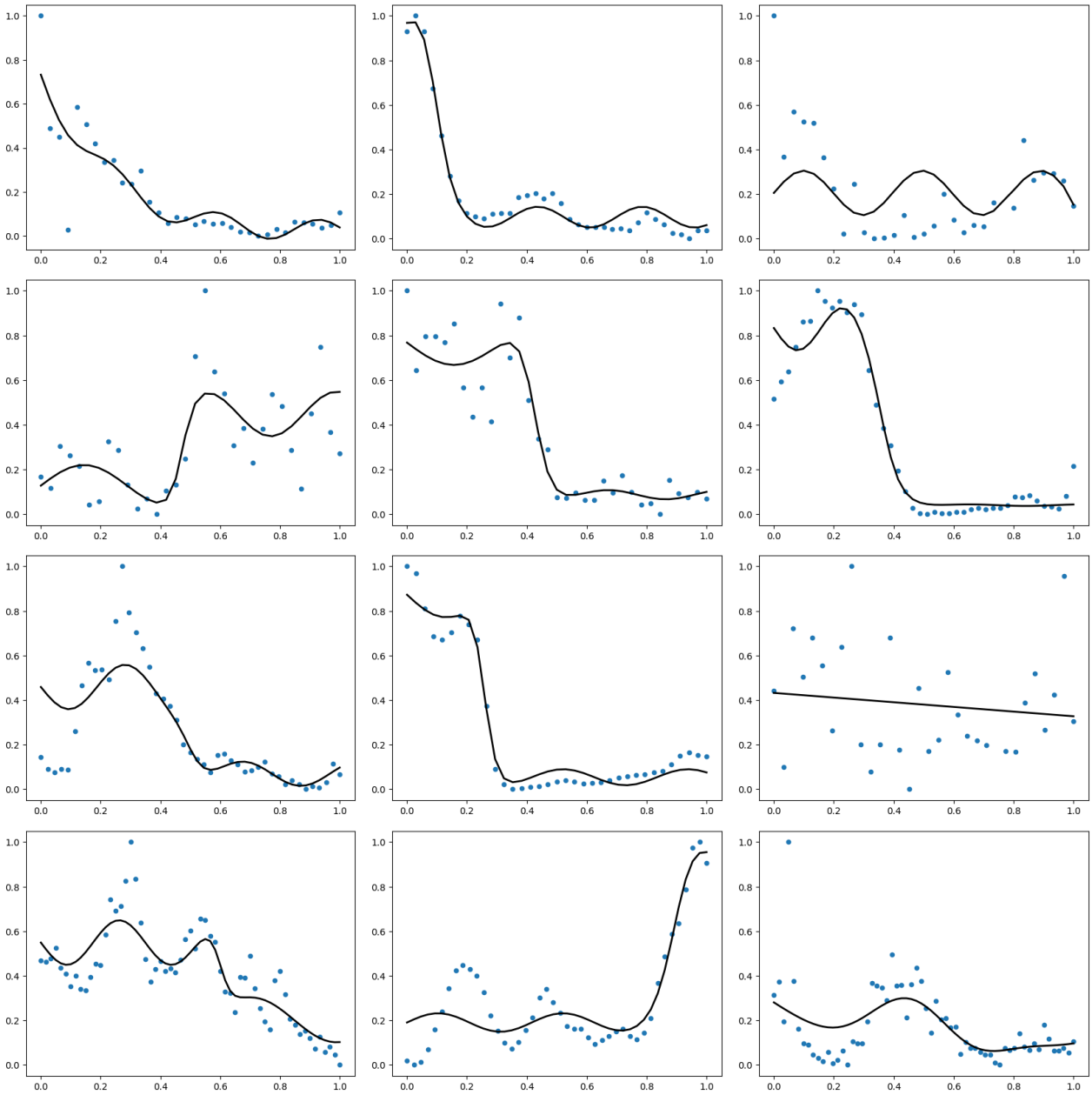
Logistic Curve, first twelve species, normalized population vs normalized time



