Visualizing global nuclear power output

Building a flexible and efficient database application with plotting functionality using Python

Timothy Holt
MSc. Student, dept of Finance
University of Lausanne
Lausanne, Switzerland
timothy.holt@unil.ch

Tommaso Massazza
MSc. Student, dept of Finance
University of Lausanne
Lausanne, Switzerland
tommaso.massazza@unil.ch

Xavier Monition
MSc. Student, dept of Finance
University of Lausanne
Lausanne, Switzerland
xavier.monition@unil.ch

***Abstract*—This paper discusses the emissions of carbon dioxide that have been prevented from being released into the atmosphere through the use of nuclear fission reactors as a heat source in the generation of electricity. In order to investigate this subject, data from the Power Reactor Information System (PRIS) provided by the International Atomic Energy Agency, was used along with data provided by the US Energy Information Agency on the carbon intensity of coal fired electricity generation. An application was written in Python that reads raw data downloaded from the PRIS and generates a database that can be easily queried by the user to generate plots and custom data sets regarding a number of interesting statistics about nuclear power generation across the world since the first plant became operational in 1954.**

When it is run by the user, the Python application first reads raw excel data files from the PRIS and generates a database within the program. The application database is easier to extract useful information from, compared to the raw data provided by the PRIS. Once the database is initialized, the application asks the user for a query and the user can proceed to select which year(s), plant or country, and statistic they would like to extract information about. Once the user has entered all parameters the application produces a plot of the requested information and allows the user to export the data. The computationally efficient architecture of the database, allows for the processing of user queries of any size in fractions of a second on laptop computer hardware.

In this paper, after a brief discussion of nuclear power generation technology and how it compares to coal, the architecture of the Python application is discussed in detail. Next, some examples of plots generated by the program are presented and discussed. Finally, the approximate answer to the research question is discussed.

Using the PRIS data, the authors estimate that as of YE 2017, nuclear power reactors have prevented 86.7 metric gigatonnes of carbon dioxide from being emitted to the atmosphere assuming that all nuclear power generation was replaced by coal power generation. Total global carbon dioxide emissions in 2017 are estimated by the Global Carbon Project to be 34.7 metric gigatonnes. Thus, nuclear power generation has prevented approximately 2.5 years of global carbon dioxide emissions at 2017 levels.

***Index Terms*—nuclear, electricity, carbon emissions, data visualization, python**

I. Introduction

The ability to harness the power of nuclear fission to generate electricity is a technology that was developed by the Soviet and US governments in the 1950s. In the core of a nuclear power reactor, the energy released from the fission of uranium nuclei is used to heat water. This heated water is then passed through a heat exchanger where the heat is used to generate steam that spins a turbine that powers an electricity producing generator. These plants operate on much the same principal as coal powered electricity plants, with the prime mover being a steam powered turbine. The main difference is that they use a different heat source. The nuclear heat source has many of the same advantages as coal, being that the fuel source is relatively cheap, the production is stable and reliable, and the plants can be built to be very large. The largest plants have capacities from 4-8 gigawatts(GW).[1] For reference, 8GW is enough electricity to power the entire country of Switzerland.[2]

The major advantage that nuclear has over coal is that generating the heat releases zero carbon dioxide or other chemical pollutants into the atmosphere. The major disadvantage of nuclear fission as a heat source is that it produces hazardous radioactive waste that needs to be properly handled and disposed of. The other disadvantage is that nuclear reactors cause a severe safety concern, should some force majeure cause the reactor to melt down. Incidents involving nuclear reactors are classified by the International Atomic Energy Agency (IAEA) according to the International Nuclear and Radiological Event Scale (INES). This scale ranges from 0 to 7 with 7 being a major accident that results in a major release of radioactive material with widespread health and environmental effects. [1]

Since the technology started to be commercialized in 1960, there have only been two major(level 7) accidents that resulted in a major release of radioactive material. One in Chernobyl, Ukraine in 1986 and one in Fukushima, Japan in 2011. There have not been any significant(level 6) events since 1960. [1] Given the number of plants that operate across the world day in and day out, a record of 2 major incidents constitutes an

Thanks to the International Atomic Energy Agency for providing the data used in this research.

[1]The worlds largest nuclear power plant by capacity is Japan's Kashiwazaki Kariwa plant with 7 reactors with a combined capacity of 8GW. Plants as large as 6GW can be found in Canada, China, and South Korea.

[2]An 8GW plant running at a load factor of 90 will produce 63TWh of electricity in a year, while total Swiss consumption in the year 2018 was 58TWh.

impressive safety record for the technology. Despite this low accident rate, the grave consequences of a major accident mean that nuclear power plants today are facing major political opposition across the world. This opposition is making it difficult to build new plants to replace the world's aging stock of nuclear power reactors. Given how well these plants substitute the carbon intensive coal plants, this political opposition is a difficult hurdle that stands in the way of mankind capping and meaningfully reducing carbon emissions.

II. Research Question

The uncertain future of nuclear power plants in much of the world leads to an interesting research question about how the world might be different if mankind had not ever built any nuclear power reactors. More precisely, *if all of the electricity that has been generated using nuclear fission as a heat source was produced using coal as a heat source, how much extra CO_2 would have been released into the atmosphere?*

In order to answer this question, the International Atomic Energy Agency's (IAEA) Power Reactor Information System (PRIS) was used. The PRIS, developed and maintained by the IAEA for over four decades, is a comprehensive database focusing on nuclear power plants worldwide. PRIS contains information on power reactors in operation, under construction, or those being decommissioned. All information and data items are collected by the IAEA via data providers nominated by Member States. [2]

The PRIS is a private access database. In order to gain access to the PRIS, a subscription request was sent to the Swiss Federal Department of Foreign Affairs (EDA). After reviewing and approving the request, the EDA then forwarded this access request onto the Permanent Mission of Switzerland to the OSCE, the UN, and the International Organizations in Vienna who requested access to the IAEA on behalf of the authors. The approval process took approximately two weeks, after which time access was granted to the authors for research purposes.

III. Methodology

A. Overview

In order to answer the research question, a database containing information on nuclear power reactors, historical load factors, carbon emissions and world population was created using python. A number of querying functions were then written in order to pull information from this database in an efficient way, and display that information in graphical form. The following statistics are available to be retrieved from the database and plotted:

1) Actual Power Output
2) Available Capacity
3) Number of Reactors
4) Load Factors
5) Number of Plants
6) CO_2 Emissions

In order to be plotted, the statistics will automatically be aggregated on either the plant level, country level, or global level depending on what the user has requested. An example of a plot showing a time series of the total CO_2 emissions of all countries who operated nuclear power reactors compared with the estimated emissions should that nuclear power generation be replaced entirely with coal generation is shown in “Fig. 1”

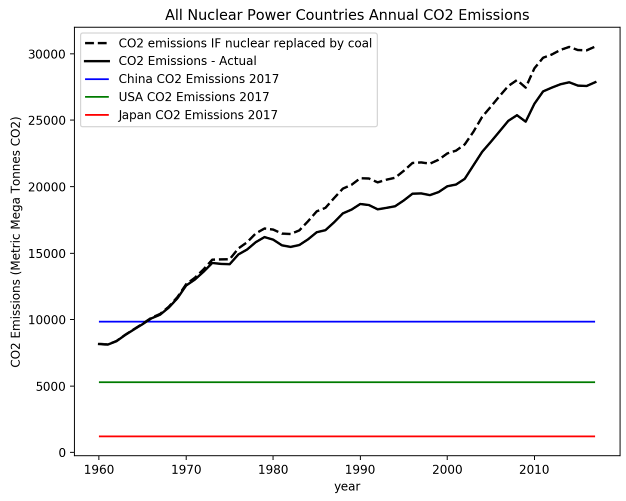


Fig. 1. Example of time series plot generated by program

Once the user sees the plot of the data that they requested, they have the option to export the data in a csv file that will be easy to work with for further analysis.

