Methods for Automated Fuel Cycle Facility Deployment

Robert R. Flanagan[1], Gwendolyn J. Chee[2], Jin Whan Bae3, Roberto E. Fairhurst[2] and Kathryn D. Huff[2]

1. Nuclear Engineering Program, University of South Carolina
2. Dept. of Nuclear, Plasma and Radiological Engineering, University of Illinois at Urbana-Champaign
3. Oak Ridge National Laboratory
flanagrr@mail.sc.edu

Developing nuclear fuel cycle scenarios requires knowledge of what facilities will be deployed as the simulation evolves. This can be done manual by a researcher, or it can be done computationally. Manual deployment requires a researcher to calculate the deployment of each facility in the nuclear fuel cycle. This can be simple for once-through fuel cycles; however, it is much more difficult for closed fuel cycles. These more advanced fuel cycles have complex interactions between different reactors types. Automated deployment can rely on several methods, from complicated logic to mathematical models (such as time series methods). Time series methods have been used historically in other energy-based utilities to predict solar radiation and wind behavior. By using these methods to predict the quantities and quality of nuclear material, it is possible to determine when nuclear facilities need to be built to match the demands of a nuclear fuel cycle.

This work expands upon previous work done using two specific types of time series methods; autoregressive moving average (ARMA) and autoregressive conditional heteroskedasticity (ARCH). Like the previous work, this also utilizes the d3ploy module for the Cyclus Fuel Cycle Simulator to do the automated deployment using predictive time series methods. This new work explores a much wider range of predictive techniques and demonstrates their unique capabilities with regards to the nuclear fuel cycle. These new methods include autoregressive integrated moving average (ARIMA), fast Fourier transforms (FFT), polynomial regression fitting, exponential smoothing, and the Holt Winters method. The implementation of each of these methods will be explained in this work, as well as their advantages and short comings regarding the nuclear fuel cycle.

I. Introduction

Analysis of nuclear fuel cycles often may be done to examine how future facility deployments might affect various fuel cycle metrics. Performing these deployments can be done either by hand (manual by a researcher), or automatically using software.

D3ploy (1) is a module for the fuel cycle simulator Cyclus (2) designed to automate facility deployment. This module uses the Cyclus Institution class as a basis to deploy facilities based on supply and demand curves. Facilities are deployed as individual facilities (i.e. a reactor represents a unique reactor building) and not fleets. This is because of the agent-based nature of the Cyclus architecture. D3ploy relies on several methods to predict the demand and supply in future time steps.

Three main types of methods are investigated. First, non-optimizing time series methods will be explored. The second type of methods are deterministic time series models. Finally, a system that implements a linear algebra solution to the fuel cycle is discussed.

The first class of time series methods are the non-optimizing (NO) time series methods. The two NO methods are Autoregressive Moving Average (ARMA) and Autoregressive Condition Heteroskedasticity (ARCH).

The determinist time series methods that will be examined are Fast Fourier Transform (FFT), Polynomial Fit Regression (Poly-fit), Exponential Smoothing, and Holt-Walters method. Each of these methods is implemented in the code through external software.

The final method investigated relies on solving a matrix representing the system of equations that characterize the demand and supply behavior of each facility. The discussion of this method is covered in more detail as the algorithm used to implement it inside D3ploy is more complicated than the other methods.

This literature review will briefly summarize the methods used. Each of these methods is used in several ways as time series prediction methods. Insights into how well these methods might work for the whole fuel cycle or specific parts of the fuel cycle are included with each overview. Additional how each method is implemented in the software will also be mentioned.

II. Methodology

II.A. Module Implementation

D3ploy operates within the Cyclus architecture and leverages the specific advantages that are unique to Cyclus. Due to the agent-based nature of Cyclus each agent inside Cyclus can operate independently during a simulation. This allows each agent to report to the system how effective it was during a given time step.