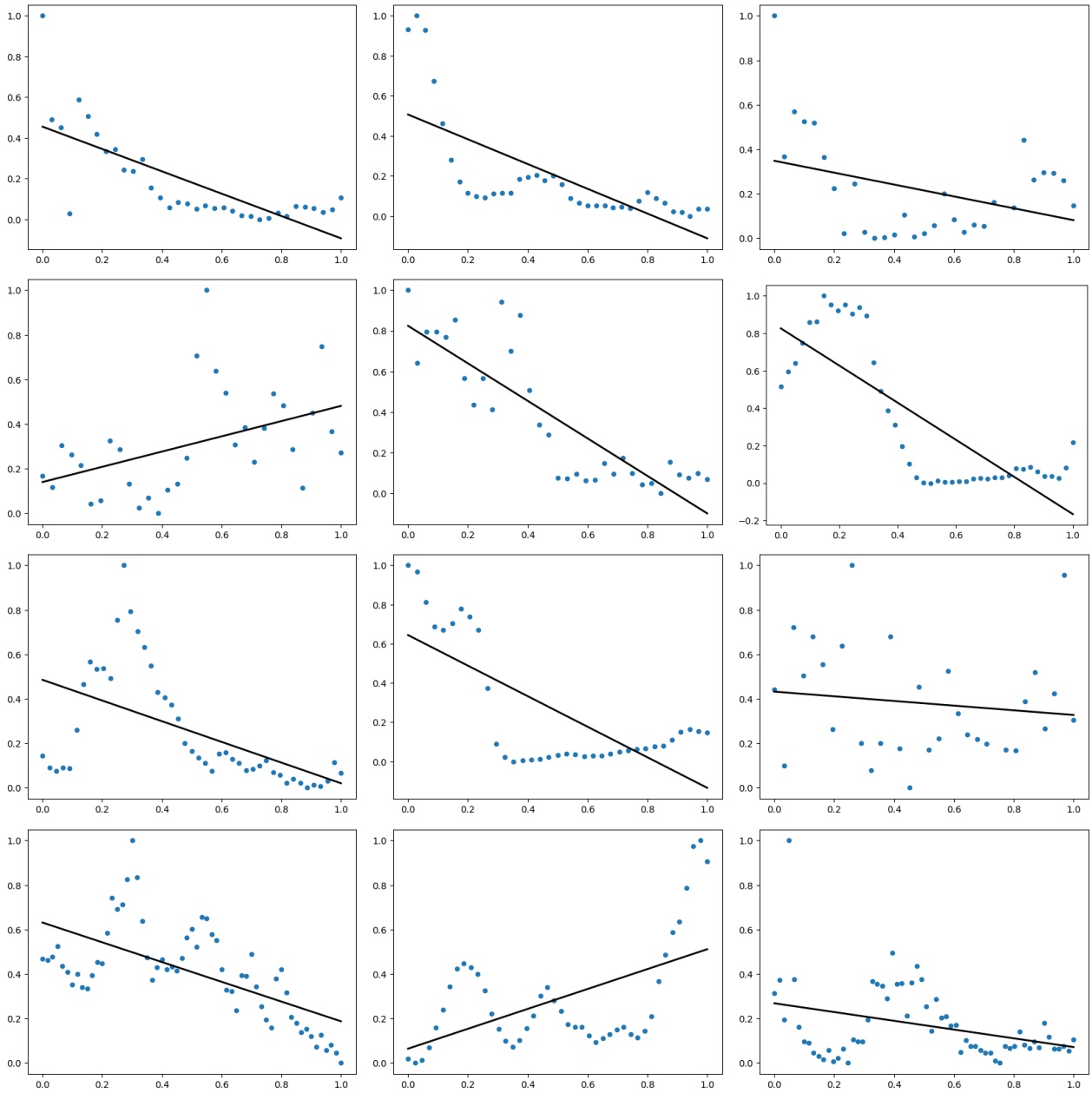
Logistic Linear Curve, first twelve species, normalized population vs normalized time



