World4: Systems Modeling of Carrying Capacity

Team US-9365

April 19, 2019

Contents

1 Summary Sheet

The rapid rise of global population over the past half century has been accompanied by increased stress on Earth's ecological systems. As humanity cautiously steps through the twenty-first century, achieving sustainable growth must be a policy priority. This paper uses a novel approach to develop a systems model of Earth's carrying capacity and the associated factors. We call this model *World4*.

First, we compiled a uniquely comprehensive dataset of 35 global indicators over several decades. We sourced data from United Nations, the Global Footprint Network, the OECD, and the World Bank. We deployed R to preprocess the data and Excel to combine the datasets.

Next, we utilized Spearman's Rank-Order Correlation to isolate important indicators and relationships. We wrote a Python script to calculate Spearman's correlation between each pair of variables and compile the results into a matrix. We wrote a second Python script to generate potential diagrams. All pairs of variables with a Spearman's correlation greater than 0.9 were represented in the graph as an edge.

Finally, we used these generated graphs to guide the construction of a stock and flow diagram of our model. We used another Python script to fit each of the relationships to a linear, logistic, exponential, or polynomial curve. We implemented a Nelder-Mead algorithm in Python to find optimal strategies for raising carrying capacity.

We found that the most significant limiting factors were **CO2 emissions, nitrogen use, phosphorous use, and biodiversity**. These results were consistent with the findings of the Stockholm Resiliency Center. We used World4 to project world population growth until 2201 under current environmental conditions and found that **Earth's carrying capacity is approximately 12.5 billion humans**. We simulated potential strategies for achieving sustainabiltiy and found that **reductions in industrial carbon emissions could increase carrying capacity to 15.5 billion**.

The World4 model is not perfect: it is limited by the available data and still relies much on intuition. However, the model performs remarkably well, demonstrating the efficacy of our novel approach.

2 Introduction

2.1 Background

In March of 2019, the United Nations released its sixth *Global Environment Outlook*. The report called on policymakers to confront the challenge of sustainable growth, highlighting climate change, pollution, land degradation, biodiversity loss and water scarcity as pressing concerns (Elkins, Gupta, & Boileau, 2019). At the same time, the United Nations projects that global population will reach 9.8 billion people by the year 2050, and 11.2 billion people by 2100 (UN, 2017). As our global ecological footprint continues rise, many researchers have become interested in modeling Earth's carrying capacity.

Earth's carrying capacity describes the number of people it can sustainably host. Previous research has disagreed greatly on the number, with estimates ranging from half a million to one sixtillion (Bruce, 2012). Some studies argue that we have already far surpassed Earth's planetary boundaries, whereas others can foresee no probems at all. These disagreements are a barrier to public knowledge and effective policymaking.

We believe that before implementing policy solutions, researchers need to clearly define the problem. The goal of this paper is to design a model that can use accessible, objective factors to predict carrying capacity.

2.2 Interpretation

We make the following two assumptions throughout the paper:

First, we use Herman Daly's three criteria for sustainability. For a renewable resource, consumption must be limited by the rate of regeneration; for a non-renewable resource, consumption must be limited by the rate at which a renewable resource can be substituted for it; for a pollutant, emission must be limited by the rate at which it can be absorded by a sink (Daly, 1996). Daly's guidelines are widely accepted in academic, business, government, and civic circles.

Second, we choose to use a dynamic systems approach. Most previous studies have identified a single constraining factor or a set of possible constraints (such as water, food, and land) and then used it to calculate the carrying capacity (Cohen, 1995). Our approach seeks to identify the relationships between several constraining factors and calculate the ultimate, combined impact on limiting poulation growth.

2.3 Inspiration

This paper draws great inspiration from *Beyond the Limits*, a pioneering piece of work in the application of dynamic systems to carrying capacity (Meadows, Randers, & Meadows, 2005). The work built a powerful case for sustainability and was accessible to the general public.

Their model, dubbed World3, aims to model carrying capacity not as a product of a set of disjoint factors, but rather as a network of constraints. Factors, such as births per year, fertility, or food, are related to one another with either positive or negative connections. See **Appendix A** for a diagram of World3.

The goal of this paper is to construct World4: an update to the World3 model informed by modern machine learning techiques. First, we will use the most recent datasets to isolate key factors that determine carrying capacity. Second, we will use this model to calculate Earth's current carrying capacity. Finally, we will deploy meachine learning techniques to generate future scenarios to test possible courses of action.

3 Methods

3.1 Data Collection

No single database includes all of the factors we need for a comprehensive analysis. To take full advantage of the range of available sources, we utilized data from the Global Footprint Network (GFN), the United Nations Food and Agriculture Organization (FAO), the Organisation for Economic Co-operation and Development (OECD), and the World Bank.

The FAO data includes all food and agricultural products imported and exported for over 245 countries and territories from 1961 to 2013 (FAO, 2019). We considered livestock production, energy usage, and CO2 emissions, among other factors. The data were collected according to the Standard International Merchandise Trade Statistics Methodology. See **Appendix B** for the FAO methodology.

The GFN data includes the ecological resource use and resource capacity of over 200 countries and territories from 1961 to 2018 (GFN, 2018). We considered the total and per capita ecological footprint by crop land, grazing land, forest land, fishing ground, built-up land, and carbon. The calculations were made from datasets published by the United Nations and supplementary sources. See **Appendix C** for the GFN methodology.

The OECD data includes the output of industrial establishments and covers sectors such as mining, manufacturing, electricity, gas and steam and air-conditioning for all OECD countries from 1919 to 2018 (OECD, 2019). We considered the combined production and manufacturing indices. The dataset was compiled from data provided by member countries. See **Appendix D** for the OECD methodology.

The World Bank data includes the a variety of world development indicators, including education, poverty, infrastructure, and equality, across 264 countries, regions, and territories (WB, 2017). We considered the birth rate and mortality rate. The World Bank data is compiled from a variety of official sources which we judged to be reputatable. See **Appendix E** for the World Bank methodology.

The datasets were downloaded in CSV format and pre-processed in R and Excel. We segmented each of the datasets by year, region, and element and then used a left-join operation to combine the data into one file. We eliminated unnecesssary or redundant factors (e.g. by combining poultry, lamb, and beef under one "livestock" production element) and excluded variables for which data was too sparse (e.g. percent water devoted to agriculture). See **Appexdix F** for the pre-processing code in R.

Ultimately, our dataset spanned 35 variables and 1482 datapoints. The data from 1992 to 2014 was complete, and other time periods had varying levels of missing data. A complete

list of the variables can be found in **Appendix G**.

We then scaled down each factor to a range of 0 to 1. The normalized data is graphed below.

Figure 1: Sample of Preprocessed Data

3.2 Correlation Analysis

We needed to narrow our dataset down to only the relevant factors. This task was difficult in two respects. First, two factors may not be directly correlated, but can be have an indirect correlation mediated through other factors. Second, factors may be related to one another differently, including linear, exponential, logistic, and other relationships. These two factors made conventional linear regression unsuited for our task.

To address this issue, we take advantage of Spearman's Rank Order Correlation. Spearman's correlation is an effective way to measure the strength and direction of association between two variables (Salkind, 2010). Our variables are measured on a interval scale and are posess monotonic relationships, which conforms with the assumptions of Spearman's correlation.

Our data are continuous and sampled at a small subset of yearly intervals, so we do not expect values to repeat. Thus, we deploy the no tied-ranks formula:

$$\rho = 1 - \frac{6 \cdot \sum d_i^2}{n(n^2 - 1)}$$

where d_i is the difference in paired ranks and n is the number of cases.

To conduct the computationally intense calculationss, we developed a multi-threaded Python code. Our implementation performs fourteen times faster than the standard, equivalent implementation in R. See **Appendix H** for our implementation.

These results were compiled in a matrix, with columns and rows representing each varaibles and the cells containing their Spearman's correlation. A sample of this matrix is below.

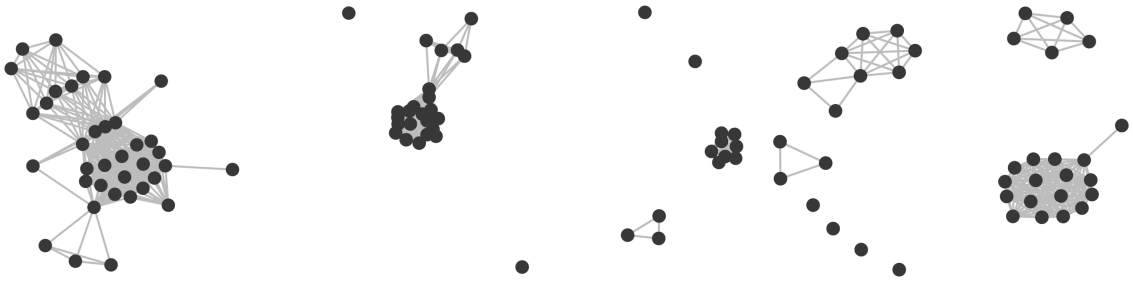
	Population	Cropland	Percipitation	Emissions	Birth Rate	...
Population	1	0.997334	0.84781	0.590706	-0.99234	...
Cropland	0.997334	1	0.844196	0.588741	-0.98913	...
Percipitation	0.84781	0.844196	1	0.810161	-0.84559	...
Emissions	0.590706	0.588741	0.810161	1	-0.59071	...
Birth Rate	-0.99234	-0.98913	-0.84559	-0.59071	1	...
...	...	...	...	...	...	

Figure 2: Sample of Spearman's Correlation Matrix

3.3 Graph Visualization

After calculating the Spearman's matrix, we constructed a graph based on a cutoff value denoted φ. If the Spearman's correlation for two variables was greater than φ, we indicated this with an edge on the graph. Otherwise, no edge is drawn. If a variable was not sufficiently associated with any other variable, it was excluded from the final model.

We utilized the ggnet2 package to write a R program that generates visualizations based off input values of φ. These graphs were used to intuit an ideal φ value. See **Appendix I** for the implementation.



(a) $\varphi = 0.8$ (b) $\varphi = 0.85$ (c) $\varphi = 0.9$

Figure 3: Graph Visualizations for Sample Values of φ

Upone examining the graphs, we concluded that φ values of 0.8 amd 0.85 were inappropriate, as the generated graphs were too densely connected and thus unable to isolate meaningful relations. We decided to use a φ value of 0.9. We used the Viridis package to generate distinct colors to represent each of the variables, resulting in the following graph.

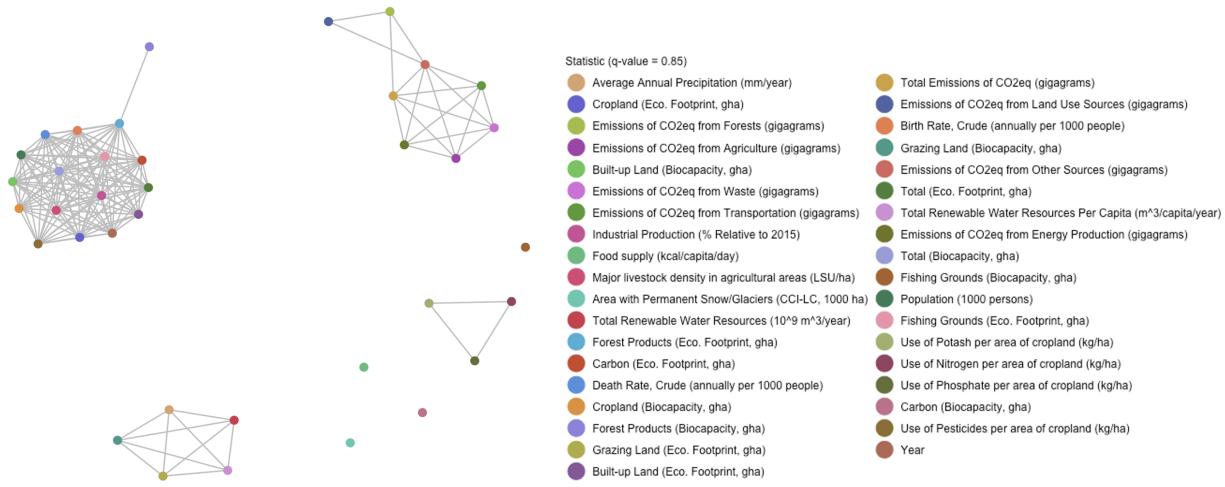


Figure 4: Color-Coded Visualization for $\varphi = 0.9$

Eliminating the variables with no associations, we were able to utilize a condensed set of associated variables to isolate and model important relations.

3.4 Model Construction

We used the results of the graph visualization to guide the construction of our model. The model is represented with a stock and flow diagram, with rectangles representing stocks and circles representing flows. The diagram was designed in Microsoft Powerpoint.