For example, a reactor can report to the simulation that it could produce only 80% of its specified power during a time step because it was down for refueling. It may also report that because it needs to go down for refuel it requires fuel. Since this is done on a reactor by reactor basis it allows for higher fidelity than fleet-based simulations.

As all facilities can report to the simulation how much of a given commodity they are supplying or demanding on a given time-step, D3ploy must be able to read these reports, and then determine if deployments need to be made to ensure that the simulation does not suffer from an under supply of material.

To accomplish this D3ploy records all demand and supply calls for any commodity that it tracks, and stores these for a given time step. These datasets are then used to drive the decision-making process for all the methods D3ploy employs except for the matrix method, which will be discussed in more detail in **Section II.E.**

Each time-step D3ploy sends the time series data off to the method that it is set to use. The method then predicts the demand and supply for a given commodity for the next time step (t+1). D3ploy determines if there is a gap in the demand and supply curves. If a gap is noted and supply is predicted to be less than demand, d3ploy will schedule the deployment of facilities for the next time-step. By default, deployment is done first by preference, then by largest facilities first, and finally filling in with smaller facilities to reach the smallest over supply possible. Facilities with a higher preference will be deployed more frequently than those with lower preferences. The user may choose to deploy by smallest facility first as an alternative option.

The choice of which time series prediction to use can be set per commodity. This allows the user to try different methods and see which operate best for a given commodity. This makes sense for the nuclear fuel cycle because in a realistic model, reactors request fuel in large batches all at once, whereas enrichment facilities might request material at a more constant rate. D3ploy ensures that these two demand and supply curves can be handled by different methods that might fit these behaviors better.

Although each commodity is tracked within D3ploy there is only one driving commodity. This commodity is typically the commodity that will require all other commodities in some form. For example, if the driving commodity is electrical power, this will require the deployment of reactors. Reactors require fuel to operate. As the number of reactors in the simulation increases, the demand for fuel will increase, and fuel fabrication plants will need to be deployed to meet this demand. On the other side, reactors also demand space for their spent fuel, so as the number of reactors increase the demand for spent fuel space will increase and D3ploy will deploy storage facilities to meet the demand. In this way, D3ploy is capable of generate a full deployment schedule for not only reactors, but all facilities in the model nuclear fuel cycle.

II.B. Demand Response

The base behavior of D3ploy is known as the Demand Response model. In this method, the system will not predict the future time-step, but instead base the deployment on the current time-step. This means that at time-step t, if supply is less than demand, d3ploy will schedule deployment of a facility in the next time-step. This behave ensures that supply is always lagging demand, but supply is still increasing along with the demand. In systems will slow demand growth, or in systems were facilities sizes are large compared to demand growth this may result in limited periods of undersupply situations. However, in situations with a fast-growing demand, or small facility sizes, this may result in many undersupply periods. This model was created to give a base line functionality. It facilitates two important capabilities.

First, it allows for a basis for analyzing the prediction methods. It represents the worst-case scenario. All other methods can be characterized by their improvement over the worst-case scenario. Without this measure methods could only be compared to each other, without any knowledge of how poorly the system behaves without prediction methods.

This model was also created to ensure that the operation of D3ploy occurs even if the time series method it is attempting to employ does not compute a value. Some of the time series methods require several values in a time series before operating with any degree of accuracy. Additionally, all the prediction methods fail on time series of length zero (with the exception of exponential smoothing). Therefore, the Demand Response method is required for at least the first time-step, but sometimes it is also used later if solutions cannot be found for the time series models.

II.C. Non-Optimizing

The first set of time series methods being investigated are non-optimizing (NO) methods. This is the Autoregressive Moving Average (3) (ARMA) and Autoregressive Heteroskedasticity (4) (ARCH). Each method has been used previously to forecast supply in other power related fields, such as solar (5)(6) and wind (7)(8) power generation. So, their ability to predict for energy systems has previously been investigated, however these methods have not been used explore nuclear fuel cycles.