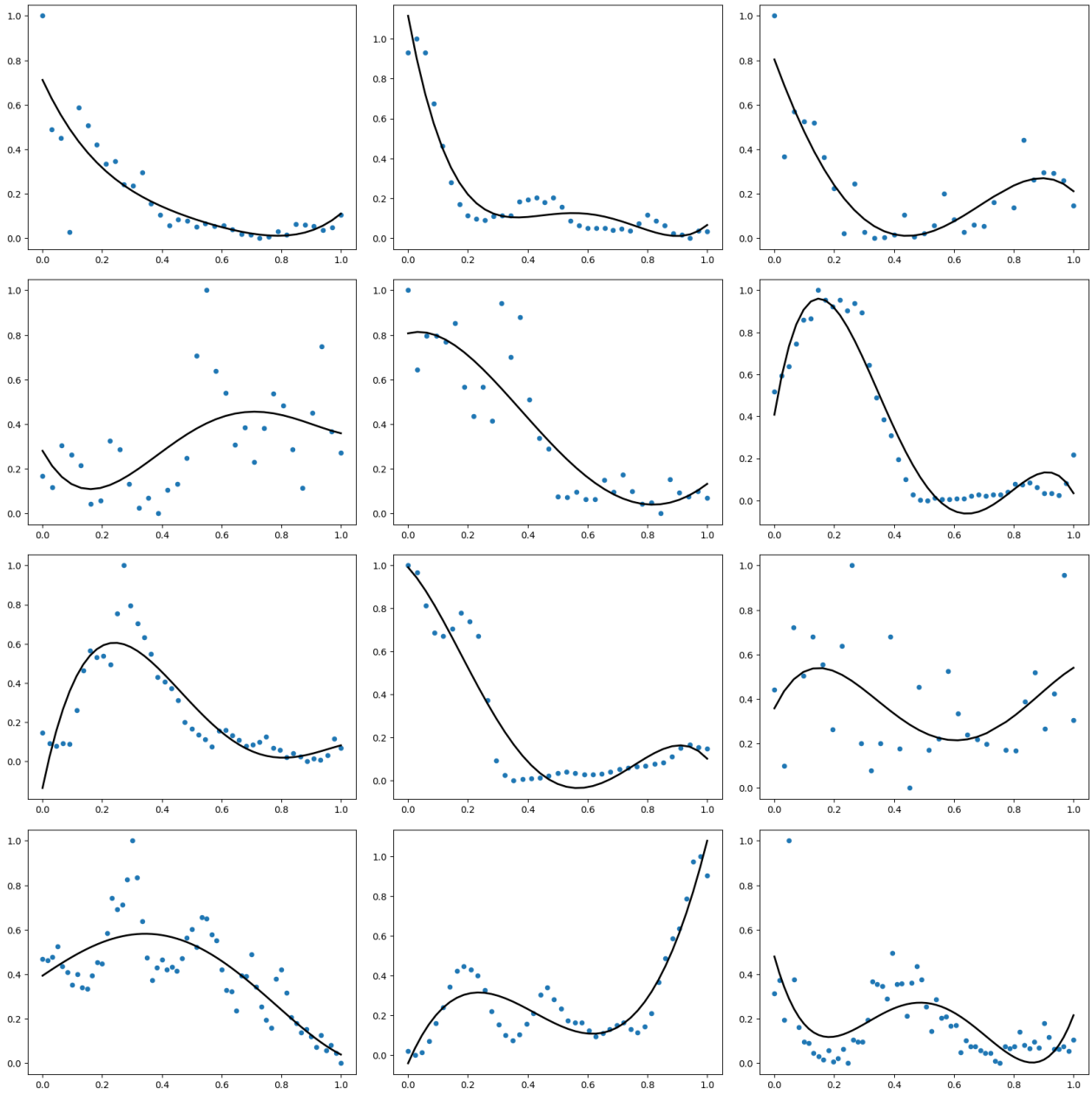
Logistic Sin Curve, first twelve species, normalized population vs normalized time