Additional functionality is available that generates a scatter plot of country nuclear power output vs population for a given year. Running the function that creates the scatter plot in a loop over all available years enabled the creation of an animated GIF that shows the progression of nuclear output on the national level for all countries who operate nuclear power reactors. “Fig. 2” shows one frame of this GIF file for 2017.

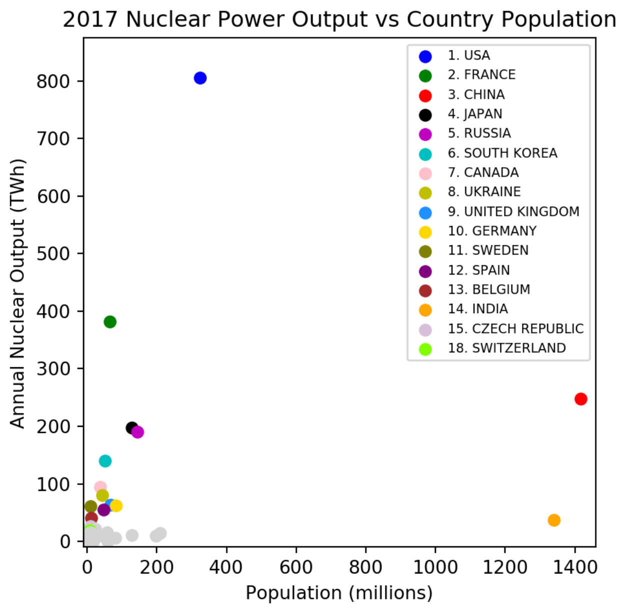


Fig. 2. Example of scatter plot generated by program

There is also a program that will produce a similar scatter plot, except with a log-log scale that better illustrates the ranking among the countries.

B. *How it works*

The first step to creating the database is to import and clean the data so that it is in a usable format. How this is done will be discussed in more detail in the *Input data and organization* section, but the basic process is as follows:

1) read data from downloaded .xlsx files
2) remove unwanted features such as merged cells
3) remove unwanted data
4) treat missing data as prescribed
5) export files to .csv format for later use
6) stitch sheets containing different years of data together

Once the cleaned and stitched .csv datasheets have been created, the data can be reliably extracted and stored in a more convenient format using a custom object class in Python called Plantyear. The Plantyear object is instantiated for each plant in each year of operations. This makes approximately 8,300 instances of this object, each instance having 9 unique attributes.

In order to create these instances of Plantyear, the relevant data is read from the .csv files into Pandas DataFrames. The data is extracted one ”cell” at a time from the DataFrames in order to collect all of the relevant information in one place. This info is then passed into a new instance of the Plantyear class.

Instances that share the same year are then stored in a list. Each list is then stored in a dictionary that is indexed by year. All of this is accomplished with a somewhat complex algorithm involving loops and search functions. The algorithm is not coded in the most intuitive way, but it is optimized for speed. Thanks to this speed optimization, the create dictionary algorithm takes only about 1.5 seconds (on a laptop) to create and efficiently store ~8,300 instances of Plantyear, each with 9 unique attributes, making ~75,000 unique data points. “Fig. 3” provides a visual representation of the final data structure with only three plants and three years.

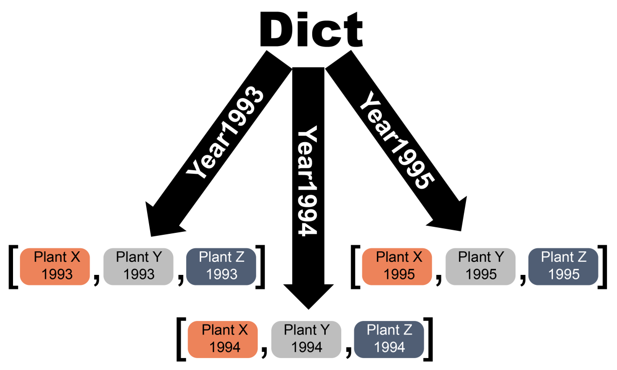
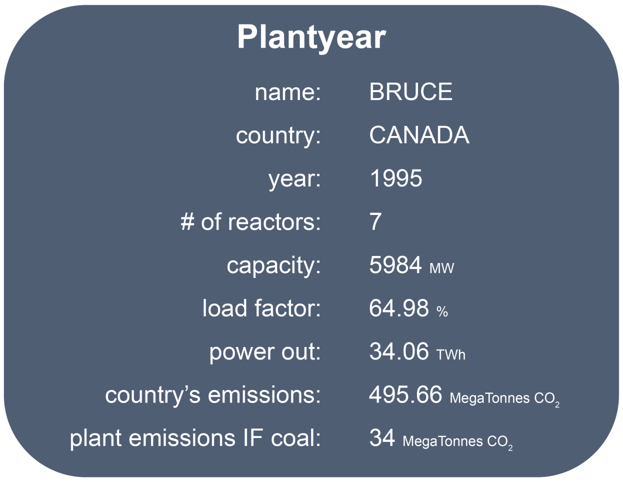


Fig. 3. Database Structure. Instances of object Plantyear are represented by colorful boxes

Once this structure has been built, the database is initialized and is ready to accept queries. In order to query the database, functions, found in the query_database_functions.py program, were written that take as inputs a range of years, the specific plant or country, and the requested statistic. The functions then proceed by looping through all of the plants inside a loop over the requested years and retrieving the desired statistic using a getter function in each cycle. The retrieved statistics are then put in a nested list that is transformed into a numpy matrix which is then returned by the function. Given the efficient architecture of the database, a query of any size takes only a small fraction of a second to complete on laptop hardware. This speed is observable given the time between the user making a query and the graph being displayed. None of the computation for these queries is done before the user submits the final parameter of the query, and thus the seemingly immediate appearance of the graph demonstrates the speed of execution.

C. *Implementation of the Plantyear Object*

The Plantyear object is the class that was created to represent one year of operations for one power plant. “Fig. 4” is a visual representation of an instance of the object Plantyear.



Plantyear

name:	BRUCE
country:	CANADA
year:	1995
# of reactors:	7
capacity:	5984 MW
load factor:	64.98 %
power out:	34.06 TWh
country’s emissions:	495.66 MegaTonnes CO_2
plant emissions IF coal:	34 MegaTonnes CO_2

Fig. 4. Visualization of instance of the Plantyear for BRUCE in 1995

The class has setter and getter methods for each attribute that are used to fetch or set the attributes of each instance of the object. Using the class in this way allows for packaging all of the desired attributes of a given plant in a given year in a single object that can be efficiently stored in a list. In addition to the getter and setter methods, the Plantyear class also has functions that automatically set power output and carbon emission equivalents given the capacity and load factor data. The Plantyear class also has a string method that allows for easy display of the attributes of an instance of the class using the built in print() function in Python.

D. *Calculation of plant emissions IF coal*

An important conversion factor (CF) for this research is the equivalency between Tera-Watt hours of electricity (TWh_e) and metric Mega-Tonnes of carbon dioxide (MT_{CO_2}). In order to calculate this figure, data provided by the US Energy

Information Administration (EIA) was analyzed. Given the available data, two different approaches were taken to arrive at this conversion factor:

1) Calculation using heat constants provided by the EIA: Details of these calculations are provided below:

$$CF = \frac{BTU}{ST_{coal}} * \frac{KWh_e}{BTU} * \frac{TWh_e}{KWh_e} * \frac{ST_{coal}}{MT_{CO_2}}$$

Substituting in figures provided by the EIA [3] we arrive at the following calculation:

$$CF = \frac{2.05 \times 10^7}{1} * \frac{1}{1.02 \times 10^4} * \frac{1}{1 \times 10^9} * \frac{9.95 \times 10^8}{1.93 \times 10^3}$$

$$CF = 1.0 \, \frac{TWh_e}{MT_{CO_2}}$$

2) Calculation using aggregated emissions and electricity generation data provided by the EIA: In order to calculate this figure, the data of electricity generated using coal in the USA for each year from 1990 to 2017 was divided by the CO_2 emissions from coal fired power plants in the USA for each year from 1990 to 2017. [4] The mean of all years from 1990 to 2017 was then taken as the CF. Details of calculation below:

$$CF_{annual} = \frac{USA \; electricity \; output \; from \; coal}{USA \; CO_2 \; emissions \; from \; coal \; plants}$$

$$CF_{2017} = \frac{1206 \;\; TWh_e}{1226 \;\; MT_{CO_2}}$$

$$CF_{2017} = 0.98 \, \frac{TWh_e}{MT_{CO_2}}$$

$$CF = \text{mean}(CF_{1990}, \;\; \dots \;\; , \; CF_{2017})$$

$$CF = 1.0 \, \frac{TWh_e}{MT_{CO_2}}$$

It can be seen that the same rounded figure was reached with the two different calculation methods. It is an interesting coincidence that the $CF = 1$. With this conversion factor, the CO_2 equivalency IF coal is easily calculated by multiplying the power out by the conversion factor. This parameter is thus calculated and set in the initialization function of the class.