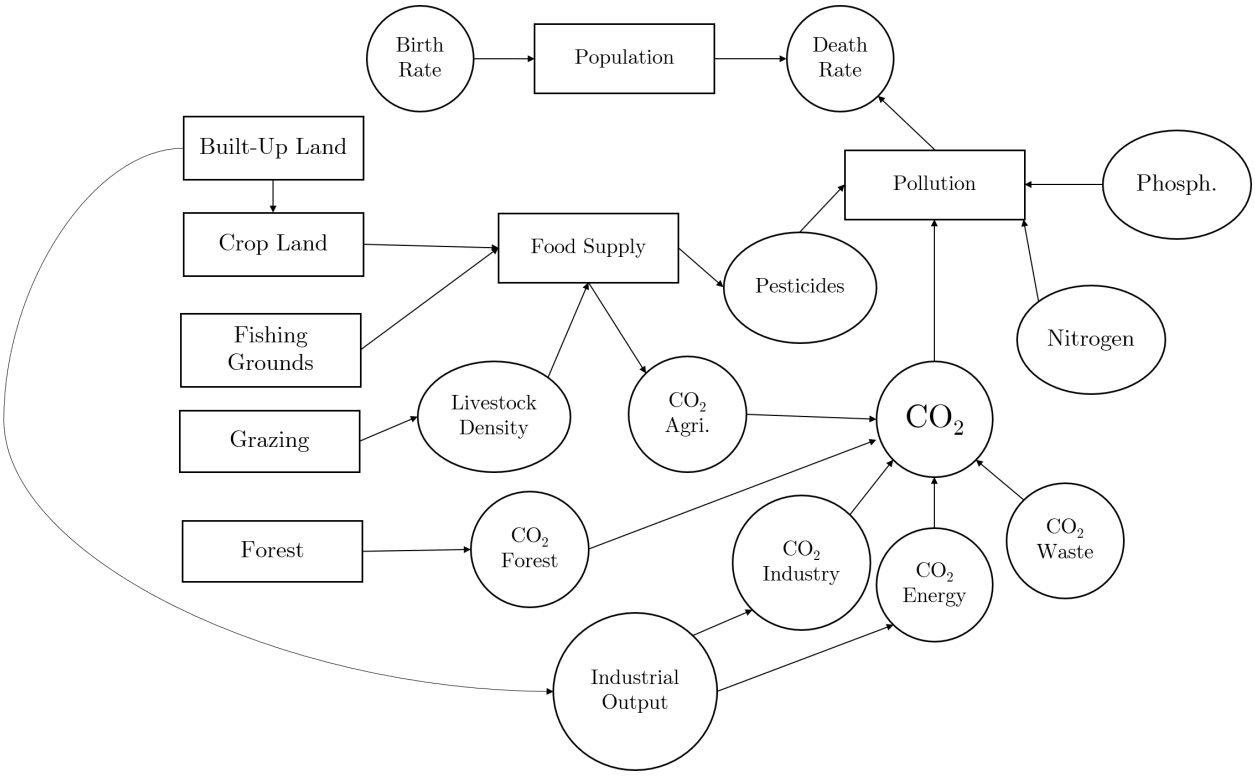


Figure 5: World4 Stock and Flow Diagram

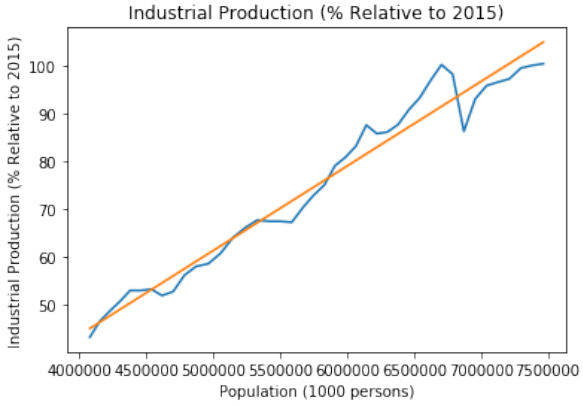
This diagram served as the foundation for World4.

3.5 Edge Relations

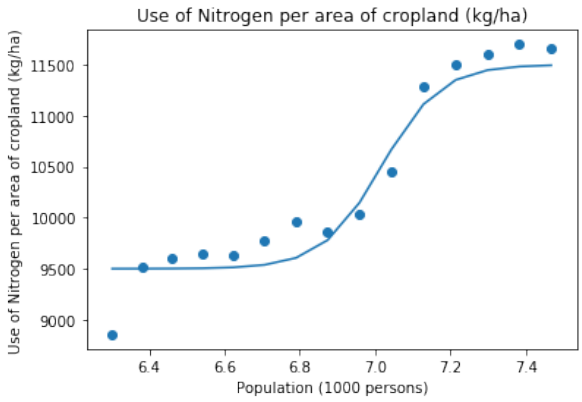
The next task was to develop a mathamatical representation of the relations between each of the variables. A common criticism of the World3 model was of its unduly pessimistic outlook. Critics claimed that the methods were inaccurate, oversimplified or biased. For example, one criticism was the assumption of continuous and exponential growth in GDP (Vaclav, 2005). Another criticized the use of linear equations to model the rate of consumption (Cole, 1973).

At the heart of the problem was a lack of access to the computational power and breadth of data to generate models purely from the data. This limitation forced a realiance on estimation and intuition.

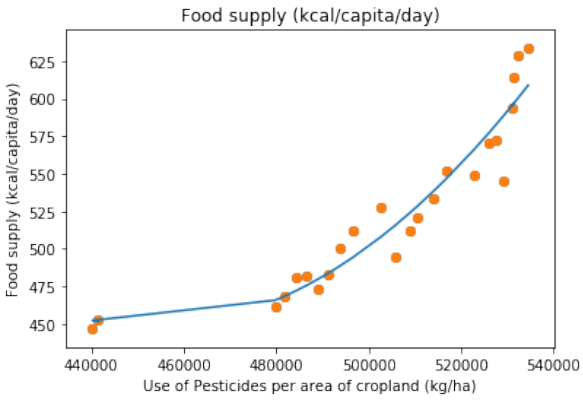
In World4, we first generated a graph of each of the relations. We observed that most of the graphs could categorized as linear, logistic, or exponential. In addition, the graphs which did not fit these categories appeared sufficient polynomial. Some sample graphs are shown below.



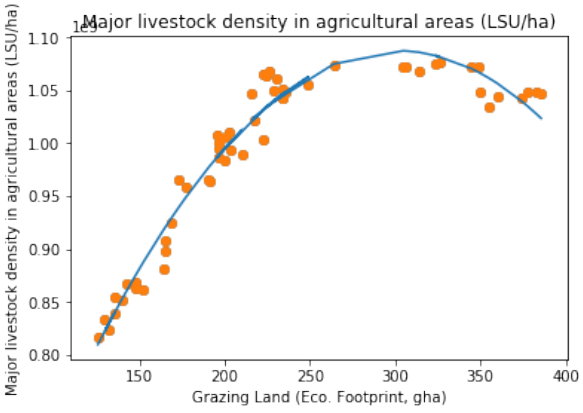
(a) Linear



(b) Logistic



(c) Exponential



(d) Polynomial

Figure 6: Sample Relations Between Variables

We wrote a Python script that would test whether a linear, logistic, or exponential regression best fit each edge. Then, assigned the best fitting curve to the corresponding edge. For select edges for which none were appropriate were appropriate, we fitted the data to a polynomial curve. See **Appendix J** for the implementation of World4.

3.6 Gradient-Free Optimization

The final task was to find an optimal set of factors to minimize the effect of limiting factors. However, the complexity and ambiguity of the World4 model renders it resistant to gradient-based optimization. Therefore, we chose instead to utilize gradient-free optimization. We deployed the Nelder-Mead method to optimize the limiting factors.

The Nelder-Mead method is an effective and computationally compact method for finding local minima (Matthews & Fink, 2004). The algorithm searches the input space by tracking three points, forming a triangle. At each iteration, the point of the triangle with the largest value is rejected and replaced with a new vertex. The process generates a series of triangles, with decreasing size, converging at a local minimum. We used Python and the Pandas package to implement the algorihtm. See **Appendix K** for our implementation.

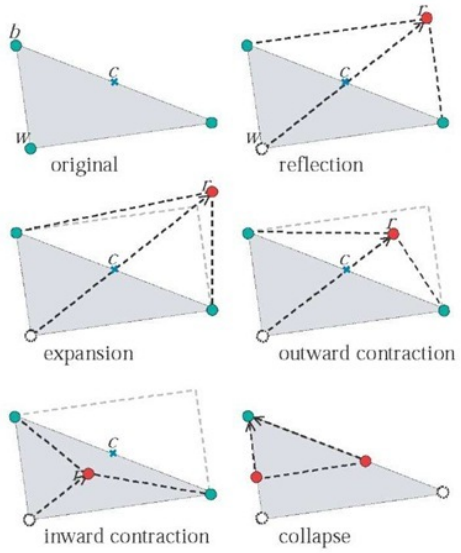


Figure 7: Visualization of the Nelder-Mead Algorithm

We allowed the agent to influence the distribution of CO2 emissions among the sources, as well as levels of food supply and industrial production. This novel application of the Nelder-Mead method was able to successfully and rapidly discover a solutions.

4 Results

4.1 Limiting Factors

Pollution and food supply were most strong associated with birth rate, death rate, and population.

Pollution is a conglomeration of presticide use, CO2 emissions, nitrogen consumption, and phosphorous consumption. Food supply is contingent on crop land and fishing grounds. These, in turn, are influenced by climate change and biodiversity.

These results cohere with recent studies which highlight biogeochemical flows, biodiversity, and climate change as key planetary boundaries (Steffen et al., 2015).

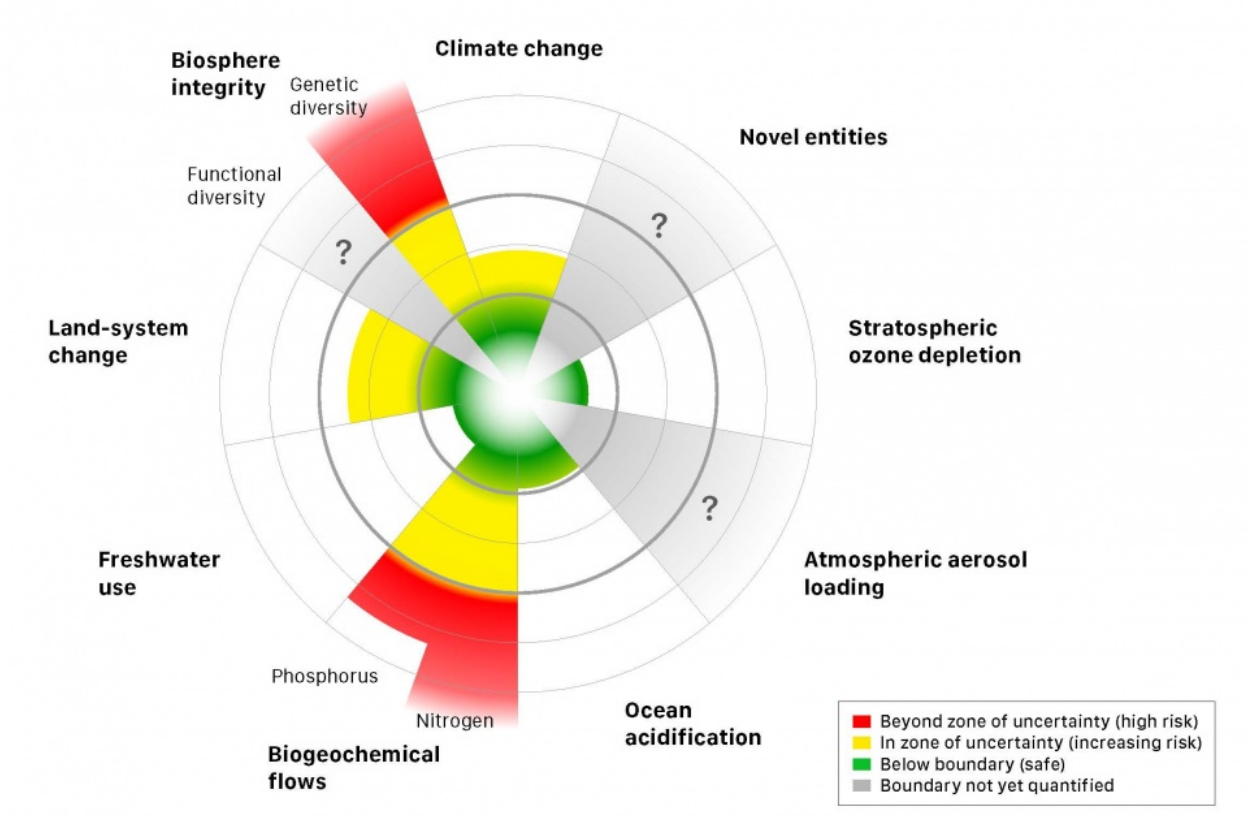


Figure 8: Stockholm Resiliency Center's Planetary Boundaries

These results are limited, however, in several respects. First, our analysis was only able to establish a correlation, and not necessarily causation. Second, we also were not able to exclude the influence of comfonding variables. Finally, our set of variables lacked many potentially important factors, such as freshwater depletion, non-renewable energy use, or family planning.