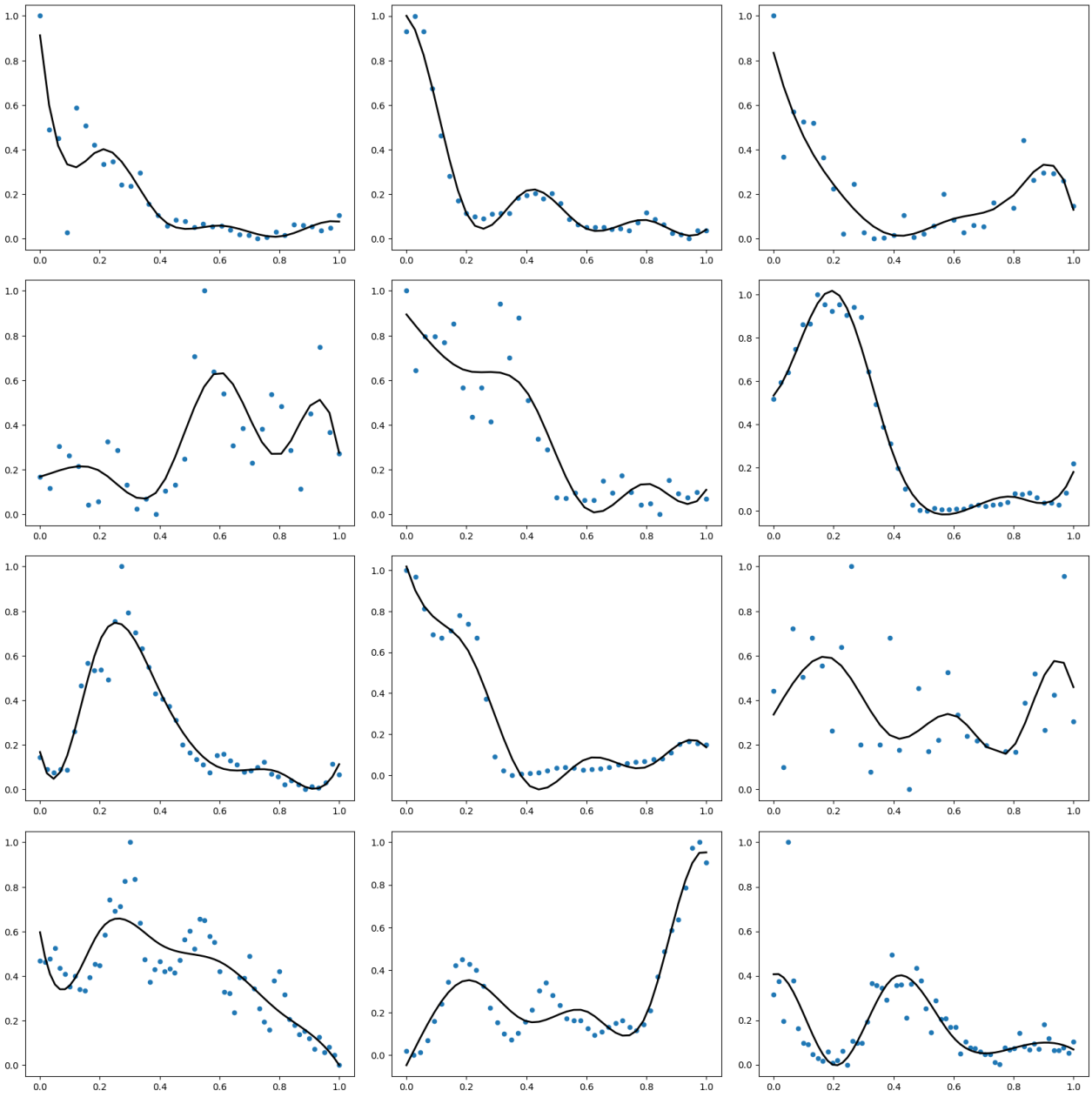
Linear Curve, first twelve species, normalized population vs normalized time



4th Degree Polynomial Curve, first twelve species, normalized population vs normalized time



Cubic Spline Curve, first twelve species, normalized population vs normalized time



9 Python Code

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import math
from scipy import stats
from scipy.optimize import curve_fit
from scipy.integrate import odeint
from scipy.interpolate import LSQUnivariateSpline, UnivariateSpline