II.C.1 ARMA

The ARMA method relies on two components, an autoregressive term, and a moving average term. These two terms can be seen in Eq. (1) and Eq. (2) respectively.

$$X_t = c + \epsilon_t + \sum_{i=1}^{p} \varphi_i X_{t-i} \qquad (1)$$

$$X_t = \mu + \epsilon_t + \sum_{i=1}^{q} \theta_i \epsilon_{t-i} \qquad (2)$$

Where c is a constant, ε is assumed to be a white nose error term, X is the time series of interest, μ is the expectation of X, φ and θ are weights, and p and q are the order of the AR and MA functions respectively.

The full ARMA equation is a combination of these two equations and can be seen in Eq. (3).

$$X_t = c + \epsilon_t + \sum_{i=1}^{p} \varphi_i X_{t-i} + \sum_{i=1}^{q} \theta_i \epsilon_{t-i} \qquad (3)$$

The variables here are the same in Eq. (1), and Eq. (2). The behavior of this function is autoregressive in nature and therefore will regresses to previous values, however the error terms allow this to catch a general underlying trend in the data. This does not make it ideal for growing systems, but in systems with spiking values it allows the predictor to slowly return to values after a dramatic increase.

Another downfall of this system is tied to the amount of data in the time series that it uses. If the full time series is used the moving average component of ARMA could cause issues. In a system with high growth rates, the range in the values of the time series can be quite large, driving the predicted value down due to earlier time series values. To reduce the effect of this issue, D3ploy allows the user to set the number of time-steps to be used in the calculation. Therefore allowing the user to effectively set how sensitive the prediction is to more recent values.

D3ploy implements the Statsmodels (9) Python library to determine the fit of the ARMA model and predict the value at the next time-step.

II.C.2 ARCH

Unlike the ARMA model, the ARCH model is not composed to two unique terms. Instead the ARCH model modifies the original moving average term by changing the error term in Eq. (1) into a combination of a stochastic term, and a standard deviation. This new error term can be seen in Eq. (4).

$$\epsilon_t = \zeta_t \sigma_t \qquad (4)$$

Where ζ_t represents the stochastic term (a strong white noise process) and σ_t is determined by Eq. (5).

$$\sigma_t^2 = \alpha_0 + \sum_i^p \alpha_i \epsilon_{i-1}^2 \qquad (5)$$

As with ARMA this method is an autoregressive method, however, their ability to handle volatile systems is much better than ARMA models (4).

The ARCH method is implemented in D3ploy using the ARCH (10) package for Python.

II.D. Deterministic

The deterministic methods are those that do not deal with randomized solutions. Therefore, if a deterministic method predicts using a time series, it will always produce the same prediction from that time series, no matter how many times the method is called.

II.D.1 Fast Fourier Transform (FFT)

FFT attempts to break a time series down into a combination of sine and cosine waves (or sine waves that are with different phases), to produce characteristic amplitudes and frequencies for the sine or cosine wave that would reproduce the time series (11)(12). There are several methods to do this. D3ploy uses SciPy (13) (scientific computing library for Python) to perform the FFT and the prediction of the next value in the sequence. The values of the FFT are calculated as in Eq. 6.

$$y[k] = \sum_{n=0}^{N-1} e^{-2\pi j \frac{k_n}{N}} x[n] \quad (6)$$

Where y[k] is the FFT, and N is the length of sequence x[n]. This FFT returns a real component and an imaginary component. In this work only the real component is used to predict the next value in the time series.

FFT could a powerful tool for simulating nuclear fuel cycles because of the cyclical nature of the fuel cycle. In systems containing many reactors the predictable nature of refuel cycles fits the operation of FFTs. Further study will need to be performed to understand the full capability of this method with large scale systems. However, other facilities in the nuclear fuel cycle don't operate on a cyclical behavior. FFT may not be the best solution for these facilities.