E. Input data organization and structure

The data that can be downloaded from the PRIS is in the form of MS Excel sheets. The sheets are automatically generated by the PRIS according to the user selected parameters. These automatically generated sheets are in a format that is designed to aide readability by the user and ease of use in MS Excel. What this means is that the sheets contain features such as merged cells with labels spanning several columns, rows with collapsible headers and other special formatting. While these features may make navigation in MS Excel a little bit easier, it makes doing analysis on the data more difficult since it is not obvious how such sheets are read into analytics packages such as a Pandas DataFrame. Thus, one of the first tasks was to clean the data and put it into a form that is more desirable for analytics purposes. There are four programs that do this task:

- clean_reactor_sheets.py
- clean_population_data.py
- clean_carbon_sheet.py
- generate_load_factor_matrix.py

Each of these programs takes as an input a datasheet in MS excel format and outputs a cleaned version as a csv file. Clean csv files can be found in the cleaned data folder.

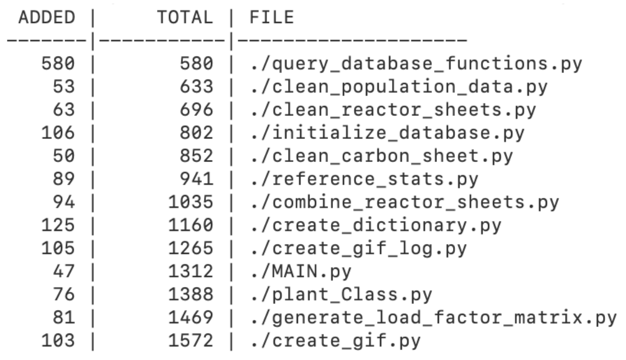
The clean_population_data.py program does not use input data from the PRIS database, but instead downloaded from the United Nations DESA World Population database. This sheet shares some of the problems of PRIS data, and also contains data from many countries that have never had operational nuclear power reactors. Thus, the data from these countries must be removed from the sheets before processing. In addition, the sheet uses different naming conventions for some countries than those used in the PRIS data.

The clean_carbon_sheet.py uses data from yet another source: The Global Carbon Atlas. This sheet is plain and simple, but has data from many countries that have never had operational nuclear power reactors. Again, the data from these countries must be removed. In addition, the sheet also uses different naming conventions for some countries than those used in the PRIS data.

An additional challenge with the data from the PRIS is that there are restrictions on the size of spreadsheets that can be generated. This provided the biggest challenge with main reactor information sheets. The time series data had to be downloaded in four sheets with each sheet containing ~15 years of information. Because each sheet here contains a different set of reactors, due to reactors being constructed or decommissioned, it was necessary to stitch the sheets together with a computationally intense algorithm. This stitching is done by the combine_reactor_sheets.py program. This program runs after the sheets have been cleaned and is the most time consuming step in the program, taking ~11 seconds to run on a laptop.

F. Organization of code

In order to make the program easier to develop and update, the code has been broken into sub-programs that each complete a discrete task. The program is broadly executed in 3 parts. The whole program is run from the MAIN.py program which contains very little code, but is the highest level aggregation of the sub-programs. To better understand how the code is distributed across the various programs, “Fig. 5” shows how many lines of code are in each of the constituent sub-programs. It can be seen that the single largest file is the query_database_functions.py program. This is not surprising given that this is the file where all of the functions that are used to interact with the user and process their requests are stored. This amounts to 10 functions of varying complexity.



```
 ADDED |     TOTAL | FILE
-------|-----------|--------------------
   580 |       580 | ./query_database_functions.py
    53 |       633 | ./clean_population_data.py
    63 |       696 | ./clean_reactor_sheets.py
   106 |       802 | ./initialize_database.py
    50 |       852 | ./clean_carbon_sheet.py
    89 |       941 | ./reference_stats.py
    94 |      1035 | ./combine_reactor_sheets.py
   125 |      1160 | ./create_dictionary.py
   105 |      1265 | ./create_gif_log.py
    47 |      1312 | ./MAIN.py
    76 |      1388 | ./plant_Class.py
    81 |      1469 | ./generate_load_factor_matrix.py
   103 |      1572 | ./create_gif.py
```

Fig. 5. Number of lines of code in each constituent program

G. *Order of program execution*

When the MAIN.py program is executed, it first runs the initialize_database.py program in order to prepare the database for queries. The initialize_database.py program then calls the query_database_functions.py program which imports all of the functions that are used to execute user queries of the database. Once the two sub programs have finished running, the MAIN.py will commence executing user queries. “Fig. 6” shows a graphical representation of this process.

H. *Execution of user queries*

Once MAIN.py has finished the steps of preparing the database for user queries, it then starts interacting with the user by asking them what data they would like to pull from the database. All of this functionality is coded in functions that are stored in the query_database_functions.py program. The reason for this extensive use of functions is so that the functionality is repeatable for as many queries that the user wants to submit. This also improves the organization of the code, by making things more modular. Thus, the only code that appears in the MAIN.py program are the loops which continue executing queries as until the user says to stop. A typical execution of a user query follows these steps:

1) ask_user_params() - function to get from user their desired start year, end year, country, or plant
2) ask_user_statistic() - function to get from the user which statistic they would like to see
3) process_request() - function that takes user inputs and processes them

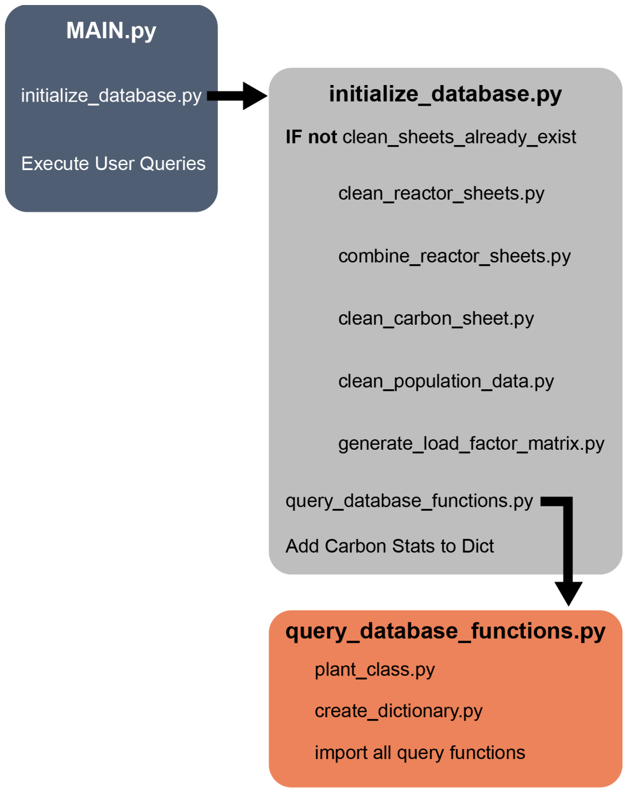


Fig. 6. Program order of execution

   a) gen_matrix() - function to pull requested data matrix from database
   b) elim_zero_rows() - function that eliminates rows containing only zeros from data matrix
   c) plot_timeseries() - function that plots the time series data including automatically determined relevant reference statistics
   d) export data matrix as .csv if user requested
4) repeat if prompted

I. *Single frame scatter plot generation*

The scatter plots of nuclear power output vs country population are generated through a different algorithm than the time series plots. The algorithm to generate the scatter plots is more compact than that used to generate the time series plots, mostly due to reduced number of options that the user has to customize the plot. The program only requires that the user input a year and the scatter plot will be generated for that year. The program executes the user query following these steps:

1) ask_user_year() - function to get from the user the year to plot
2) plot_scatter() - main scatter plotting function
   a) gen_matrix() - function to pull requested data matrix from database. In this case, the matrix will be a 1-dimensional array
   b) plot the data

On the plot the countries that are in the bottom half of countries for nuclear power output will appear as grey dots in order to make the legend more readable.