# if youre not using google drive to upload the data then you need to change this section
from google.colab import drive
drive.mount('/content/drive')
df = pd.read_csv('/content/drive/MyDrive/fish-stocks.csv') # Read Our World In Data fish
                                                    stocks csv from google drive MyDrive folder

df = df[["Entity","Year","total_number"]] # Only include necessary columns
df = df.dropna(subset="total_number") # Filter out rows missing data
df = df.dropna(subset="Year") # Filter out rows missing data
df = df.dropna(subset="Entity") # Filter out rows missing data
Entities_total = pd.unique(df['Entity']) # list all species
Entities = Entities_total # cut down number of species tested if needed
print(str(len(Entities)) + "/" + str(len(pd.unique(df['Entity']))) + " species analyzed")

# LOGISTIC MODEL ----------------------------------------------------------------
# progress bar (ignore)
from IPython.display import HTML, display
def progress(value, max=len(Entities)-1):
    return HTML("""
        <progress
            value='{value}'
            max='{max}',
            style='width: 100%'
        >
            {value}
        </progress>
    """.format(value=value, max=max))
out = display(progress(0, len(Entities)-1), display_id=True)



# define the solved logistic function
def fit_func(x, K, r, a, b):
  return (K-b)/(1+np.exp(-r*(x-a)))+b

MAE = np.array([])
MSE = np.array([])
its = 0

# loop across the species ------------------------------------------------------

for i in Entities:
  dfi = df.query("Entity == @i") # set a new data frame of just the ith species
  Year = np.array((dfi["Year"]-min(dfi["Year"]))/np.ptp(dfi["Year"])) # normalize the years
  Pop = np.array((dfi["total_number"]-min(dfi["total_number"]))/np.ptp(dfi["total_number"]))
                                                      # normalize the pops

  # try a logistic fit, if curv_fit fails due to the data not permitting it, then try a
                                                     linear fit
  # a linear fit is guaranteed to work and zoomed in parts of the logistic curve are linear
                                                     so its not entirely irrelivent
  try:
    B_fit, B_cov = curve_fit(fit_func, Year, Pop, maxfev=1000, bounds=([-1,-150,-1,-1],[2,
                                                      150,2,2])) # fit the logistic curve
    fit = fit_func(Year, *B_fit) # define y values of fit curve
  except:
```

```
    B_fit = stats.linregress(Year, Pop) # fit a linear line
    fit = B_fit.intercept+B_fit.slope*(Year) # define y values of fit curve

  MSE = np.append(MSE, np.sum((fit-Pop)**2)/len(Pop)) # mean squared error
  MAE = np.append(MAE, np.sum(np.absolute((fit-Pop)))/len(Pop)) # mean absolute error
  # plt.figure()
  # plt.scatter(Year, Pop, s=20, label="data")
  # plt.plot(Year, fit, linewidth=2, color="black", label="solved logistic fit")
  out.update(progress(its, len(Entities)-1)) # update progress bar
  its += 1

# end of loop ----------------------------------------------------------------

logMSE = np.average(MSE)
logMAE = np.average(MAE)
logRMSE = np.sqrt(np.average(MSE))
logRMAE = np.sqrt(np.average(MAE))
#print(MAE)
print("Logistic MAE: " + str(np.average(MAE)))
# print(MSE)
print("Logistic MSE: " + str(np.average(MSE)))
print("Logistic RMSE: " + str(np.sqrt(np.average(MSE))))

# WEIRD LOGISTIC MODEL WHERE K DEPENDS ON TIME LINEARLY ------------------------
# progress bar (ignore)
from IPython.display import HTML, display
def progress(value, max=len(Entities)-1):
    return HTML("""
        <progress
            value='{value}'
            max='{max}',
            style='width: 100%'
        >
            {value}
        </progress>
    """.format(value=value, max=max))
out = display(progress(0, len(Entities)-1), display_id=True)



# define the solved logistic function
def fit_func(x, K, r, a, b, c, d):
  return (d*(x-1)+K-(c*x+b))/(1+np.exp(-r*(x-a)))+(c*x+b)