4.2 Current Carrying Capacity

Our model generated the following prediction for global population growth, assuming that current conditions are held constant.

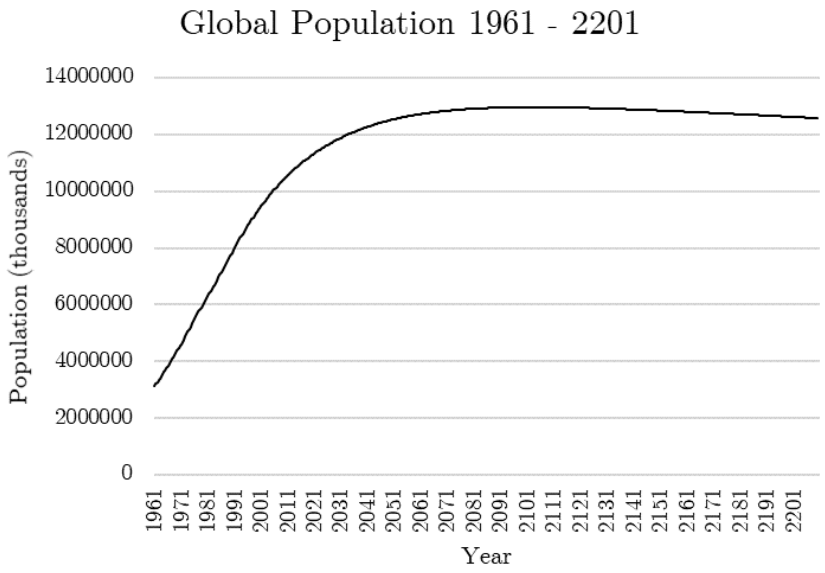


Figure 9: World4 Current Population Growth Projection

Our results cohere with the United Nations "medium fertility" scenario, which supports the general efficacy of our model. As population appears to be bounded at 12.5 billion humans, we conclude that Earth's current carrying capacity is 12.5 billion.

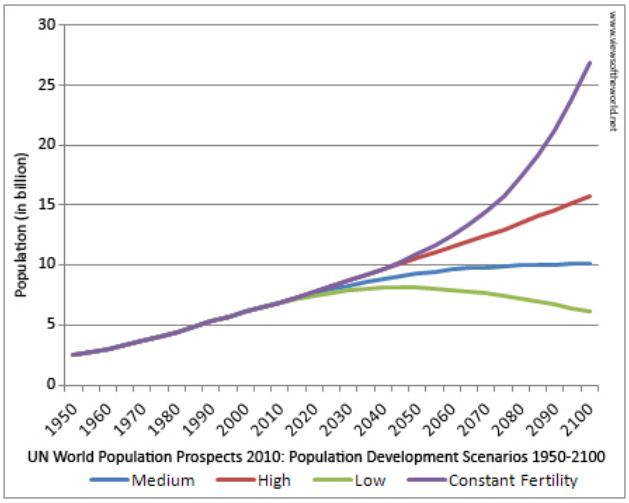
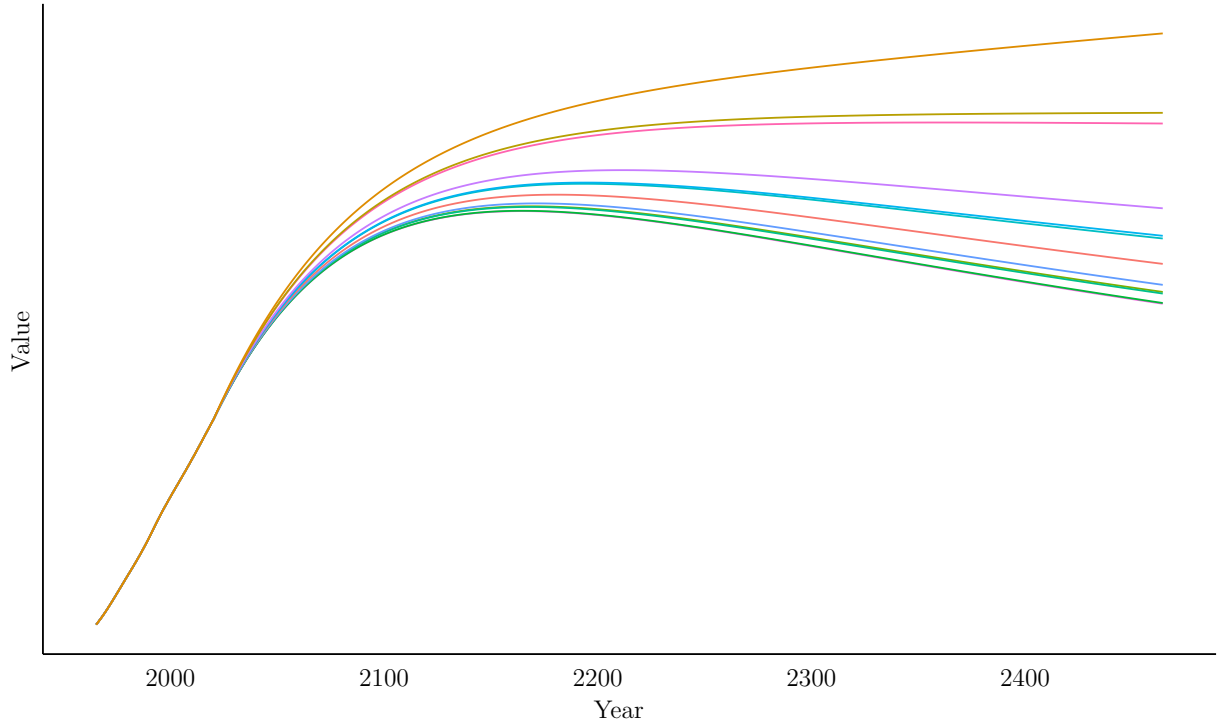


Figure 10: United Nations Scenarios for Population Growth

4.3 Future Projections

We ran multiple Nelder-Mead searches to simulate potential strategies for sustainability. The algorithm chose strategies that heavily minimized industry CO2, while showing little to no interest in CO2 from transportation, CO2 from waste, foot production, or industrial output. We found that in the best case scenario, significant reductions in CO2 from industry increase Earth's carrying capacity to 15.5 billion, an additional 3 billion people.



5 Conclusion

Our model sought to use a systems modeling approach, analyzing the interdepednence and influences of a network of varaibles. From the results of the Spearman's correlation, we were able to isolate the important relationships between the variables. The result was a strong emphasis on the influence of environmental pullutants and food supply. This implied the importance of CO2 emissions, nitrogen use, phospohrous use, and biodiversity.

Next, we use the correlation analysis to build a stock and flow diagram. We fit the relationships between the variables to linear, logistic, exponential, or polynomial curves as necessary. We conducted a simulation with this model using current conditions and found that global population would be limited by a carrying capacity of approximately 12.5 billion humans.

Finally, we used a Nelder-Mead algorithm to search for optimal strategies for sustainable growth. We discovered that by eliminating select sources of carbon emissions, the carying capacity could be raised to 15.5 billion.

Futures studies could use our method to generate dynamic system diagrams from a broader set of a data. This would enable the World4 model to incorporate a braoder array of indicators and thus provide more accurate results. Studies may also consider using neural networks or other complicated data structures to model the relationships between variables. This would enable the model capture nuances that would other be lost if fit to a simple curve.

6 Appendix

Appendix A: World3 Diagram

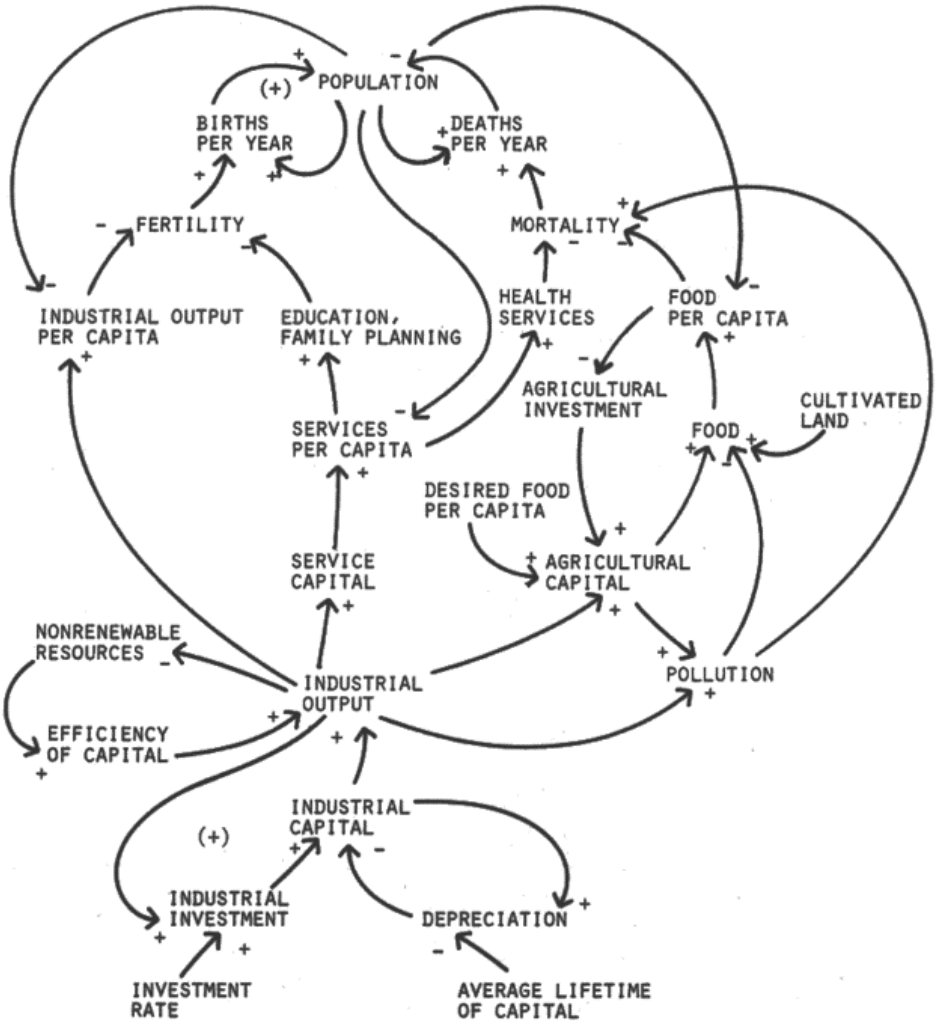


Figure 11: Important Feedback Cycles in the World3 Model

Appendix B: FAO Methodology

The FAO data is mainly provided by UNSD, Eurostat, and other national authorities as needed. This source data is checked for outliers, trade partner data is used for non-reporting countries or missing cells, and data on food aid is added to take into account total cross-border trade flows. All crops and livestock products were registered by the customs office in the country. The full methodology can be found here: http://www.fao.org/cwp-on-fishery-statistics/handbook/introduction/methodology-for-data-collection/en/

Appendix C: GFN Methodology

The GFN calculations utilitze datasets published by the Food and Agriculture Organization, United Nations Commodity Trade Statistics Database, and the UN Statistics Division, as well as the International Energy Agency. Supplementary data sources include studies in peer-reviewed science journals and thematic collections. Of the countries, territories, and regions analyzed in the National Footprint Accounts, 150 had populations over one million and typically have more complete and reliable data sets.