J. Multiple frame scatter plot generation

In order to create the GIF files that show country nuclear power output vs population over time, the single frame scatter plot generation algorithm was essentially put inside a loop over the available years. There are some complications that arise when taking this approach. Firstly the scale constantly changes over time in order to best fit the plotted data to the frame of the plot. This is a desirable feature for a single year scatter plot, but not for a time series that will be made into a GIF since it will confuse the viewer. Thus, the plot scale was fixed. In addition to this, the single frame algorithm assigned colors to countries based on their rank of nuclear power output. While this works well for single year data, it is not desirable for a GIF, since the user will lose track of countries if they are changing color every time they change rank. Thus a large dictionary was created that assigned a unique color to each country. In addition to this, the function was modified so that Switzerland will always appear as a colorful dot, regardless of its rank in nuclear power output.

Once these modifications were made, the program was looped over every available year using both an actual scale, and a log scale in order to generate the frames of the GIF. A bash script was then executed using imagemagick, which turned the frames into an animated GIF. The GIFs were created in 3 speeds in order to allow the user to better see the trends evolving over time.

IV. Discussion of Implementation of Algorithms

A. Frequently used functions

The program relies heavily on functions written by the authors that make the database queries easily repeatable. In this section the detailed functionality of the most commonly used functions will be discussed.

1) ***gen_matrix():*** This function takes the user requested inputs and queries the database to collect the data then finally outputs a data matrix of the chosen parameters, and 2 lists of row labels: One containing countries names, and the other containing plant names.

Inputs:

- statistic - statistic on the nuclear power plants that the user has requested from the available list discussed in section III. A.
- start_year=1954 - the year for which the time series data should start. Must be in the interval [1954, 2018].
- end_year=2019 - the year for which the time series data should end. Must be in the interval [1954, 2018], AND be greater than start_year
- country=None - if the user requests data for a specific country the name of that country goes here. An argument of None will result in data from all countries that had operational plants during the time interval being returned.
- plant=None - if the user requests data for a specific plant the name of that plant goes here. An argument of None will result in data from all plants that were operational during the time interval being returned.
- database=Dict - this argument passes in the database that the statistics will drawn from. Default argument of Dict is always used.

Outputs:

- matrix - $n * m$ matrix of raw data requested. Where n is the number of plants, and m is the number of years. Matrix contains only raw data, has no attached row or column labels.
- country_labels - list of length n containing the country names. Countries appear in the same order as those in the data matrix.
- plant_labels - list of length n containing the plant names. Plants appear in the same order as those in the data matrix.

This function starts by initializing 3 empty lists. One for the matrix, one for the country labels, and one for the plant labels.

The function then starts a loop indexed by i over all of the years that were requested. For each i it initializes an empty list, x.

It then starts a loop indexed by j over all of the plants that exist inside the database for that given year. Inside this loop the function then gets the appropriate statistic from the current Plantyear object by using a getter function, and then appends that statistic onto list x. During the cycle $i == 0$ the country name and plant name are drawn from the plant objects using getter functions and then appended onto the lists of country and plant labels.

Once the j loop has finished it appends the list x onto the list $matrix$, and then cycles over a new j loop, until all years have been exhausted.

The data structure that results is a nested list:

$$matrix = [\,[1,2,3],\ [1,2,3],\ [1,2,3]\,]$$

This nested list is then sent through the numpy.vstack() function that transforms it from a nested python list object to a numpy ndarray object that is a $n * m$ matrix of data. The numpy.transpose() function is then used to put the data into the orientation that is consistent with the customary orientation in which plants are rows and years are columns. The resulting data structure is pictured below:

$$matrix = \begin{bmatrix} 1 & 1 & 1 \\ 2 & 2 & 2 \\ 3 & 3 & 3 \end{bmatrix}$$

The function then initializes an empty list to_delete and if a specific country or a plant has been requested it finds the indices of all rows that do not contain that specific country or plant, and generates a list of row indices for rows that contain irrelevant data. Thus, we have a list of rows to be deleted.

If the list is non-empty the function then eliminates the rows from the matrix using the numpy.delete() function, and then loops over the list of rows to delete and pops these elements out of the lists of country and plant labels. Elements are popped out starting from the back of list so that the indexing of these lists does not cause undesired elements to be popped from the list as would happen if the elements were popped out from the front.

Once all undesired data has been removed from the matrix, the country label list, and the plant label list, all three of these objects are returned.

*2) **plot_timeseries():*** This function plots the data matrix that is input. This function must be flexible enough to handle a wide variety of data inputs. This flexibility makes it relatively complicated and long at 90 lines of code.

Inputs:

- matrix - the data matrix produced by the gen_matrix() function
- year_labels - a list of years over which the time series develops
- chart_label - input used to adapt plot title based on data series plotted
- stat - statistic on the nuclear power plants that the user has requested from the available list discussed in section III. A.
- sec_matrix=None - in case of a carbon plot, two lines are plotted, the function therefore needs a second data matrix for the CO_2 emissions IF nuclear replaced with coal.
- cumulative=False - whether user has requested a plot of a cumulative timeseries.
- country_labels - list of country names that matches the matrix
- statistic_dict=statistic_dict - the dictionary of titles for the axis and plot title for reference statistics. The purpose of these ref stats is to give the reader some idea of relative size of data which is in units that most users will not understand.
- ref_nums_dict=ref_nums_dict - the dictionary of reference statistics.
- ref_labels_dict=ref_labels_dict - the dictionary of names for plot legends for reference statistics.

Outputs: plot

The first thing this function does is generate a vector of y values that will be plotted with the x values of years from the year_labels that can be plotted. To achieve this it first initializes an empty list y_vector.

Generating a vector from a matrix requires us to somehow collapse the rows of the matrix into a single row. Depending on the type of plot that is requested the y_vector will be made by either averaging all of the rows (load factor data) or summing all of the rows (all other data)

To generate this y_vector, the function loops over index y which is the number of years to be plotted, taking a sum (mean) of all rows of the matrix for year y and appending the sum (mean) to the y_vector. In the case of the mean, the matrix may have zeros for plants that were not yet operational and hence should not be included in the computation of average load factor. In order to fix this issue, zero values in the matrix are replaced with NaN using the function numpy.where(). Then the mean is taken using the function numpy.nanmean() that will ignore NaN values when computing the mean.

Now we have a y_vector of values to be plotted. Next the function will accumulate the y_vector using the numpy.cumsum() function if the user requested to do so.

Next the function determines which reference stats to use. The purpose of the reference stats is to give the user an idea of the scale of the data they are looking at. We can see a plot of the nuclear power output of Country X, but most users will not know whether the numbers represented by the plot are large or small unless there is some frame of reference provided.

For the references to be valuable they must be relevant. That means it would not make sense to plot, as a reference, the power consumption of the USA on a plot of the nuclear power output of Switzerland. Such a reference stat would be far too large. Instead it would make more sense to plot the power consumption of Hawaii as a reference stat. Thus the next section of code automatically determines what are the most relevant reference statistics that are contained in the reference_stats.py file.