MAE = np.array([])
MSE = np.array([])
its = 0

# loop across the species ----------------------------------------------------

for i in Entities:
  dfi = df.query("Entity == @i") # set a new data frame of just the ith species
  Year = np.array((dfi["Year"]-min(dfi["Year"]))/np.ptp(dfi["Year"])) # normalize the years
  Pop = np.array((dfi["total_number"]-min(dfi["total_number"]))/np.ptp(dfi["total_number"]))
                                                    # normalize the pops

  # try a logistic fit, if curv_fit fails due to the data not permitting it, then try a
                                                    linear fit
  # a linear fit is guaranteed to work and zoomed in parts of the logistic curve are linear
                                                    so its not entirely irrelivent
  try:
    B_fit, B_cov = curve_fit(fit_func, Year, Pop, maxfev=1000, bounds=([-1,-50,-1,-1,-1,-1],
                                                    [2,50,2,2,1,1])) # fit the logistic curve
    fit = fit_func(Year, *B_fit) # define y values of fit curve
  except:
    B_fit = stats.linregress(Year, Pop) # fit a linear line
    fit = B_fit.intercept+B_fit.slope*(Year) # define y values of fit curve
```

```
  MSE = np.append(MSE, np.sum((fit-Pop)**2)/len(Pop)) # mean squared error
  MAE = np.append(MAE, np.sum(np.absolute((fit-Pop)))/len(Pop)) # mean absolute error
  # plt.figure()
  # plt.scatter(Year, Pop, s=20, label="data")
  # plt.plot(Year, fit, linewidth=2, color="black", label="solved logistic fit")
  out.update(progress(its, len(Entities)-1)) # update progress bar
  its += 1

# end of loop ----------------------------------------------------------------

loglinMSE = np.average(MSE)
loglinMAE = np.average(MAE)
loglinRMSE = np.sqrt(np.average(MSE))
loglinRMAE = np.sqrt(np.average(MAE))
#print(MAE)
print("Logistic-Linear MAE: " + str(np.average(MAE)))
# print(MSE)
print("Logistic-Linear MSE: " + str(np.average(MSE)))
print("Logistic-Linear RMSE: " + str(np.average(np.sqrt(MSE))))

# WEIRD LOGISTIC MODEL WHERE K DEPENDS ON TIME SIN WAVE ------------------------
# progress bar (ignore)
from IPython.display import HTML, display
def progress(value, max=len(Entities)-1):
    return HTML("""
        <progress
            value='{value}'
            max='{max}',
            style='width: 100%'
        >
            {value}
        </progress>
    """.format(value=value, max=max))
out = display(progress(0, len(Entities)-1), display_id=True)



# define the solved logistic function
def fit_func(x, K, r, a, b, a1, f1, a2, f2):
  return (K+a1*np.sin(f1*x)-b-a2*np.sin(f2*x))/(1+np.exp(-r*(x-a)))+(b+a2*np.sin(f2*x))

MAE = np.array([])
MSE = np.array([])
its = 0

# loop across the species ----------------------------------------------------

for i in Entities:
  dfi = df.query("Entity == @i") # set a new data frame of just the ith species
  Year = np.array((dfi["Year"]-min(dfi["Year"]))/np.ptp(dfi["Year"])) # normalize the years
  Pop = np.array((dfi["total_number"]-min(dfi["total_number"]))/np.ptp(dfi["total_number"]))
                                                  # normalize the pops

  # try a logistic fit, if curv_fit fails due to the data not permitting it, then try a
                                                  linear fit
  # a linear fit is guaranteed to work and zoomed in parts of the logistic curve are linear
                                                  so its not entirely irrelivent
  try:
    B_fit, B_cov = curve_fit(fit_func, Year, Pop, maxfev=1000, bounds=([-1,-50,-1,-1,-0.1,10
                                                    ,-0.1, 10],[2,50,2,2,0.1,20,0.1,20])) #
                                                    fit the logistic curve
    fit = fit_func(Year, *B_fit) # define y values of fit curve
  except:
    B_fit = stats.linregress(Year, Pop) # fit a linear line
    fit = B_fit.intercept+B_fit.slope*(Year) # define y values of fit curve