Ecological footprint is derived by using the yields of primary products (from cropland, forest, grazing land and fisheries) to calculate the area necessary to support a given activity. Biocapacity is measured by calculating the amount of biologically productive land and sea area available to provide the resources a population consumes and to absorb its wastes, given current technology and management practices. The full methodology can be found here: https://www.footprintnetwork.org/resources/data/

Appendix D: OECD Methodology

The OECD indices reference data from the Main Economic Indicators database. The weights used to combine country data in forming aggregates differ depending on the subject. For example, Industrial Production, Composite Leading Indicators, Retail Trade, Hourly Earnings, Consumer Price Indices, Producer Price Indices and Monetary Aggregates all use relevant weights for the most recently available year (e.g. GDP) adjusted by Purchasing Power Parities. Other subject aggregates, for example Labour Statistics, Balance of Payments and International Trade are simply based on the sum of contributing countries data. The full methodology can be found here: https://www.oecd.org/sdd/business-stats/2073632.pdf

Appendix E: World Bank Methodology

The World Bank's database is a compilation of primary sources, including national statistical agencies, central banks, and customs services. Considerable effort has been made to standardize the data, but full comparability cannot be assured, so care must be taken in interpreting the indicators. Aggregates are sums of available data. Missing values are not imputed. The full methodology can be found here: http://datatopics.worldbank.org/world-development-indicators/sources-and-methods.html/

Appendix F: Data Pre-Processing Code

```
library(dplyr)
load("IMMC_Data.RData")

emissions_agri_dfs <- split(emissions_agri, emissions_agri$Element)
for (i in 1:length(emissions_agri_dfs)){
  if (length(unique(emissions_agri_dfs[[i]][["Area"]])) > 1){
    emissions_agri_dfs[[i]] <- select(emissions_agri_dfs[[i]], c(Year, Area,
       ↪  Element, Item, Value, Unit))
  }
  else if (length(unique(emissions_agri_dfs[[i]][["Country"]])) > 1){
    emissions_agri_dfs[[i]] <- rename(select(emissions_agri_dfs[[i]], c(Year
       ↪ , Country, Element, Item, Value, Unit)), Area=Country)
  }
}
emissions_landuse_dfs <- split(emissions_landuse, emissions_landuse$Element)
for (i in 1:length(emissions_landuse_dfs)){
  if (length(unique(emissions_landuse_dfs[[i]][["Area"]])) > 1){
    emissions_landuse_dfs[[i]] <- select(emissions_landuse_dfs[[i]], c(Year,
       ↪  Area, Element, Item, Value, Unit))
  }
  else if (length(unique(emissions_landuse_dfs[[i]][["Country"]])) > 1){
    emissions_landuse_dfs[[i]] <- rename(select(emissions_landuse_dfs[[i]],
       ↪ c(Year, Country, Element, Item, Value, Unit)), Area=Country)
  }
}
environment_dfs <- split(environment, environment$Element)
for (i in 1:length(environment_dfs)){
  if (length(unique(environment_dfs[[i]][["Area"]])) > 1){
    environment_dfs[[i]] <- select(environment_dfs[[i]], c(Year, Area,
       ↪ Element, Item, Value, Unit, Months))
  }
  else if (length(unique(environment_dfs[[i]][["Country"]])) > 1){
    environment_dfs[[i]] <- rename(select(environment_dfs[[i]], c(Year,
       ↪ Country, Element, Item, Value, Unit, Months)), Area=Country)
  }
}
foodbal_dfs <- split(foodbal, foodbal$Element)
for (i in 1:length(emissions_agri_dfs)){
```

```
  if (length(unique(emissions_agri_dfs[[i]][["Area"]])) > 1){
    emissions_agri_dfs[[i]] <- select(emissions_agri_dfs[[i]], c(Year, Area,
        ↪  Element, Item, Value, Unit))
  }
  else if (length(unique(emissions_agri_dfs[[i]][["Country"]])) > 1){
    emissions_agri_dfs[[i]] <- rename(select(emissions_agri_dfs[[i]], c(Year
        ↪ , Country, Element, Item, Value, Unit)), Area=Country)
  }
}
foodsupply_dfs <- split(foodsupply, foodsupply$Element)
for (i in 1:length(foodsupply_dfs)){
  if (length(unique(foodsupply_dfs[[i]][["Area"]])) > 1){
    foodsupply_dfs[[i]] <- select(foodsupply_dfs[[i]], c(Year, Area, Element
        ↪ , Item, Value, Unit))
  }
  else if (length(unique(foodsupply_dfs[[i]][["Country"]])) > 1){
    foodsupply_dfs[[i]] <- rename(select(foodsupply_dfs[[i]], c(Year,
        ↪ Country, Element, Item, Value, Unit)), Area=Country)
  }
}
forestry_dfs <- split(forestry, forestry$Element)
for (i in 1:length(forestry_dfs)){
  if (length(unique(forestry_dfs[[i]][["Area"]])) > 1){
    forestry_dfs[[i]] <- select(forestry_dfs[[i]], c(Year, Area, Element,
        ↪ Item, Value, Unit))
  }
  else if (length(unique(forestry_dfs[[i]][["Country"]])) > 1){
    forestry_dfs[[i]] <- rename(select(forestry_dfs[[i]], c(Year, Country,
        ↪ Element, Item, Value, Unit)), Area=Country)
  }
}
population_dfs <- split(population, population$Element)
for (i in 1:length(population_dfs)){
  if (length(unique(population_dfs[[i]][["Area"]])) > 1){
    population_dfs[[i]] <- select(population_dfs[[i]], c(Year, Area, Element
        ↪ , Item, Value, Unit))
  }
  else if (length(unique(population_dfs[[i]][["Country"]])) > 1){
```

```
    population_dfs[[i]] <- rename(select(population_dfs[[i]], c(Year,
        ↪ Country, Element, Item, Value, Unit)), Area=Country)
  }
}
prod_dfs <- split(prod, prod$Element)
for (i in 1:length(prod_dfs)){
  var.name <- paste0(names(prod_dfs)[[i]])
  if (length(unique(prod_dfs[[i]][["Area"]])) > 1){
    prod_dfs[[i]] <- rename(select(prod_dfs[[i]], c(Year, Country, Item,
        ↪ Value)), paste0(names(prod_dfs)[[i]], "␣(", head(prod_dfs[[i]][["
        ↪ Unit"]]), ")")=Value)
  }
  else if (length(unique(prod_dfs[[i]][["Country"]])) > 1){
    prod_dfs[[i]] <- rename(select(prod_dfs[[i]], c(Year, Country, Item,
        ↪ Value)), Area=Country, paste0(names(prod_dfs)[[i]], "␣(", prod_dfs
        ↪ [[i]][["Unit"]][[1]], ")")=Value)
  }
}


rm(emissions_agri, emissions_landuse, environment, foodbal, foodsupply,
    ↪ forestry, population, prod)
rm(asti, commodity, cpi, deflators, employment, exchange, foodaidshipments,
    ↪ foodsecdata, inputs, investment, pricesA, pricesE, pricesM, trade,
    ↪ value)

population <- population_dfs[[2]]
glaciers_permasnow <- filter(environment_dfs[[5]], Item=="Permanent␣snow␣and
    ↪ ␣glaciers")
#nh3_emissions <- environment_dfs[[1]]
#water_withdraw <- environment_dfs[[2]]
#carbon_soil <- environment_dfs[[7]]

#land_degradation <- environment_dfs[[8]]

#soil_erosion <- environment_dfs[[9]]
#bioenergy_percent <- environment_dfs[[10]]
livestock_density <- filter(environment_dfs[[11]], Item=="Major␣livestock␣
    ↪ types")
```

```
co2eq_total <- filter(environment_dfs[[12]], Item=="Sources␣total")
co2eq_from_agriculture <- filter(environment_dfs[[12]], Item=="Agriculture␣
   ↪ total")
co2eq_from_energy_prod <- filter(environment_dfs[[12]], Item=="Energy␣total"
   ↪ )
co2eq_from_land_use <- filter(environment_dfs[[12]], Item=="Land␣use␣sources
   ↪ ")
co2eq_from_forests <- filter(environment_dfs[[12]], Item=="Forest")
co2eq_from_waste <- filter(environment_dfs[[12]], Item=="Waste")
co2eq_from_other <- filter(environment_dfs[[12]], Item=="Other␣sources")
co2eq_from_transport <- filter(environment_dfs[[12]], Item=="Transport")
avg_nitrogen_per_cropland <- filter(environment_dfs[[43]], Item=="Nutrient␣
   ↪ nitrogen␣N␣(total)")
avg_phosphate_per_cropland <- filter(environment_dfs[[43]], Item=="Nutrient␣
   ↪ phosphate␣P2O5␣(total)")
avg_potash_per_cropland <- filter(environment_dfs[[43]], Item=="Nutrient␣
   ↪ potash␣K2O␣(total)")
avg_pesticides_per_cropland <- filter(environment_dfs[[43]], Item=="
   ↪ Pesticides␣(total)")
#std_dev_temp <- environment_dfs[[40]]
#temp_change <- environment_dfs[[42]]
kcal_per_person_per_day <- filter(foodbal_dfs[["Food␣supply␣(kcal/capita/day
   ↪ )"]], Item=="Grand␣Total")

avg_nitrogen_per_cropland <- rename(select(avg_nitrogen_per_cropland, c(Year
   ↪ , Area, Value)), "Use␣of␣Nitrogen␣per␣area␣of␣cropland␣(kg/ha)"=Value
   ↪ )
avg_pesticides_per_cropland <- rename(select(avg_pesticides_per_cropland, c(
   ↪ Year, Area, Value)), "Use␣of␣Pesticides␣per␣area␣of␣cropland␣(kg/ha)"
   ↪ =Value)
avg_phosphate_per_cropland <- rename(select(avg_phosphate_per_cropland, c(
   ↪ Year, Area, Value)), "Use␣of␣Phosphate␣per␣area␣of␣cropland␣(kg/ha)"=
   ↪ Value)
avg_potash_per_cropland <- rename(select(avg_potash_per_cropland, c(Year,
   ↪ Area, Value)), "Use␣of␣Potash␣per␣area␣of␣cropland␣(kg/ha)"=Value)
glaciers_permasnow <- rename(select(glaciers_permasnow, c(Year, Area, Value)
   ↪ ), "Area␣with␣Permanent␣Snow/Glaciers␣(CCI-LC,␣1000␣ha)"=Value)
#water_withdraw <- rename(select(water_withdraw, c(Year, Area, Value)), "%
   ↪  of total withdrawl used for agriculture"=Value)
```

```
#carbon_soil <- rename(select(carbon_soil, c(Year, Area, Value)), "Average
    ↪  carbon content in topsoil (% weight)"=Value)
#soil_erosion <- rename(select(soil_erosion, c(Year, Area, Value)), "
    ↪ Average soil erosion (GLASOD erosion degree)"=Value)
#land_degradation <- rename(select(land_degradation, c(Year, Area, Value))
    ↪ , "Average land degradation (GLASOD erosion degree)"=Value)
livestock_density <- rename(select(livestock_density, c(Year, Area, Value)),
    ↪  "Major␣livestock␣density␣in␣agricultural␣areas␣(LSU/ha)"=Value)
co2eq_total <- rename(select(co2eq_total, c(Year, Area, Value)), "Total␣
    ↪ Emissions␣of␣CO2eq␣(gigagrams)"=Value)
co2eq_from_waste <- rename(select(co2eq_from_waste, c(Year, Area, Value)), "
    ↪ Emissions␣of␣CO2eq␣from␣Waste␣(gigagrams)"=Value)
co2eq_from_energy_prod <- rename(select(co2eq_from_energy_prod, c(Year, Area
    ↪ , Value)), "Emissions␣of␣CO2eq␣from␣Energy␣Production␣(gigagrams)"=
    ↪ Value)
co2eq_from_forests <- rename(select(co2eq_from_forests, c(Year, Area, Value)
    ↪ ), "Emissions␣of␣CO2eq␣from␣Forests␣(gigagrams)"=Value)
co2eq_from_land_use <- rename(select(co2eq_from_land_use, c(Year, Area,
    ↪ Value)), "Emissions␣of␣CO2eq␣from␣Land␣Use␣Sources␣(gigagrams)"=Value
    ↪ )
co2eq_from_other <- rename(select(co2eq_from_other, c(Year, Area, Value)), "
    ↪ Emissions␣of␣CO2eq␣from␣Other␣Sources␣(gigagrams)"=Value)
co2eq_from_transport <- rename(select(co2eq_from_transport, c(Year, Area,
    ↪ Value)), "Emissions␣of␣CO2eq␣from␣Transportation␣(gigagrams)"=Value)
co2eq_from_agriculture <- rename(select(co2eq_from_agriculture, c(Year, Area
    ↪ , Value)), "Emissions␣of␣CO2eq␣from␣Agriculture␣(gigagrams)"=Value)
temp_change <- rename(aggregate(temp_change["Value"], by=temp_change[c("Year
    ↪ ", "Area")], function(x){sum(x)/12}), "Temperature␣change␣(Celsius/
    ↪ year)"=Value)
#std_dev_temp <- rename(select(std_dev_temp, c(Year, Area, Value)), "
    ↪ Standard Deviation of Temperature (Celsius)"=Value)
kcal_per_person_per_day <- rename(select(kcal_per_person_per_day, c(Year,
    ↪ Area, Value)), "Food␣supply␣(kcal/capita/day)"=Value)
population <- rename(select(filter(population, Year > 1960 & Year < 2017 &
    ↪ Area == "World"), c(Year, Area, Value)), "Population␣(1000␣persons)"=
    ↪ Value)

final_list <- list(population, glaciers_permasnow, livestock_density, co2eq_
    ↪ total, co2eq_from_agriculture, co2eq_from_energy_prod, co2eq_from_
```

```
   ↪ land_use, co2eq_from_forests, co2eq_from_waste, co2eq_from_other,
   ↪ co2eq_from_transport, avg_nitrogen_per_cropland, avg_phosphate_per_
   ↪ cropland, avg_potash_per_cropland, avg_pesticides_per_cropland, kcal_
   ↪ per_person_per_day)

final_df <- Reduce(function(x, y) merge(x, y, by=c("Year", "Area"), all=TRUE
   ↪ ), final_list)
#final_df <- filter(final_df, Year > 1990 & Year < 2015)
world_df <- aggregate(select(final_df, -c("Area", "Year")), by=final_df["
   ↪ Year"], function(x){sum(x, na.rm=TRUE)})

industrial_oecd <- read.csv("DP_LIVE_19042019021654237.csv")
world_df[["Industrial␣Production␣(%␣Relative␣to␣2015)"]] <- c(rep(0,14),
   ↪ filter(industrial_oecd, LOCATION=="OECD")$Value)

aquastat_tables <- read.csv("aquastat_tables.csv")
aquastat_tables <- aquastat_tables %>% dplyr::filter(row_number() %% 6 != 5
   ↪ & row_number() %% 6 != 4 & row_number() %% 6 != 0)
avg_precipitation <- select(aquastat_tables %>% filter(row_number() %% 3 ==
   ↪ 1), c(X1978.1982, X1983.1987, X1988.1992, X1993.1997, X1998.2002,
   ↪ X2003.2007, X2008.2012, X2013.2017))
total_renewable_rscs <- select(aquastat_tables %>% filter(row_number() %% 3
   ↪ == 2), c(X1978.1982, X1983.1987, X1988.1992, X1993.1997, X1998.2002,
   ↪ X2003.2007, X2008.2012, X2013.2017))
total_renewable_rscs_per_capita <- select(aquastat_tables %>% filter(row_
   ↪ number() %% 3 == 0), c(X1978.1982, X1983.1987, X1988.1992, X1993
   ↪ .1997, X1998.2002, X2003.2007, X2008.2012, X2013.2017))
avg_precipitation[avg_precipitation == ""] <- NA
total_renewable_rscs[total_renewable_rscs == ""] <- NA
total_renewable_rscs_per_capita[total_renewable_rscs_per_capita == ""] <- NA
avg_precipitation <- colSums(sapply(avg_precipitation, as.numeric), na.rm=
   ↪ TRUE)
total_renewable_rscs <- colSums(sapply(total_renewable_rscs, as.numeric), na
   ↪ .rm=TRUE)
total_renewable_rscs_per_capita <- colSums(sapply(total_renewable_rscs_per_
   ↪ capita, as.numeric), na.rm=TRUE)
world_precip <- rep(NA, 39)
world_renewable_water <- rep(NA, 39)
world_rn_water_per_capita <- rep(NA, 39)
```

```
for (i in 1:39){
  world_precip[[i]] <- avg_precipitation[[ceiling(i/5)]]
  world_renewable_water[[i]] <- total_renewable_rscs[[ceiling(i/5)]]
  world_rn_water_per_capita[[i]] <- total_renewable_rscs_per_capita[[ceiling
    ↪ (i/5)]]
}
world_df[["Average␣Annual␣Precipitation␣(mm/year)"]] <- c(rep(0, 17), world_
  ↪ precip)
world_df[["Total␣Renewable␣Water␣Resources␣(10^9␣m^3/year)"]] <- c(rep(0,
  ↪ 17), world_renewable_water)
world_df[["Total␣Renewable␣Water␣Resources␣Per␣Capita␣(m^3/capita/year)"]]
  ↪ <- c(rep(0, 17), world_rn_water_per_capita)

biocapacity <- read.csv("Biocapacity.csv")
eco_footprint <- read.csv("ecological_footprint.csv")
biocapacity <- rename(select(biocapacity, -c(Country.Name, Short.Name,
  ↪ Record, Data.Quality.Score, isoa2)), Year=year, "Built-up␣Land␣(
  ↪ Biocapacity,␣gha)"=Built.up.Land, "Carbon␣(Biocapacity,␣gha)"=Carbon,
  ↪  "Cropland␣(Biocapacity,␣gha)"=Cropland, "Fishing␣Grounds␣(
  ↪ Biocapacity,␣gha)"=Fishing.Grounds, "Forest␣Products␣(Biocapacity,␣
  ↪ gha)"=Forest.Products, "Grazing␣Land␣(Biocapacity,␣gha)"=Grazing.Land
  ↪ , "Total␣(Biocapacity,␣gha)"=Total)
eco_footprint <- rename(select(eco_footprint, -c(Country.Name, Short.Name,
  ↪ Record, Data.Quality.Score, isoa2)), Year=year, "Built-up␣Land␣(Eco.␣
  ↪ Footprint,␣gha)"=Built.up.Land, "Carbon␣(Eco.␣Footprint,␣gha)"=Carbon
  ↪ , "Cropland␣(Eco.␣Footprint,␣gha)"=Cropland, "Fishing␣Grounds␣(Eco.␣
  ↪ Footprint,␣gha)"=Fishing.Grounds, "Forest␣Products␣(Eco.␣Footprint,␣
  ↪ gha)"=Forest.Products, "Grazing␣Land␣(Eco.␣Footprint,␣gha)"=Grazing.
  ↪ Land, "Total␣(Eco.␣Footprint,␣gha)"=Total)
world_df <- merge(merge(world_df, eco_footprint, by="Year", all=TRUE),
  ↪ biocapacity, all=TRUE)

birth_rate <- read.csv("Birth␣Rate.csv", header=FALSE)
death_rate <- read.csv("Death␣Rate.csv", header=FALSE)
brdr_header <- c("Country␣Name", "Country␣Code", "Indicator␣Name", "
  ↪ Indicator␣Code", paste0(rep("Y", 60), 1960:2019))
names(birth_rate) <- brdr_header -> names(death_rate)
birth_rate <- select(filter(birth_rate, `Country Name`=="World"), paste0(rep
  ↪ ("Y", 56), 1961:2016))
```

```
death_rate <- select(filter(death_rate, `Country Name`=="World"), paste0(rep
    ↪ ("Y", 56), 1961:2016))
world_df[["Birth␣Rate,␣Crude␣(annually␣per␣1000␣people)"]] <- t(birth_rate)
world_df[["Death␣Rate,␣Crude␣(annually␣per␣1000␣people)"]] <- t(death_rate)

write.csv(world_df, "World_Data.csv")

#<- select(, c(Year, Area, Element, Item, Value, Unit))
#_df <- split(, $Element)
```