The function will plot the highest 3 reference statistics that are lower than the highest value in the y_vector. It does this by looping through the relevant sections of the ref_nums_dict dictionary and evaluating whether the stat is larger than max(y_vector).

Next the y_vector is plotted followed by the reference lines. The plot is then labeled, the legend generated, and the plot shown.

B. Packages used and effects on program performance

This project was written in Python 3.7.3 language using the following packages and libraries:

- Pandas (v0.24 required)
- Matplotlib v3.0.3
- Numpy v1.16
- Itertools v2.3
- threading
- os
- sys
- time

1) Pandas: Pandas was frequently used for its DataFrame functionality. When the program is executed, the first lines of code import the raw data from excel sheets using the Pandas read_excel() function. The resulting Pandas DataFrames are then processed to extract the data that we are interested in. Pandas is heavily relied upon in the initialize database steps that involve cleaning the data to a more usable form.

Pandas was found to be very versatile and able to handle a wide range of inputs and tasks, but these processes were found to be relatively slow, suggesting that the package is not optimal for computational efficiency. This was particularly noticeable when implementing algorithms that relied on manipulating DataFrames on the ”cell” level. For this reason, Pandas is not used once the data is ”clean” and the database is initialized.

The combine_reactor_sheets.py program executes a series of Boolean evaluations on DataFrame ”cells” using the DataFrame.loc[] function to assess whether there is redundant data in the two DataFrames, and then .loc indexing to write data to ”cells” as required to stitch DataFrames together. While this methodology was reliable and required only 15 lines of code, this process was found to be computationally taxing, requiring ~11 seconds to execute for a data set of $2 * 8300$ unique objects. Achieving a time for this process as low as 11 seconds was the result of substantial algorithmic optimization efforts.

Aside from algorithmic optimization, further speedup was attempted on the combine_reactor_sheets.py program with the use of a Just In Time (JIT) compiler. The JIT compiler used was that of the NUMBA package. No speedup was found with this JIT compiler, and it appeared that the use of the compiler required significant overhead, since the execution time was longer in every case where it was used. Upon review of the NUMBA documentation, a disclaimer was found that stated that the JIT was not optimized for use with Pandas, explaining the disappointing results.

2) Matplotlib: The plotting functionality of Matplotlib was used extensively to plot the data after being drawn from the database.

3) Numpy: Numpy is used extensively in processing data that is drawn from the database. Numpy is used primarily because of its optimal computational efficiency that allows for very fast execution of data extraction and processing from the database.

Numpy functions are scattered throughout the code, but are used the most in the gen_matrix() function. The data matrices that are fed into the plotting functions are stored as numpy ndarrays.

4) Itertools and threading: The itertools and threading libraries were primarily used to create the loading animation that plays while the computations are happening while initializing the database. Itertools.chain() function was additionally used to create the lists that will be displayed if the user asks to ”show countries” or ”show plants” when prompted.

V. Debugging

A continuous process of debugging was conducted at all stages of development of the application. Due to the amount of data to be handled by the application, before creating and implementing each section code, draft algorithms were created in order to assure that the algorithm produced the desired outcome.

During the spreadsheet import and cleaning process, each of the spreadsheets was re-exported and manually verified to contain identical information to the initial spreadsheet downloaded form the PRIS database. Then, in the process of creating the lists of plant attributes, numerous plants at each year were printed using a custom debugging function in order to verify that the order and the content of the list was as expected. Once satisfied with the outcome, the lists were used as input to create an instance of the Plantyear object. In order to debug the dictionary, another custom debugging function was written that printed all of the attributes of each instance of the class which were randomly verified against the original spreadsheets downloaded from the PRIS. Indexing the dictionary by year facilitated debugging when subsequent changes were made.

In the MAIN.py document, the interface with the user has been thoroughly tested and updated in order to avoid any undesirable outcome based on unexpected user inputs. For other smaller parts of the code, the same procedure of manually verifying that outcomes were identical to those expected was repeated frequently throughout the development process. Once the application was functional, when structural changes were made, all of the dependant parts of the code were rechecked using verification procedures as described above.

VI. Maintenance and Updates

A. Updating when new data sets are released

The PRIS data sets are updated annually with the latest data on load factors, plant construction, plant operation, and decommissioning. In order to update the code with the latest data, all that needs to be done is download the new data sheet from the PRIS, delete top rows in the data sheet that displays the PRIS logo, and then replace the file in the virgin_data directory with the updated file. There are a few locations in the code where the year is restricted to 2019 which will need to be changed to accommodate the new date range. This should be easy to do using a global search for ”2019”.

It is recommended that thorough verification be done on the application output data should such an update be done, in case of an anomaly introduced by something such as an updated format of the downloaded PRIS datasheet.

B. Code base maintenance

It is recommended that a verification of output data be conducted when there is a new release of Pandas that may effect some of the default settings on Pandas DataFrames. During the construction of the application, it was discovered that a change in the way that Pandas interprets column headers between two different versions caused an error when running the application.

If there is a problem with any output data generated by the application, it is probable that the error will be obvious, or else cause the application to throw an error message.

C. Functionality updates

Given the high degree of modularity of the code, and the thorough documentation and commenting, updating to add functionality should be reasonably straight forward to an experienced Python user. Some possible functionality improvements that could be implemented in the future include:

- larger library of reference statistics with some ”fun” statistics

- added support for reactors in construction and those being decommissioned
- forecasting support
- reactor age information support
- reactor aggregation based on different groupings such as continent, age, technology etc.
- support for economic impacts of nuclear
- support for other types of alternative energy generation such as natural gas, petroleum, hydro etc.

VII. Limitations of Estimates

Although the data in the PRIS is of high quality, there are some missing values present in the data set for load factors. These missing values were most common during the 1970s and 1980s and only exist for countries with low transparency such as the USSR, IRAN, etc. In addition to missing values for non-transparent countries, The load factors were not collected before 1970, so there is a time when there is no load factor data available at all. In order to handle this missing data, and attempt to most accurately model the actual output of the nuclear plant, an adjustable treatment was written in the code.

In the create_dictionary.py code, 3 possible treatments have been created. Treatment 0 is the most conservative, replacing each plant missing load factors with zeros values. Treatment 1, sets a more aggressive assumption but is still reasonably conservative by replacing the missing values of each plant with the respective plant's minimum load factor in the available years. Treatment 2, sets a more aggressive assumption replacing the plants missing load factors with the respective plant's average load factors over the available years. The findings below have been generated setting the treatment assumption to 2.

VIII. Results

This section will display the findings on power production, nuclear capacity, trends on global reactor stocks, load factors, power plants and carbon emissions. Results will be displayed at a Global level as well as for individual countries. For each of the available statistics, some of the most interesting plots are shown. If a specific country or a plant is the subject of the reader's interest, selecting the desired country or plant while running the MAIN.py will allow the reader to generate the same plots for the country of interest.

The application produces plots of both cumulative and non-cumulative data. The plots that are cumulative are shaded in the area under the black line.

A. Findings on power production

The global cumulative annual nuclear electricity output has rapidly increased since the first nuclear power plat was constructed in 1954. The overall cumulative nuclear electricity output is now equivalent to ~3.8 times the World electricity consumption in 2015.

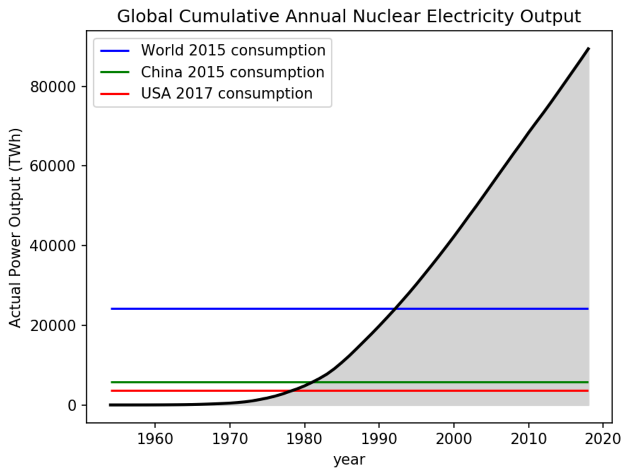


Fig. 7.