  MSE = np.append(MSE, np.sum((fit-Pop)**2)/len(Pop)) # mean squared error
  MAE = np.append(MAE, np.sum(np.absolute((fit-Pop)))/len(Pop)) # mean absolute error
```

```
  # plt.figure()
  # plt.scatter(Year, Pop, s=20, label="data")
  # plt.plot(Year, fit, linewidth=2, color="black", label="solved logistic fit")
  out.update(progress(its, len(Entities)-1)) # update progress bar
  its += 1

# end of loop ----------------------------------------------------------------

logsinMSE = np.average(MSE)
logsinMAE = np.average(MAE)
logsinRMSE = np.sqrt(np.average(MSE))
logsinRMAE = np.sqrt(np.average(MAE))
#print(MAE)
print("Logistic-Sin MAE: " + str(np.average(MAE)))
# print(MSE)
print("Logistic-Sin MSE: " + str(np.average(MSE)))

# LINEAR MODEL ---------------------------------------------------------------
# progress bar (ignore)
from IPython.display import HTML, display
def progress(value, max=len(Entities)-1):
    return HTML("""
        <progress
            value='{value}'
            max='{max}',
            style='width: 100%'
        >
            {value}
        </progress>
    """.format(value=value, max=max))
out = display(progress(0, len(Entities)-1), display_id=True)


MAE = np.array([])
MSE = np.array([])
its = 0

# loop across the species ----------------------------------------------------

for i in Entities:
  dfi = df.query("Entity == @i") # set a new data frame of just the ith species
  Year = np.array((dfi["Year"]-min(dfi["Year"]))/np.ptp(dfi["Year"])) # normalize the years
  Pop = np.array((dfi["total_number"]-min(dfi["total_number"]))/np.ptp(dfi["total_number"]))
                                                  # normalize the pops
  B_fit = stats.linregress(Year, Pop) # fit a linear line
  fit = B_fit.intercept+B_fit.slope*(Year) # define y values of fit curve

  MSE = np.append(MSE, np.sum((fit-Pop)**2)/len(Pop)) # mean squared error
  MAE = np.append(MAE, np.sum(np.absolute((fit-Pop)))/len(Pop)) # mean absolute error
  # plt.figure()
  # plt.scatter(Year, Pop, s=20, label="data")
  # plt.plot(Year, fit, linewidth=2, color="black", label="solved logistic fit")
  out.update(progress(its, len(Entities)-1)) # update progress bar
  its += 1

# end of loop ----------------------------------------------------------------

linMSE = np.average(MSE)
linMAE = np.average(MAE)
linRMSE = np.sqrt(np.average(MSE))
linRMAE = np.sqrt(np.average(MAE))
# print(MAE)
print("Linear MAE: " + str(np.average(MAE)))
# print(MSE)
print("Linear MSE: " + str(np.average(MSE)))

# 4th DEGREE POLYNOMIAL MODEL
                                                  ----------------------------------------------------------
```

```
# progress bar (ignore)
from IPython.display import HTML, display
def progress(value, max=len(Entities)-1):
    return HTML("""
        <progress
            value='{value}'
            max='{max}',
            style='width: 100%'
        >
            {value}
        </progress>
    """.format(value=value, max=max))
out = display(progress(0, len(Entities)-1), display_id=True)


MAE = np.array([])
MSE = np.array([])
its = 0
degree = 4

# loop across the species ---------------------------------------------------

for i in Entities:
  dfi = df.query("Entity == @i") # set a new data frame of just the ith species
  Year = np.array((dfi["Year"]-min(dfi["Year"]))/np.ptp(dfi["Year"])) # normalize the years
  Pop = np.array((dfi["total_number"]-min(dfi["total_number"]))/np.ptp(dfi["total_number"]))
                                                  # normalize the pops
  fit = np.polyval(np.polyfit(Year,Pop,degree),Year) # fit a polynomial
  MSE = np.append(MSE, np.sum((fit-Pop)**2)/len(Pop)) # mean squared error
  MAE = np.append(MAE, np.sum(np.absolute((fit-Pop)))/len(Pop)) # mean absolute error
  # plt.figure()
  # plt.scatter(Year, Pop, s=20, label="data")
  # plt.plot(Year, fit, linewidth=2, color="black", label="solved logistic fit")
  out.update(progress(its, len(Entities)-1)) # update progress bar
  its += 1