Appendix G: Variables List

In addition to year and population, following variables were considered in our model:

1. Major Livestock Density in Agricultural Areas (LSU/ha)
2. Total Emiggions of CO2eq (gigagrams)
3. Emissions of CO2 from Agriculture, Energy Production, Land Use Sources, Forests, Waste, Transportation, and Other Sources (gigagrams)
4. Use of Nitrogen, Phosphate, Potash, and Pesticide per Area of Cropland (kg/ha)
5. Food Supply (kcal/capita/day)
6. Industrial Production (% Relative to 2015)
7. Average Annual Percipitation (mm/year)
8. Total Renewable water Resources ($10\hat{9}$ $m\hat{3}$/year)
9. Total Renewable water Resources per Capita ($m\hat{3}$/capita/year)
10. Total Ecological Footprint (gha)
11. Ecological Footprint of Built-up Land, Carbon, Cropland, Fishing Grounds, Forest Products, and Grazing Land
12. Total Biocapicity (gha)
13. Biocapacity of Built-up Land, Carbon, Cropland, Fishing Grounds, Forest Products, and Grazing Land (gha)
14. Birth Rate (annually per 1000 people)
15. Death Rate (annually per 1000 people)

Appendix H: Spearman's Correlation Program

```
import multiprocessing as mp
import random
import time
import numpy as np
from scipy.stats import spearmanr as sp

# Make all combinations for Spearman
fins = []
for col in df:
    for sec_col in df:
        tot = [col, sec_col]
        tot.sort()

        fins.append(tot)

# Remove duplicates
proper = list(set(map(tuple,fins)))

split_ways = 14
final = [[] for i in range(0, split_ways)]

for i in range(0, len(proper)):
    final[i%split_ways].append(proper[i])

print(np.asarray(final).shape)



print(mp.cpu_count())

global alls
alls = []

def spearman(L, df, arra):
    for group in arra:
        row = []
        row.append(group[0])
        row.append(group[1])
```

```
        rho, p = sp(df[group[0]], df[group[1]])
        row.append(rho)
        L.append(row)

procc = []

start = time.time()

print("Started")
with mp.Manager() as manager:
    L = manager.list()
    for i in range(0, 14):
        procc.append(mp.Process(target=spearman, args=(L, df, final[i])))
        procc[i].start()

    for i in range(0, 14):
        procc[i].join()

    alls = list(L)

print("Finishing")

end = time.time()
print("Multiprocessing␣Time:␣"+str(end-start))

pd.DataFrame(np.asarray(alls)).to_csv("results.csv")
```

Appendix I: Graph Visualization Program

Package is available at: https://briatte.github.io/ggnet/

```
pkg <- c("ggplot2", "GGally", "network", "sna", "dplyr", "tidyverse")
new.pkg <- pkg[!(pkg %in% installed.packages())]
if (length(new.pkg)) {
  install.packages(new.pkg, repos = "http://cran.rstudio.com")
}
library(GGally)
library(network)
```

```
library(sna)
library(ggplot2)
library(dplyr)
library(tidyverse)

q <- 0.70 #Input threshold for Spearman's correlation

spearm <- rename(select(read.csv("results.csv"), -X), Var1=X0, Var2=X1,
    ↪ Correlation=X2)

spearm[[1]] <- as.character(spearm[[1]])
spearm[[2]] <- as.character(spearm[[2]])
spearm[[3]] <- as.numeric(spearm[[3]])
nodes <- data.frame(stat=c(unique(union(spearm[[1]], spearm[[2]]))))
nodes <- nodes %>% rowid_to_column("id")
nodes[[1]] <- as.numeric(nodes[[1]])
nodes[[2]] <- as.character(nodes[[2]])
pairs <- spearm %>% filter(abs(Correlation) > q) %>% group_by(Var1, Var2)
    ↪ %>% ungroup()
edges <- pairs %>% left_join(nodes, by = c("Var1"="stat")) %>% rename(from=
    ↪ id)
edges <- edges %>% left_join(nodes, by = c("Var2"="stat")) %>% rename(to=id)
edges <- select(edges, from, to, Correlation)

stat_network <- network(edges, vertex.attr = nodes, matrix.type="edgelist",
    ↪ ignore_eval=FALSE, directed=FALSE)

ggnet2(stat_network, node.size = 6, edge.size=1, edge.color='grey', mode='
    ↪ circle', color="stat", color.legend="Statistic␣(q-value␣=␣0.85)",
        palette = c("Average␣Annual␣Precipitation␣(mm/year)"="#d6a46e",
                    "Cropland␣(Eco.␣Footprint,␣gha)"="#685ed3",
                    "Emissions␣of␣CO2eq␣from␣Forests␣(gigagrams)"="#9ebe35",
                    "Emissions␣of␣CO2eq␣from␣Agriculture␣(gigagrams)"="#a33caa
                        ↪ ",
                    "Built-up␣Land␣(Biocapacity,␣gha)"="#62c655",
                    "Emissions␣of␣CO2eq␣from␣Waste␣(gigagrams)"="#d56ad9",
                    "Emissions␣of␣CO2eq␣from␣Transportation␣(gigagrams)"="#509
                        ↪ b2e",
                    "Industrial␣Production␣(%␣Relative␣to␣2015)"="#cd4a97",
```

```
"Food␣supply␣(kcal/capita/day)"="#54bd7b",
"Major␣livestock␣density␣in␣agricultural␣areas␣(LSU/ha)"="
    ↪ #da4376",
"Area␣with␣Permanent␣Snow/Glaciers␣(CCI-LC,␣1000␣ha)"="#55
    ↪ c8b0",
"Total␣Renewable␣Water␣Resources␣(10^9␣m^3/year)"="#ce364a
    ↪ ",
"Forest␣Products␣(Eco.␣Footprint,␣gha)"="#46aed7",
"Carbon␣(Eco.␣Footprint,␣gha)"="#cb4626",
"Death␣Rate,␣Crude␣(annually␣per␣1000␣people)"="#528fdf",
"Cropland␣(Biocapacity,␣gha)"="#e08f2e",
"Forest␣Products␣(Biocapacity,␣gha)"="#907fe0",
"Grazing␣Land␣(Eco.␣Footprint,␣gha)"="#aead3b",
"Built-up␣Land␣(Eco.␣Footprint,␣gha)"="#8a549b",
"Total␣Emissions␣of␣CO2eq␣(gigagrams)"="#cea339",
"Emissions␣of␣CO2eq␣from␣Land␣Use␣Sources␣(gigagrams)"="
    ↪ #5162a5",
"Birth␣Rate,␣Crude␣(annually␣per␣1000␣people)"="#e87c47",
"Grazing␣Land␣(Biocapacity,␣gha)"="#3a9a87",
"Emissions␣of␣CO2eq␣from␣Other␣Sources␣(gigagrams)"="#
    ↪ d4645d",
"Total␣(Eco.␣Footprint,␣gha)"="#488133",
"Total␣Renewable␣Water␣Resources␣Per␣Capita␣(m^3/capita/
    ↪ year)"="#d28cd4",
"Emissions␣of␣CO2eq␣from␣Energy␣Production␣(gigagrams)"="
    ↪ #6c751f",
"Total␣(Biocapacity,␣gha)"="#9a9cdb",
"Fishing␣Grounds␣(Biocapacity,␣gha)"="#a95f26",
"Population␣(1000␣persons)"="#347c52",
"Fishing␣Grounds␣(Eco.␣Footprint,␣gha)"="#ee91aa",
"Use␣of␣Potash␣per␣area␣of␣cropland␣(kg/ha)"="#9db269",
"Use␣of␣Nitrogen␣per␣area␣of␣cropland␣(kg/ha)"="#96405f",
"Use␣of␣Phosphate␣per␣area␣of␣cropland␣(kg/ha)"="#627037",
"Carbon␣(Biocapacity,␣gha)"="#c26d8a",
"Use␣of␣Pesticides␣per␣area␣of␣cropland␣(kg/ha)"="#8c6d2c"
    ↪ ,
"Year"="#b26850"))
```