The country producing the most nuclear electricity output on an annual basis is the United States of America. Similar to other countries, the annual nuclear electricity output increased substantially form 1970 to 2000. After which time the nuclear electricity output growth starts stagnating. The USA 2018 nuclear electricity output, was higher than the German electricity consumption in 2018.

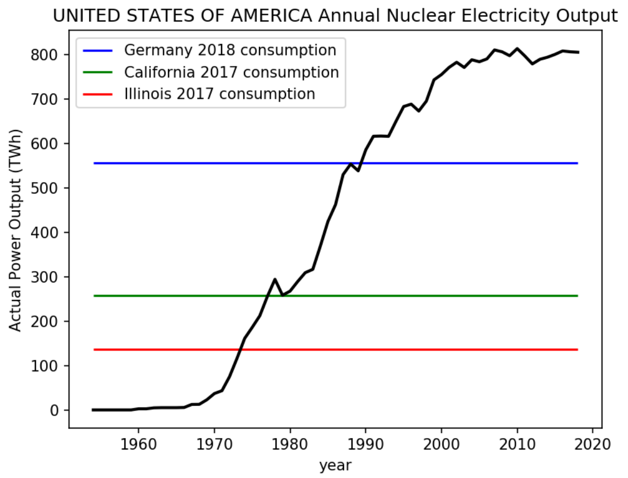


Fig. 8.

In 1968, Switzerland's first nuclear power plant (LUCENS) became operational. In 2018, Switzerland had 5 operational reactors that produced 2.6 times the 2017 Hawaiian electricity consumption. Moreover, in the same year, the electricity produced with the 5 reactors accounted for 43.14 percent of the Swiss electricity consumption.

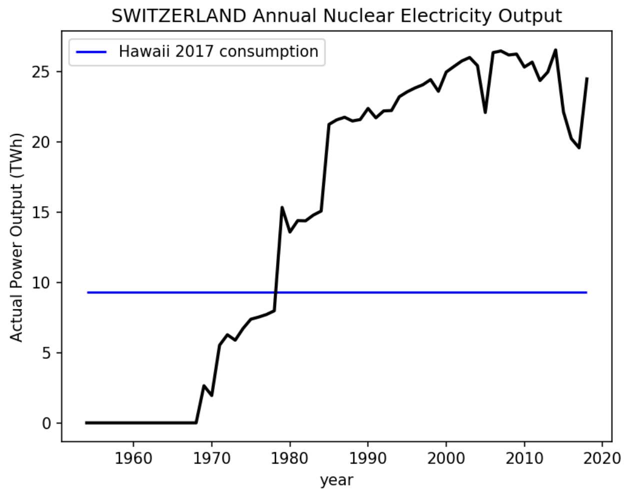


Fig. 9.

Concerning Japan, it is interesting to notice that since the Fukushima reactor meltdown in 2011 the nuclear electricity output has substantially decreased. Japan went form producing an annual electricity output that was similar to the California state electricity consumption down to that of the much smaller state of Illinois.

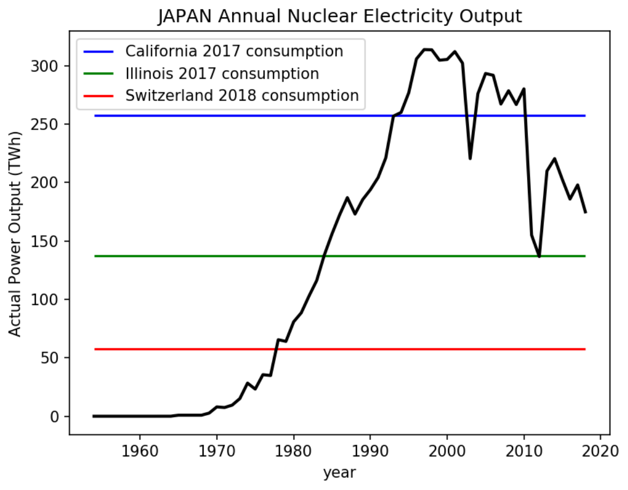


Fig. 10.

Chinese annual nuclear electricity output has grown exponentially in the last two decades. In 2018, the nuclear electricity output in China was 23 times higher than the output in 2000. According to the PRIS, in 2019 China is the country constructing (11) and planning (20) the most reactors in the world.

This is compared to the United States that are constructing 2 reactors and planning "only" 6 additional reactors. It is probable that in 2-5 years, China will overtake the United States of America in nuclear power output.

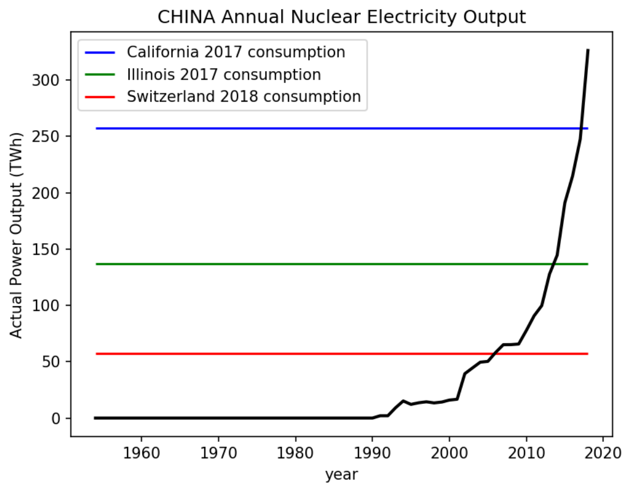


Fig. 11.

B. Findings on Nuclear Capacity

The global annual nuclear power capacity experienced substantial growth from 1970 to 1990. After 1990, the growth slowed down resulting in a total available capacity in 2018 of 397 GW.

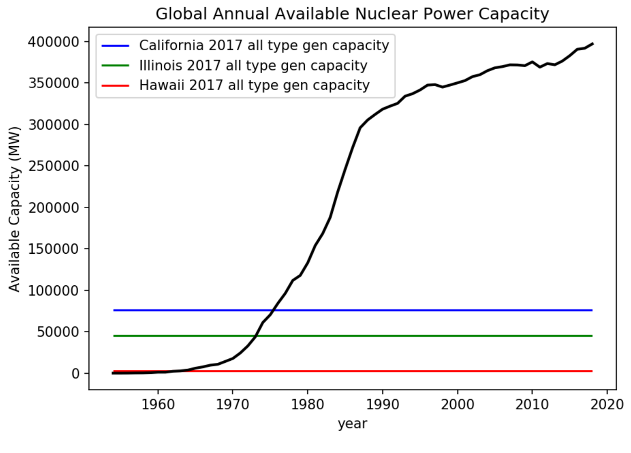


Fig. 12.

The nuclear power capacity in Switzerland increased to 3.3 GW in 2018. Each of the steps representing one or more reactors becoming operational.

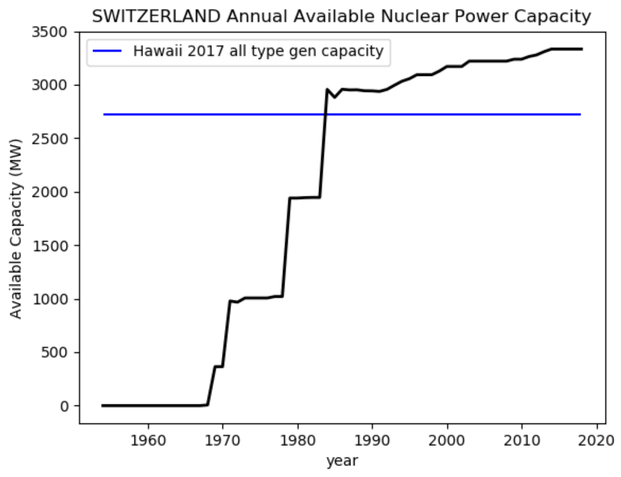


Fig. 13.