# end of loop ---------------------------------------------------------------

polyMSE = np.average(MSE)
polyMAE = np.average(MAE)
polyRMSE = np.sqrt(np.average(MSE))
polyRMAE = np.sqrt(np.average(MAE))
# print(MAE)
print("4th Degree Polynomial MAE: " + str(np.average(MAE)))
# print(MSE)
print("4th Degree Polynomial MSE: " + str(np.average(MSE)))

# CUBIC SPLINE MODEL --------------------------------------------------------
# progress bar (ignore)
from IPython.display import HTML, display
def progress(value, max=len(Entities)-1):
    return HTML("""
        <progress
            value='{value}'
            max='{max}',
            style='width: 100%'
        >
            {value}
        </progress>
    """.format(value=value, max=max))
out = display(progress(0, len(Entities)-1), display_id=True)


MAE = np.array([])
MSE = np.array([])
its = 0
num_knots = 4
```

```
knots = np.linspace(0,1,num_knots+2)[1:-1]

# loop across the species ------------------------------------------------------

for i in Entities:
  dfi = df.query("Entity == @i") # set a new data frame of just the ith species
  Year = np.array((dfi["Year"]-min(dfi["Year"]))/np.ptp(dfi["Year"])) # normalize the years
  Pop = np.array((dfi["total_number"]-min(dfi["total_number"]))/np.ptp(dfi["total_number"]))
                                                  # normalize the pops
  try:
    fit = LSQUnivariateSpline(Year,Pop,knots)(Year) # fit a spline
  except:
    B_fit = stats.linregress(Year, Pop) # fit a linear line
    fit = B_fit.intercept+B_fit.slope*(Year) # define y values of fit curve
  MSE = np.append(MSE, np.sum((fit-Pop)**2)/len(Pop)) # mean squared error
  MAE = np.append(MAE, np.sum(np.absolute((fit-Pop)))/len(Pop)) # mean absolute error
  # plt.figure()
  # plt.scatter(Year, Pop, s=20, label="data")
  # plt.plot(Year, fit, linewidth=2, color="black", label="solved logistic fit")
  out.update(progress(its, len(Entities)-1)) # update progress bar
  its += 1

# end of loop ------------------------------------------------------------------

splMSE = np.average(MSE)
splMAE = np.average(MAE)
splRMSE = np.sqrt(np.average(MSE))
splRMAE = np.sqrt(np.average(MAE))
# print(MAE)
print("Cubic Spline MAE: " + str(np.average(MAE)))
# print(MSE)
print("Cubic Spline MSE: " + str(np.average(MSE)))

# ALL ERROR EVALUATIONS
                                                  ------------------------------------------------------------

print("All Root Mean Squared Errors, lower is better:")
print("Logistic RMSE:         " + '{:.3f}'.format(round(logRMSE, 3)))
print("Logistic-Linear RMSE: " + '{:.3f}'.format(round(loglinRMSE, 3)))
print("Logistic-Sin RMSE:     " + '{:.3f}'.format(round(logsinRMSE, 3)))
print("Linear RMSE:           " + '{:.3f}'.format(round(linRMSE, 3)))
print("4th Polynomial RMSE:  " + '{:.3f}'.format(round(polyRMSE, 3)))
print("Cubic Spline RMSE:     " + '{:.3f}'.format(round(splRMSE, 3)))
print()
print("All MAE Values, lower is better:")
print("Logistic MAE:         " + '{:.3f}'.format(round(logMAE, 3)))
print("Logistic-Linear MAE: " + '{:.3f}'.format(round(loglinMAE, 3)))
print("Logistic-Sin MAE:     " + '{:.3f}'.format(round(logsinMAE, 3)))
print("Linear MAE:           " + '{:.3f}'.format(round(linMAE, 3)))
print("4th Polynomial MAE:  " + '{:.3f}'.format(round(polyMAE, 3)))
print("Cubic Spline MAE:     " + '{:.3f}'.format(round(splMAE, 3)))
print()
```