Appendix J: Sample of World4 Code

The full World4 program is available for download at https://github.com/gautomdas/IMMC-World4.

```
# Model for World4
!pip install statsmodels
## Get Original Data Again
import pandas as pd
from scipy.stats import spearmanr as sp

data = open('World_Data.csv', 'r', encoding="ISO-8859-1");
# Read Bryan's file for now we will use dummy data
df = pd.read_csv(data)
a = [column for column in df]
#print(a)

del df[a[0]]
#df.head()

# Flow Chart Methods


## Industrial Ouput

'''
@proof: industrial_output
@type: linear
Proof or relationship and values
'''

x_col = 'Population (1000 persons)'
y_col = 'Industrial Production ( % Relative to 2015)'
k = df[y_col]
print(len(k))
fin = []
sec = []
count = 0
for val in k:
```

```
    if(int(val)!=0):
        fin.append(val)
        sec.append(df[x_col][count])
    count+=1
from scipy import stats
io_slope, io_intercept, r_value, p_value, std_err = stats.linregress(sec,
    ↪ fin)
print("Equation:␣␣␣y␣=␣"+str(io_slope)+"x"+"␣+␣"+str(io_intercept))
print("R-Squared:␣"+str(r_value))

%matplotlib inline

import matplotlib.pyplot as plt
import numpy as np
import pandas as pd

plt.plot(sec, fin)

plt.xlabel(x_col)
plt.ylabel(y_col)
plt.title(y_col)


'''
@name: industrial_output

Industrial output exihibits a linear relationship but does not have any
feeder variables and is thus a function of population.
'''

def industrial_output(population):
    return io_slope*population+io_intercept

plt.plot(sec, [industrial_output(val) for val in sec])
plt.show()

## CO2 Ouput

'''
```

```
@proof: co2_energy
@type: linear
Proof or relationship and values
'''

x_col = 'Industrial␣Production␣(%␣Relative␣to␣2015)'
y_col = 'Emissions␣of␣CO2eq␣from␣Energy␣Production␣(gigagrams)'
k = df[y_col]
print(len(k))
fin = []
sec = []
count = 0
for val in k:
    if(int(val)!=0):
        fin.append(val)
        sec.append(df[x_col][count])
    count+=1


%matplotlib inline

import matplotlib.pyplot as plt
import numpy as np
import pandas as pd

plt.scatter(sec, fin)

from scipy import stats
ce_slope, ce_intercept, r_value, p_value, std_err = stats.linregress(sec,
    ↪ fin)
print("Equation:␣␣␣y␣=␣"+str(ce_slope)+"x"+"␣+␣"+str(ce_intercept))
print("R-Squared:␣"+str(r_value))

plt.xlabel('Population␣(1000␣persons)')
plt.ylabel(y_col)
plt.title(y_col)


'''
```

```
@name: co2_energy

Industrial output exihibits a linear relationship but does not have any
feeder variables and is thus a function of population.
'''

def co2_industry(industrial):
    return ce_slope*industrial+ce_intercept

plt.plot(sec, [co2_industry(val) for val in sec])
plt.show()

## CO2 Ouput Cont.

'''
@proof: co2_transpo
@type: linear
Proof or relationship and values
'''

x_col = 'Industrial␣Production␣(%␣Relative␣to␣2015)'
y_col = 'Emissions␣of␣CO2eq␣from␣Transportation␣(gigagrams)'
k = df[y_col]
print(len(k))
fin = []
sec = []
count = 0
for val in k:
    if(int(val)!=0):
        fin.append(val)
        sec.append(df[x_col][count])
    count+=1


%matplotlib inline

import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
```

```
plt.scatter(sec, fin)

from scipy import stats
ct_slope, ct_intercept, r_value, p_value, std_err = stats.linregress(sec,
    ↪ fin)
print("Equation:␣␣␣y␣=␣"+str(ct_slope)+"x"+"␣+␣"+str(ct_intercept))
print("R-Squared:␣"+str(r_value))

plt.xlabel('Population␣(1000␣persons)')
plt.ylabel(y_col)
plt.title(y_col)


'''
@name: co2_transpo

Industrial output exihibits a linear relationship but does not have any
feeder variables and is thus a function of population.
'''

def co2_transpo(industrial):
    return ct_slope*industrial+ct_intercept

plt.plot(sec, [co2_transpo(val) for val in sec])
plt.show()

## CO2 Ouput Cont.

'''
@proof: co2_waste
@type: linear
Proof or relationship and values
'''

x_col = 'Population␣(1000␣persons)'
y_col = 'Emissions␣of␣CO2eq␣from␣Waste␣(gigagrams)'
k = df[y_col]
print(len(k))
```

```
fin = []
sec = []
count = 0
for val in k:
    if(int(val)!=0):
        fin.append(val)
        sec.append(df[x_col][count])
    count+=1


%matplotlib inline

import matplotlib.pyplot as plt
import numpy as np
import pandas as pd

plt.scatter(sec, fin)

from scipy import stats
cw_slope, cw_intercept, r_value, p_value, std_err = stats.linregress(sec,
    ↪ fin)
print("Equation:␣␣␣y␣=␣"+str(cw_slope)+"x"+"␣+␣"+str(cw_intercept))
print("R-Squared:␣"+str(r_value))

plt.xlabel('Population␣(1000␣persons)')
plt.ylabel(y_col)
plt.title(y_col)


'''
@name: co2_waste

Industrial output exihibits a linear relationship but does not have any
feeder variables and is thus a function of population.
'''

def co2_waste(popu):
    return cw_slope*popu+cw_intercept
```

```
plt.plot(sec, [co2_waste(val) for val in sec])
plt.show()

## Nitrogen

'''
@proof: nitrogen
@type: logistic
Proof or relationship and values
'''

x_col = 'Population␣(1000␣persons)'
y_col = 'Use␣of␣Nitrogen␣per␣area␣of␣cropland␣(kg/ha)'
k = df[y_col]
print(len(k))
fin = []
sec = []
count = 0
for val in k:
    if(int(val)!=0):
        fin.append(val)
        sec.append(df[x_col][count]/1000000)
    count+=1


%matplotlib inline

import scipy.optimize as opt
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit
import numpy as np
import pandas as pd
from sklearn.linear_model import LogisticRegression

def funcT(x, a, b):
    return (2000/ (1 + np.exp(a + b * x))) + 9500

aas, pcov = curve_fit(funcT, sec, fin)
```

```
plt.scatter(sec, fin)

plt.xlabel('Population␣(1000␣persons)')
plt.ylabel(y_col)
plt.title(y_col)


'''
@name: nitrogen

Industrial output exihibits a linear relationship but does not have any
feeder variables and is thus a function of population.
'''

def nitrogen(population):
    population = population/1000000
    return funcT(population, aas[0], aas[1])

print(sec)
plt.plot(sec, [ nitrogen(val*1000000) for val in sec])
plt.show()


## Phosphorus

'''
@proof: phosphorus
@type: log curve
Proof or relationship and values
'''

x_col = 'Population␣(1000␣persons)'
y_col = 'Use␣of␣Phosphate␣per␣area␣of␣cropland␣(kg/ha)'
k = df[y_col]
print(len(k))
fin = []
sec = []
```

```
count = 0
for val in k:
    if(int(val)!=0):
        fin.append(val)
        sec.append(df[x_col][count]/1000000)
    count+=1


%matplotlib inline
import scipy
import scipy.optimize as opt
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from sklearn.linear_model import LogisticRegression

def logFunc(x, a, b):
    return a + b*np.log(x)

#p1, pcov = curve_fit(func, sec, fin)
popt, poc = scipy.optimize.curve_fit(logFunc, sec, fin)
print(popt)

plt.scatter(sec, fin)

plt.xlabel('Population (1000 persons)')
plt.ylabel(y_col)
plt.title(y_col)


'''
@name: phosphorus

Industrial output exihibits a linear relationship but does not have any
feeder variables and is thus a function of population.
'''

def phosphorus(population):
    population = population
```

```
    return popt[0]+np.log(population)*popt[1]

plt.plot(sec, [ phosphorus(val) for val in sec])
plt.show()

## Built Up Land

'''
@proof: builtUp
@type: polynomial 2
Proof or relationship and values
'''

x_col = 'Industrial␣Production␣(%␣Relative␣to␣2015)'
y_col = 'Built-up␣Land␣(Eco.␣Footprint,␣gha)'
k = df[y_col]
print(len(k))
fin = []
sec = []
count = 0
for val in k:
    if(int(val)!=0 and df[x_col][count]!=0):
        fin.append(val)
        sec.append(df[x_col][count])
    count+=1


%matplotlib inline

import scipy.optimize as opt
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from sklearn.linear_model import LogisticRegression

z = np.polyfit(sec, fin, 2)
pBuild = np.poly1d(z)

plt.scatter(sec, fin)
```

```
plt.xlabel('Population␣(1000␣persons)')
plt.ylabel(y_col)
plt.title(y_col)


'''
@name: forest

Industrial output exihibits a linear relationship but does not have any
feeder variables and is thus a function of population.
'''

def builtUp(industrial_out):
    return pBuild(industrial_out)

plt.plot(sec, [ pBuild(val) for val in sec])
plt.show()

## Forest

'''
@proof: forest
@type: polynomial 2
Proof or relationship and values
'''

x_col = 'Built-up␣Land␣(Eco.␣Footprint,␣gha)'
y_col = 'Forest␣Products␣(Eco.␣Footprint,␣gha)'
k = df[y_col]
print(len(k))
fin = []
sec = []
count = 0
for val in k:
    if(int(val)!=0):
        fin.append(val)
        sec.append(df[x_col][count])
    count+=1
```

```
%matplotlib inline

import scipy.optimize as opt
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from sklearn.linear_model import LogisticRegression

def logFunc(x, a, b):
    return a + b*np.log(x)

#p1, pcov = curve_fit(func, sec, fin)
popt, poc = scipy.optimize.curve_fit(logFunc, sec, fin)
print(popt)

plt.scatter(sec, fin)

plt.xlabel(x_col)
plt.ylabel(y_col)
plt.title(y_col)


'''
@name: forest

Industrial output exihibits a linear relationship but does not have any
feeder variables and is thus a function of population.
'''

def forest(buildUp):
    return popt[0] + popt[1]*np.log(buildUp)

print(popt)
plt.plot(sec, [ forest(val) for val in sec])
plt.show()

## CO2 Forest
```

```
'''
@proof: fc
@type: polynomial 2
Proof or relationship and values
'''

x_col = 'Population␣(1000␣persons)'
y_col = 'Built-up␣Land␣(Eco.␣Footprint,␣gha)'
k = df[y_col]
print(len(k))
fin = []
sec = []
count = 0
for val in k:
    if(int(val)!=0 and df[x_col][count]!=0):
        fin.append(val)
        sec.append(df[x_col][count]/1000000)
    count+=1


%matplotlib inline

import scipy.optimize as opt
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from sklearn.linear_model import LogisticRegression

def logFunc(x, a, b):
    return a + b*np.log(x)

#p1, pcov = curve_fit(func, sec, fin)
popt, poc = scipy.optimize.curve_fit(logFunc, sec, fin)
print(popt)

plt.scatter(sec, fin)

plt.xlabel(x_col)
```

```
plt.ylabel(y_col)
plt.title(y_col)


z = np.polyfit(sec, fin, 2)
pfc = np.poly1d(z)

plt.scatter(sec, fin)


'''
@name: fc

Industrial output exihibits a linear relationship but does not have any
feeder variables and is thus a function of population.
'''

def fc(pop):
    return pfc(pop/1000000)

print(popt)
plt.plot(sec, [ fc(val*1000000) for val in sec])
plt.show()

## Grazing

'''
@proof: grazing
@type: polynomial 9
Proof or relationship and values
'''

x_col = 'Population␣(1000␣persons)'
y_col = 'Grazing␣Land␣(Eco.␣Footprint,␣gha)'
k = df[y_col]
print(len(k))
fin = []
sec = []
count = 0
```

```
for val in k:
    if(int(val)!=0 and df[x_col][count]!=0):
        fin.append(val)
        sec.append(df[x_col][count]/1000000)
    count+=1


%matplotlib inline

import scipy.optimize as opt
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from sklearn.linear_model import LogisticRegression

def logFunc(x, a, b):
    return a + b*np.log(x)

#p1, pcov = curve_fit(func, sec, fin)
popt, poc = scipy.optimize.curve_fit(logFunc, sec, fin)
print(popt)

plt.scatter(sec, fin)

plt.xlabel(x_col)
plt.ylabel(y_col)
plt.title(y_col)


z = np.polyfit(sec, fin, 9)
pfg = np.poly1d(z)

plt.scatter(sec, fin)


'''
@name: grazing

Industrial output exihibits a linear relationship but does not have any
```

```
feeder variables and is thus a function of population.
'''

def grazing(pop):
    return pfg(pop/1000000)

print(popt)
plt.plot(sec, [ grazing(val*1000000) for val in sec])
plt.show()

## Fishing

'''
@proof: fishing
@type: polynomial 7
Proof or relationship and values
'''

x_col = 'Population (1000 persons)'
y_col = 'Fishing Grounds (Eco. Footprint, gha)'
k = df[y_col]
print(len(k))
fin = []
sec = []
count = 0
for val in k:
    if(int(val)!=0 and df[x_col][count]!=0):
        fin.append(val)
        sec.append(df[x_col][count])
    count+=1


%matplotlib inline

import scipy.optimize as opt
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from sklearn.linear_model import LogisticRegression
```

```
def logFunc(x, a, b):
    return a + b*np.log(x)

#p1, pcov = curve_fit(func, sec, fin)
popt, poc = scipy.optimize.curve_fit(logFunc, sec, fin)
print(popt)

plt.scatter(sec, fin)

plt.xlabel(x_col)
plt.ylabel(y_col)
plt.title(y_col)


z = np.polyfit(sec, fin, 7)
pff = np.poly1d(z)

plt.scatter(sec, fin)


'''
@name: fishing

Industrial output exihibits a linear relationship but does not have any
feeder variables and is thus a function of population.
'''

from scipy import stats
aw_slope, aw_intercept, r_value, p_value, std_err = stats.linregress(sec,
    ↪ fin)
print("Equation:␣␣␣y␣=␣"+str(aw_slope)+"x"+"␣+␣"+str(aw_intercept))
print("R-Squared:␣"+str(r_value))

def fishing(pop):
    return pop*aw_slope + aw_intercept

print(popt)
plt.plot(sec, [ fishing(val) for val in sec])
```

```