C. Findings on trends in global reactor stock

The number of global operational reactors increased exponentially from 1954 until 1986 when the ramifications of the Chernobyl disaster caused the cancellation of new construction plans globally. Since 1986, the number of new reactors constructed has completely flattened out to almost exactly the replacement rate.

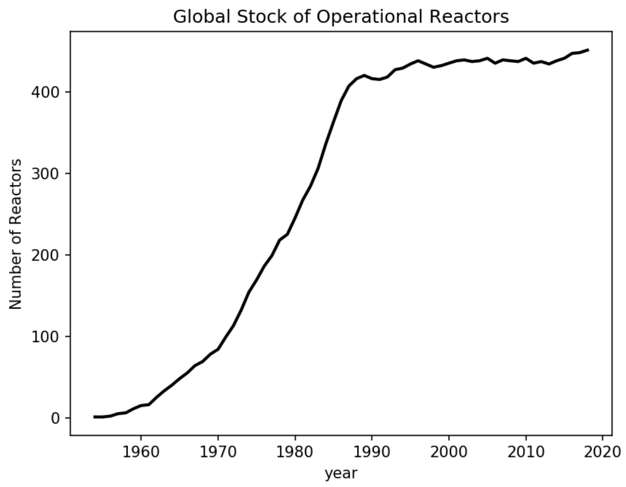


Fig. 14.

D. Findings on load factors

While comparing the average annual load factor at a global level, “Fig. 16”, to the level in the United States of America “Fig. 17”, it is noticeable that the average annual load factor of American reactors is higher that the global average annual load factor.

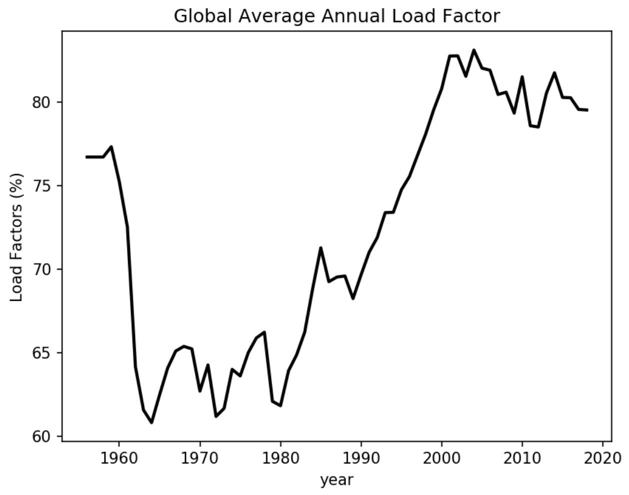


Fig. 15.

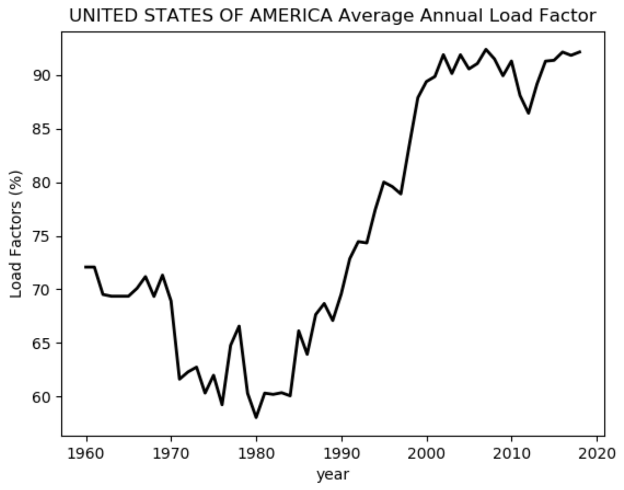


Fig. 16.

E. Findings on Plants

The number of nuclear power plants increased rapidly from 1954 to 1988. The decrease in number of nuclear power plants is correlated with the Chernobyl disaster. Since Chernobyl, countries finished constructing the plants in construction and continued to use the operational power plants but did not construct new plants. It is interesting to note that while the number of operational plants has decreased since 1986, the number of operational reactors has slightly increased. Indicating that the average size of plants is getting larger, with more multi-reactor plants being built.

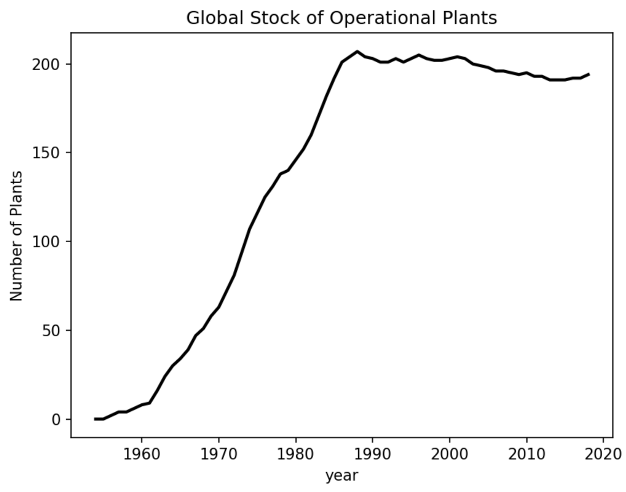


Fig. 17.

The graph below shows that the current number of nuclear power plants in Switzerland is 4. The total number of constructed Swiss plants since 1968 has been 5. However, the first Swiss nuclear power plant, LUCENS, operated only during one year due to a level 4 meltdown (accident with local consequences) in January 1969. The second Swiss power plant, BEZNAU became operational the same year.

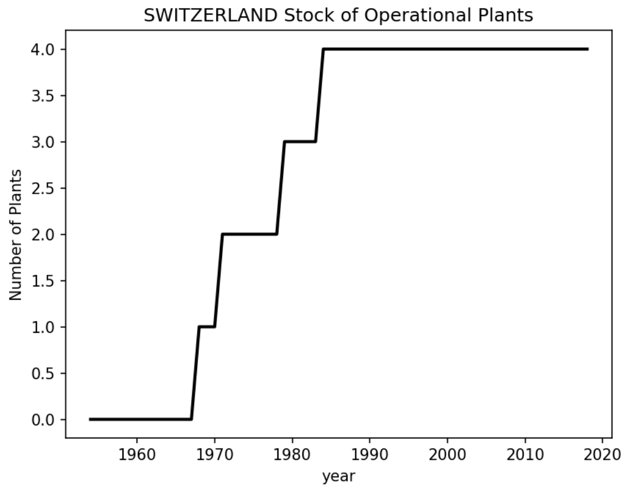


Fig. 18.

F. *Findings on carbon emissions*

The cumulative impact of nuclear power plants on emissions over time shown in “Fig. 19”. Shows that lines continue to diverge as time goes on, displaying that Nuclear has prevented substantial carbon emissions since its inception.

France is the country that produces the most nuclear power per capita in the world “Fig. 20”. Notice how large the prevented MT CO_2 has become. It is now almost as large as the combined annual CO_2 emissions of China and the United States.

In the plot of annual emissions from all countries with nuclear power reactors “Fig. 21” it is interesting to see how carbon emissions from these countries has mostly leveled off in the last decade. It is also interesting to see how much of an impact the global financial crisis and the ensuing recession had on carbon emissions.

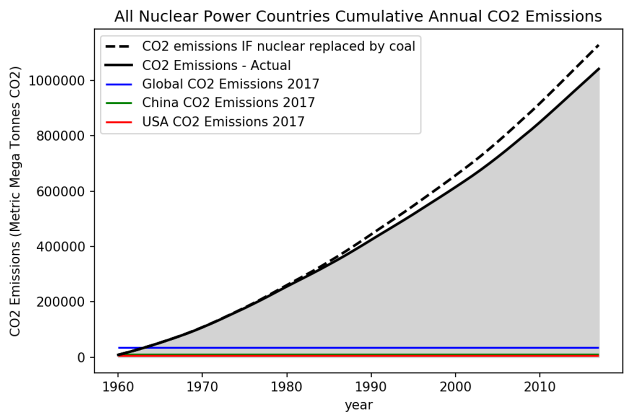


Fig. 19.

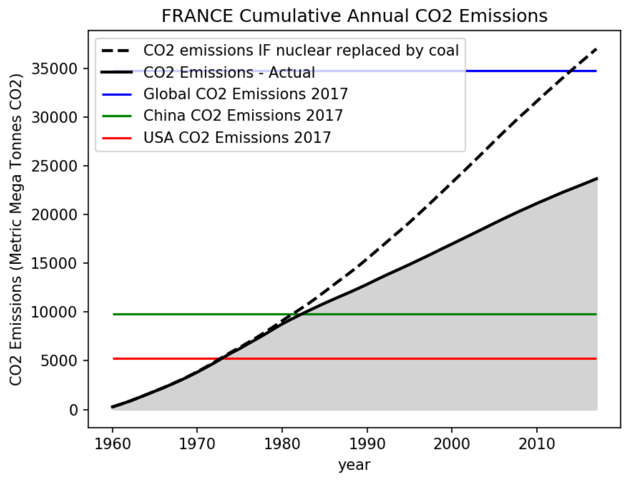


Fig. 20.

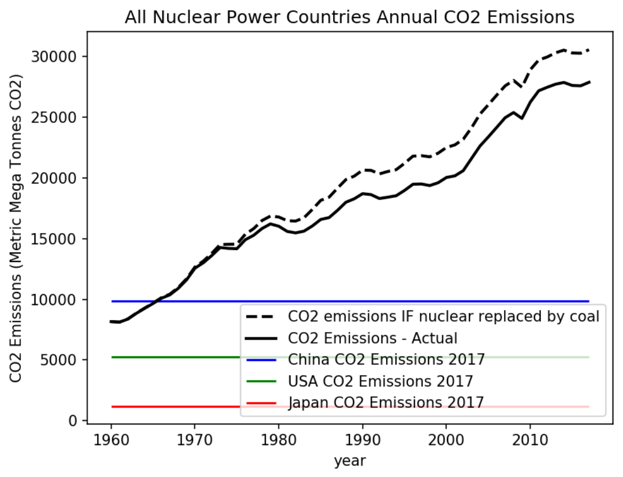


Fig. 21.

The United States produce the most nuclear power out of any country in the world, yet it is interesting to see that in comparison to the size of the US economy the amount of nuclear energy is not that large. The contrast to France or Switzerland that can be seen in the following figures is stark.

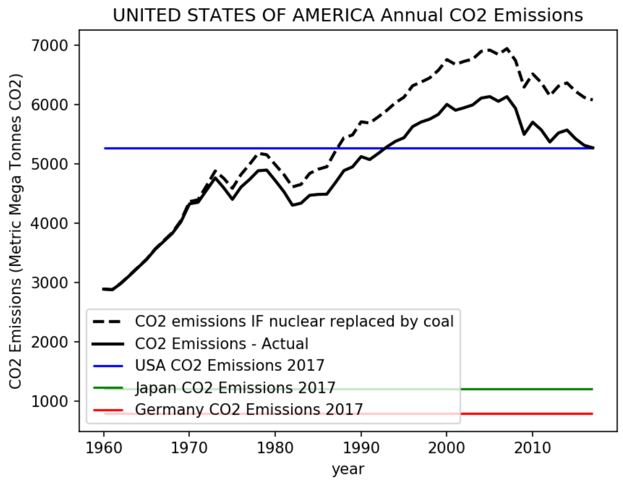


Fig. 22.

If France did not use nuclear power their carbon emissions would be very close to that of Germany, but with their vast amount of nuclear power generation they emit less than half as much as Germany.

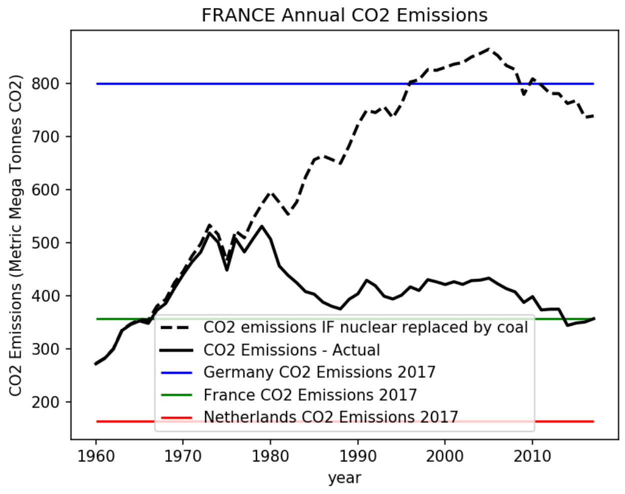


Fig. 23.

For a small country, Switzerland produces much nuclear power. Thanks to this production, Switzerland has kept their CO_2 emissions flat since the 1970s. Notice how much nuclear production has dropped off in Switzerland since the Fukushima disaster in 2011.

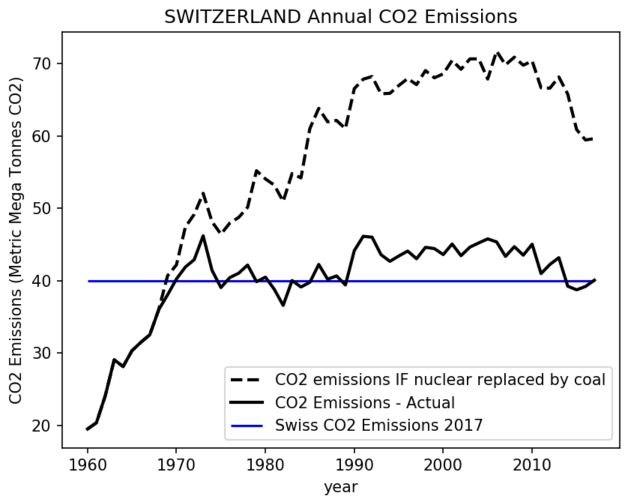


Fig. 24.

IX. Conclusion

The data support the conclusion that the use of nuclear fission reactors as a heat source in electricity generation has made a material impact on global CO_2 emissions. Given that the most likely technology to be used as a substitute for nuclear heat sources is coal burners, we estimate that nuclear fission reactors have prevented the release of 86.7 gigatonnes of CO_2 as of YE 2017. This is approximately 2.5 times the global CO_2 emissions of mankind worldwide from all sources in 2017.

Given the large difference that this technology has made in global emissions, it is concerning that the construction of new plants has slowed to replacement rate since 1985, and perhaps will fall below replacement rate in the future.

The one country where nuclear power does seem to have a future is China. China has exponentially increased their nuclear power output since the turn of the millennium. The pace of the construction of nuclear power plants in China has especially increased since the appointment of president Xi Jinping 7 years ago. This shows that the current leadership sees nuclear as a central piece in China's energy future. Given that China is the largest global CO_2 emitter by a factor of 1.9, and that much of these emissions are from coal fired power plants, the widespread implementation of nuclear power generation in China could make a meaningful reduction in global CO_2 emissions.

References

[1] Wikipedia contributors. International nuclear event scale — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=International_Nuclear_Event_Scale&oldid=897014290, 2019. [Online; accessed 17-May-2019].

[2] International Atomic Energy Agency. What is the power reactor information system. https://pris.iaea.org/PRIS/About.aspx, 2019. [Online; accessed 17-May-2019].

[3] United States Energy Information Administration. Consumption of fuels used to generate electircity, generation and thermal output. https://www.eia.gov/electricity/data.php, 2019. [Online; accessed 17-May-2019].

[4] United States Energy Information Administration. United states electricity profile 2017. https://www.eia.gov/electricity/state/unitedstates/index.php, 2019. [Online; accessed 17-May-2019].