plt.show()

## Livestock

'''
@proof: livestock
@type: polynomial 7
Proof or relationship and values
'''

x_col = 'Grazing Land (Eco. Footprint, gha)'
y_col = 'Major livestock density in agricultural areas (LSU/ha)'
k = df[y_col]
print(len(k))
x_val = []
y_val = []
count = 0
for val in k:
    if(int(val)!=0 and df[x_col][count]!=0):
        y_val.append(val)
        x_val.append(df[x_col][count])
    count+=1


%matplotlib inline

import scipy.optimize as opt
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from sklearn.linear_model import LogisticRegression

def logFunc(x, a, b):
    return a + b*np.log(x)

#p1, pcov = curve_fit(func, sec, fin)
popt, poc = scipy.optimize.curve_fit(logFunc, x_val, y_val)
print(popt)
```

```
plt.xlabel(x_col)
plt.ylabel(y_col)
plt.title(y_col)


z = np.polyfit(x_val, y_val, 2)
pfl = np.poly1d(z)

plt.scatter(x_val, y_val)


'''
@name: fishing

Industrial output exihibits a linear relationship but does not have any
feeder variables and is thus a function of population.
'''

def livestock(pop):
    return pfl(pop)

print(popt)
plt.plot(x_val, [ livestock(val) for val in x_val])
plt.show()

## CropLand

'''
@proof: cropland
@type: polynomial 7
Proof or relationship and values
'''

y_col = 'Built-up␣Land␣(Eco.␣Footprint,␣gha)'
x_col = 'Cropland␣(Eco.␣Footprint,␣gha)'
k = df[y_col]
print(len(k))
fin = []
```

```
sec = []
count = 0
for val in k:
    if(int(val)!=0 and df[x_col][count]!=0):
        sec.append(val)
        fin.append(df[x_col][count])
    count+=1


%matplotlib inline

import scipy.optimize as opt
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from sklearn.linear_model import LogisticRegression

def logFunc(x, a, b):
    return a + b*np.log(x)

#p1, pcov = curve_fit(func, sec, fin)
popt, poc = scipy.optimize.curve_fit(logFunc, sec, fin)
print(popt)

plt.scatter(sec, fin)

plt.xlabel(y_col)
plt.ylabel(x_col)
plt.title(y_col)


z = np.polyfit(sec, fin, 2)
plc = np.poly1d(z)

plt.scatter(sec, fin)


'''
@name: cropland
```

```
Industrial output exihibits a linear relationship but does not have any
feeder variables and is thus a function of population.
'''

def cropland(pop):
    return plc(pop)

print(popt)
plt.plot(sec, [cropland(val) for val in sec])
plt.show()

## Food Suppyl

'''
@proof: foodsupply
@type: polynomial 7
Proof or relationship and values
'''
x_col = 'Major␣livestock␣density␣in␣agricultural␣areas␣(LSU/ha)'
y_col = 'Fishing␣Grounds␣(Eco.␣Footprint,␣gha)'
z_col = 'Cropland␣(Eco.␣Footprint,␣gha)'

output = 'Food␣supply␣(kcal/capita/day)'

#y_col = 'Use of Pesticides per area of cropland (kg/ha)'
k = df[output]
print(len(k))
out = []
x = []
y = []
z = []

count = 0
for val in k:
    if(int(val)!=0 and df[x_col][count]!=0 and df[y_col][count]!=0 and df[
        ↪ z_col][count]!=0):
        out.append(val)
        x.append(df[x_col][count])
```

```
        y.append(df[y_col][count])
        z.append(df[z_col][count])
    count+=1


from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt
import numpy as np

fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

c = out

img = ax.scatter(x, y, z, c=c, cmap=plt.hot())
fig.colorbar(img)
plt.show()


import statsmodels.api as sm



%matplotlib inline

from sklearn.linear_model import SGDClassifier


import scipy.optimize as opt
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from sklearn import linear_model

clf = linear_model.SGDRegressor(max_iter=10, tol=1e-3)

X = np.array([x, y, z])
X = X.transpose()
y = out
```

```
modelF = sm.OLS(y, X).fit()
predictions = modelF.predict(X) # make the predictions by the model
print(predictions)
# Print out the statistics
print(modelF.summary())



'''
@name: pests

Industrial output exihibits a linear relationship but does not have any
feeder variables and is thus a function of population.
'''

def foodsupply(listA):
    return modelF.predict([listA])

from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt
import numpy as np

fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

predictions = modelF.predict(X)
c = out
cA = []
for i in range(0, len(x)):
    cA.append(foodsupply(X[i])[0])

print(cA)
img = ax.scatter(x, y, z, c=predictions, cmap=plt.hot())
fig.colorbar(img)
plt.show()

## Pesticides
```

```
'''
@proof: pesticides
@type: polynomial 7
Proof or relationship and values
'''

y_col = 'Use␣of␣Pesticides␣per␣area␣of␣cropland␣(kg/ha)'
x_col = 'Cropland␣(Eco.␣Footprint,␣gha)'
k = df[y_col]
print(len(k))
fin = []
sec = []
count = 0
for val in k:
    if(int(val)!=0 and df[x_col][count]!=0):
        sec.append(val)
        fin.append(df[x_col][count])
    count+=1


%matplotlib inline

import scipy.optimize as opt
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from sklearn.linear_model import LogisticRegression

def logFunc(x, a, b):
    return a + b*np.log(x)

#p1, pcov = curve_fit(func, sec, fin)
popt, poc = scipy.optimize.curve_fit(logFunc, sec, fin)
print(popt)

plt.scatter(sec, fin)

plt.xlabel(x_col)
```

```
plt.ylabel(y_col)
plt.title(y_col)


z = np.polyfit(sec, fin, 2)
ppc = np.poly1d(z)

plt.scatter(sec, fin)


'''
@name: pesticides

Industrial output exihibits a linear relationship but does not have any
feeder variables and is thus a function of population.
'''

def pesticides(food):
    return ppc(food)

print(popt)
plt.plot(sec, [ pesticides(val) for val in sec])
plt.show()

## Birth Rate

'''
@proof: birth
@type: polynomial 7
Proof or relationship and values
'''

y_col = 'Year'
x_col = 'Birth␣Rate,␣Crude␣(annually␣per␣1000␣people)'
k = df[y_col]
print(len(k))
fin = []
sec = []
count = 0
```

```
for val in k:
    if(int(val)!=0 and df[x_col][count]!=0):
        sec.append(val)
        fin.append(df[x_col][count])
    count+=1


%matplotlib inline

import scipy.optimize as opt
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from sklearn.linear_model import LogisticRegression

def func(x, a, b):
    return a * np.exp(-b * x) + 7

sec = [val-1960 for val in sec]

#p1, pcov = curve_fit(func, sec, fin)
popWt, poc = scipy.optimize.curve_fit(func, sec, fin)
print(popWt)

def next_func(x):
    return popWt[0]* np.exp(-popWt[1] * x) + 7

plt.scatter(sec, fin)

plt.xlabel(x_col)
plt.ylabel(y_col)
plt.title(y_col)

plt.scatter(sec, fin)

#p1, pcov = curve_fit(func, sec, fin)
z = np.polyfit(sec, fin, 7)
paca = np.poly1d(z)
```

```
'''
@name: birth

Industrial output exihibits a linear relationship but does not have any
feeder variables and is thus a function of population.
'''

def birth(year):
    return next_func(year-1960)

plt.plot(sec, [ birth(val+1960) for val in sec])
plt.show()


## Death Rate

'''
@proof: death
@type: polynomial 7
Proof or relationship and values
'''

y_col = 'Year'
x_col = 'Death Rate, Crude (annually per 1000 people)'
k = df[y_col]
print(len(k))
fin = []
sec = []
count = 0
for val in k:
    if(int(val)!=0 and df[x_col][count]!=0):
        sec.append(val)
        fin.append(df[x_col][count])
    count+=1


%matplotlib inline
```

```
import scipy.optimize as opt
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from sklearn.linear_model import LogisticRegression

def funsc(x, a, b, c):
    return a * np.exp(-b * x) + c

sec = [val-1960 for val in sec]
#p1, pcov = curve_fit(func, sec, fin)
popYt, poc = scipy.optimize.curve_fit(funsc, sec, fin)
print(popYt)

def next_funcG(x):
    return popYt[0]* np.exp(-popYt[1] * x) + popYt[2]

plt.scatter(sec, fin)

plt.xlabel(x_col)
plt.ylabel(y_col)
plt.title(y_col)


plt.scatter(sec, fin)


'''
@name: death

Industrial output exihibits a linear relationship but does not have any
feeder variables and is thus a function of population.
'''

def death(year):
    return next_funcG(year-1960)

print(popt)
plt.plot(sec, [ death(val+1960) for val in sec])
```

```
plt.show()

## Pesticides
'''
@proof: pest
@type: polynomial 7
Proof or relationship and values
'''

y_col = 'Food␣supply␣(kcal/capita/day)'
x_col = 'Use␣of␣Pesticides␣per␣area␣of␣cropland␣(kg/ha)'
k = df[y_col]
print(len(k))
fin = []
sec = []
count = 0
for val in k:
    if(int(val)!=0 and df[x_col][count]!=0):
        sec.append(val)
        fin.append(df[x_col][count])
    count+=1


%matplotlib inline

import scipy.optimize as opt
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from sklearn.linear_model import LogisticRegression

def logFunc(x, a, b):
    return a + b*np.log(x)

#p1, pcov = curve_fit(func, sec, fin)
popt, poc = scipy.optimize.curve_fit(logFunc, sec, fin)
print(popt)

plt.scatter(sec, fin)
```

```
plt.xlabel(x_col)
plt.ylabel(y_col)
plt.title(y_col)

z = np.polyfit(sec, fin, 2)
ppCc = np.poly1d(z)

plt.scatter(sec, fin)


'''
@name: pest

Industrial output exihibits a linear relationship but does not have any
feeder variables and is thus a function of population.
'''

def pest(food):
    return ppCc(food)

print(popt)
plt.plot(sec, [ pest(val) for val in sec])
plt.show()



################################
"""
Final Model
"""
##############################

ALL_DATA = []


def black_box_world(asa):
    print(asa)
```

```
year_start = 1961

years = 500

yea = []
pop = []
br = []
dr = []
idn = []
co2A = []
co2B = []
co2C = []
nit = []
phos = []
buil = []
fores = []
co2f = []
graz = []
fish = []
live = []
crop =[]
food = []

def check(k):
    try:
        len(k)
        return k[0]
    except:
        return k

tot = []
mort = 0
for year in range(1961, (1961+years)):
    population = 0
    if(year<=2016):
        population = df['Population␣(1000␣persons)'][year-1961]
    else:
        population = prev_population + (birth(year)*24*365) - ((death(
            ↪ year)+mort)*24*365)
```

```
        yea.append(year)
        pop.append(population)
        br.append(int(birth(year)*24*365))
        dr.append(death(year)*24*365)

        idn.append(industrial_output(population))
        co2A.append(co2_industry(industrial_output(population)))
        co2B.append(co2_transpo(industrial_output(population)))
        co2C.append(co2_waste(population))
        nit.append(nitrogen(population))
        phos.append(phosphorus(population))
        buil.append(builtUp(industrial_output(population)))
        fores.append(builtUp(industrial_output(population)))
        co2f.append(fc(population))
        graz.append(grazing(population))
        fish.append(fishing(population))
        live.append(livestock(population))
        crop.append(cropland(population))
        eat = foodsupply(np.asarray([check(livestock(population)), check(
            ↪ fishing(population)), check(cropland(population))]))

        food.append(eat)

        tot_co2 = co2_industry(industrial_output(population))*asa[0] +
            ↪ co2_transpo(industrial_output(population))*asa[1] + co2_waste(
            ↪ population)*asa[2]

        mort = (tot_co2*0.00000001)-1 + (1/foodsupply(np.asarray([check(
            ↪ livestock(population)), check(fishing(population)), check(
            ↪ cropland(population))]))) * asa[3] + (1/industrial_output(
            ↪ population)) * asa[4]

        prev_population = population

    header = []

    header.append("Year")
    tot.append(yea)
```

```
header.append("Population")
tot.append(pop)

sides = []
sides.append(yea)
sides.append(pop)
ALL_DATA.append(sides)

header.append("Birth␣Rate␣per␣Year")
tot.append(br)
header.append("Death␣Rate␣per␣year")
tot.append(dr)
header.append("Industrial␣CO2")
tot.append(idn)
header.append("Transporation␣CO2")
tot.append(co2A)
header.append("Industry␣CO2")
tot.append(co2B)
header.append("Waste␣CO2")
tot.append(co2C)
header.append("Nitrogen")
tot.append(nit)
header.append("Phosphorous")
tot.append(phos)
header.append("Built␣Up␣Land")
tot.append(buil)
header.append("Forest␣Amount")
tot.append(fores)
header.append("Forest␣CO2")
tot.append(co2f)
header.append("Grazing")
tot.append(graz)
header.append("Fishing␣Grounds")
tot.append(fish)
header.append("Livestock␣Density")
tot.append(live)
header.append("Crop␣Land")
tot.append(crop)
header.append("Food␣Supply")
```

```
    tot.append(food)

    tot = np.asarray(tot).transpose()
    out = tot.tolist()
    out.insert(0, header)

    import matplotlib.pyplot as plt
    plt.xlabel("Year")
    plt.ylabel("Value")
    plt.title("Maybe?")

    plt.plot(yea, pop)
    #plt.show()



    import pandas as pd
    pd.DataFrame(np.asarray(out)).to_csv("tot_2.csv")



    return -max(pop)

from scipy.optimize import minimize
x0 = np.array([0.6, 0.2, 0.5, 0.4, 0.3])

#black_box_world([0.6, 0.2, 0.5, 0.4, 0.3])
res = minimize(black_box_world, x0, method='nelder-mead', options={'xtol': 1
    ↪ e-8, 'disp': True, 'maxiter':14})
```

Appendix K: Nelder-Mead Algorithm

```
def black_box_world(asa):
    print(asa)

    year_start = 1961

    years = 500
```

```
yea = []
pop = []
br = []
dr = []
idn = []
co2A = []
co2B = []
co2C = []
nit = []
phos = []
buil = []
fores = []
co2f = []
graz = []
fish = []
live = []
crop =[]
food = []

def check(k):
    try:
        len(k)
        return k[0]
    except:
        return k

tot = []
mort = 0
for year in range(1961, (1961+years)):
    population = 0
    if(year<=2016):
        population = df['Population␣(1000␣persons)'][year-1961]
    else:
        population = prev_population + (birth(year)*24*365) - ((death(
            ↪ year)+mort)*24*365)

    yea.append(year)
    pop.append(population)
```

```
        br.append(int(birth(year)*24*365))
        dr.append(death(year)*24*365)

        idn.append(industrial_output(population))
        co2A.append(co2_industry(industrial_output(population)))
        co2B.append(co2_transpo(industrial_output(population)))
        co2C.append(co2_waste(population))
        nit.append(nitrogen(population))
        phos.append(phosphorus(population))
        buil.append(builtUp(industrial_output(population)))
        fores.append(builtUp(industrial_output(population)))
        co2f.append(fc(population))
        graz.append(grazing(population))
        fish.append(fishing(population))
        live.append(livestock(population))
        crop.append(cropland(population))
        eat = foodsupply(np.asarray([check(livestock(population)), check(
            ↪ fishing(population)), check(cropland(population))]))

        food.append(eat)

        tot_co2 = co2_industry(industrial_output(population))*asa[0] +
            ↪ co2_transpo(industrial_output(population))*asa[1] + co2_waste(
            ↪ population)*asa[2]

        mort = (tot_co2*0.00000001)-1 + (1/foodsupply(np.asarray([check(
            ↪ livestock(population)), check(fishing(population)), check(
            ↪ cropland(population))]))) * asa[3] + (1/industrial_output(
            ↪ population)) * asa[4]

        prev_population = population

    header = []

    header.append("Year")
    tot.append(yea)
    header.append("Population")
    tot.append(pop)
    header.append("Birth␣Rate␣per␣Year")
```

```
tot.append(br)
header.append("Death␣Rate␣per␣year")
tot.append(dr)
header.append("Industrial␣CO2")
tot.append(idn)
header.append("Transporation␣CO2")
tot.append(co2A)
header.append("Industry␣CO2")
tot.append(co2B)
header.append("Waste␣CO2")
tot.append(co2C)
header.append("Nitrogen")
tot.append(nit)
header.append("Phosphorous")
tot.append(phos)
header.append("Built␣Up␣Land")
tot.append(buil)
header.append("Forest␣Amount")
tot.append(fores)
header.append("Forest␣CO2")
tot.append(co2f)
header.append("Grazing")
tot.append(graz)
header.append("Fishing␣Grounds")
tot.append(fish)
header.append("Livestock␣Density")
tot.append(live)
header.append("Crop␣Land")
tot.append(crop)
header.append("Food␣Supply")
tot.append(food)

tot = np.asarray(tot).transpose()
out = tot.tolist()
out.insert(0, header)

import matplotlib.pyplot as plt
plt.xlabel("Year")
plt.ylabel("Value")
```

```
    plt.title("Maybe?")

    plt.plot(yea, pop)
    #plt.show()

    import pandas as pd
    pd.DataFrame(np.asarray(out)).to_csv("tot_2.csv")



    return -max(pop)

from scipy.optimize import minimize
x0 = np.array([0.6, 0.2, 0.5, 0.4, 0.3])

#black_box_world([0.6, 0.2, 0.5, 0.4, 0.3])
res = minimize(black_box_world, x0, method='nelder-mead', options={'xtol': 1
    ↪ e-8, 'disp': True, 'maxiter':15})
```

Appendix L: Nelder-Mead Results

CO2 Industry	CO2 Transport	CO2 Waste	Food Supply	Industrial Output
0.6	0.2	0.5	0.4	0.3
0.63	0.2	0.5	0.4	0.3
0.6	0.21	0.5	0.4	0.3
0.6	0.2	0.525	0.4	0.3
0.6	0.2	0.5	0.42	0.3
0.6	0.2	0.5	0.4	0.315
0.57	0.204	0.51	0.408	0.306
0.54	0.206	0.515	0.412	0.309
0.576	0.1924	0.516	0.4128	0.3096
0.5664	0.19936	0.4874	0.41792	0.31344
0.55296	0.199104	0.50736	0.425088	0.297816
0.534144	0.1987456	0.510304	0.4071232	0.3119424
0.501216	0.1981184	0.515456	0.4006848	0.3179136
0.4946304	0.19799296	0.5164864	0.42739712	0.31910784
0.4419456	0.19698944	0.5247296	0.44109568	0.32866176
0.46500864	0.20742874	0.50397824	0.42591539	0.31713254
0.4340521	0.20369623	0.53920954	0.42399355	0.31476956
0.36787814	0.20586435	0.5651143	0.42703032	0.31543434
0.37345935	0.20665637	0.54235126	0.41760248	0.3374409
0.3198031	0.20002292	0.54565176	0.43293147	0.33763326
0.20970464	0.19703437	0.56097764	0.4433972	0.35194989
0.24198255	0.20747091	0.56340442	0.46133163	0.34233417
0.18897948	0.19817744	0.59865265	0.45026753	0.35319588
0.0509649	0.19355179	0.64598985	0.46244361	0.37122755

References

Bruce, P. (2012). One planet, how many people? a review of earth's carrying capacity. *Environmental Development*, *4*, 114–135.

Cohen, J. E. (1995). Population growth and earth's human carrying capacity. *Science*, *269*, 341–346.

Cole, H. (1973). *Models of doom: A critique of the limits to growth.* Vhps Rizzoli.

Daly, H. (1996). Toward some operational principles of sustainable development. *Ecological Economics*, 1–6.

Elkins, P., Gupta, J., & Boileau, P. (2019). *Global environmetal outlook* (6th ed.). Cambridge University Press.

FAO. (2019). *United nations food and agriculture organization database.* (Available at http://www.fao.org/faostat/en/data)

GFN. (2018). *National footprints account dataset 2018.* (Available at https://data.world/footprint)

Matthews, J., & Fink, K. (2004). *Numerical methods using matlab, 4th edition.* Prentice-Hall.

Meadows, D., Randers, J., & Meadows, D. (2005). *The limits to growth.* Earthscan.

OECD. (2019). *Industrial production (indicator).* (Available at https://data.oecd.org/industry/industrial-production.htm)

Salkind, N. (2010). Spearman rank order correlation. *Encyclopedia of Research Design*.

Steffen, W., et al. (2015). Planetary boundaries: Guiding human development on a changing planet. *Science*, *347*.

UN. (2017). *World population prospects 2017.*

Vaclav, S. (2005). *Energy at the crossroads; global perspectives and uncertainties.* MIT Press.

WB. (2017). *World development indicators 2017.* (Available at https://databank.worldbank.org/data/source/world-development-indicators)