II.D.2 Polynomial Fit

Polynomial fit, or polynomial regression is a form of regression that attempts to fit an independent variable and a dependent variable as an nth degree polynomial (14). It has been used to model several different natural processes, for example disease epidemics (15). Polynomial methods typically use a least squares method to minimize variance between the time series values and the polynomial model.

D3ploy implements polynomial regression using NumPy (16) a computation library for Python. In order to allow for users to control over this operation D3ploy allows the user to pass the order of the polynomial as an import variable. NumPy minimizes the squared error term to determine the proper coefficients for the polynomial selected by the user.

The long cycle length of reactors might not fit polynomials well, however other support structures might be modeled well with polynomial fits. For example, conversion facilities and enrichment facilities do not produce material in large batches followed by long periods of no activity. These facilities instead operate in a more consistent fashion throughout the nuclear fuel cycle. In some cases, these might best be described by linear functions (a polynomial of one).

II.D.3 Exponential Smoothing and Holt-Winters

Exponential smoothing, like moving average techniques, uses the previous values to inform the prediction of the next value. However, in exponential smoothing all previous values are not weighted equally, instead the values are weighted by how far they are from the prediction time-step. Values at times further from the current time-step are given lower value than those closer (17)(18)(19). The goal of the exponential smoothing technique is to smooth out the data in the time series. This is done using a least squares method. Unlike other smoothing methods, for example moving average, this technique does not require a minimum number of observations in the time series to produce results. However, more accurate predictions are possible as the length of the analyzed time series is increased (to a limit, the weighting means that values very far back will eventually be of trivial worth). As simple example of how the smoothing works is demonstrated in Eq (7) and Eq. (8).

$$g_0 = x_0 \quad (7)$$

$$g_t = \alpha x_t + (1-\alpha) g_{t-1} \quad (8)$$

Where g_t is the predicted value for the next time-step, x_t is the value of the data series at the current time step, and alpha represents the smoothing factor. This factor must be between [0,1]. Eq. (7) shows that if there is only one value in the time series exponential smoothing still produces a predicted value, but that predicted value is equal to the current value.

The behavior of the smoothing factor is such that at $\alpha=1$ the predicted value is just the value of the current time-step. Values of α closer to zero reduce importance of time series values closer to the given timestep. This factor has a big impact on the level of smoothing that is done. Instead of choosing this value for the user, it is left open for the user to choose as an input to D3ploy.

The unique weighting of this technique may allow this method to be useful for simulations that have steep growth curves. In transition scenarios this method has the capability to better track the decrease in demand of one fuel type and the increase in others.

Holt-Winters is a special version of exponential smoothing that is designed to handle time series data that demonstrates some sort of seasonal trend (20)(21). This type of trending can be analogous to the refueling cycle demands of nuclear reactors, in which there are long periods of zero fuel demand, with peaks of high demand. It is possible to imagine these are season trends and therefore Holt-winters was added to D3ploy to specifically investigate if this method could handle realistic reactor behavior.

D3ploy implements the algorithm for exponential smoothing, and Holt-Winters from the Statsmodels package for python.

II.E. Matrix Solution

This section demonstrates the capability of a new method being added to d3ploy. This method relies on the use of a linear algebra solution to a matrix representing the relationship of all facilities in a fuel cycle.

Given that the relationship of facilities is known ahead of time by the user of the fuel cycle simulator it can by a represented as a system of equations. For example, take the simplified fuel cycle of front end, light water reactor, fast reactor, storage. In this fuel cycle each of these facilities has a relationship with at least one facility. The following set of equations represents this fuel cycle.

$$N_{LWR} * C_{LWR} + N_{FR} * C_{FR} = Power\ Demand \tag{9}$$

$$N_{LWR} * Pu_{LWR} - N_{FR} Pu_{FR} = 0 \tag{10}$$

$$N_{LWR} * F_{LWR} - N_{FE} * C_{FE} = 0 \tag{11}$$

$$N_{LWR} * (F_{LWR} * Pu_{LWR}) + N_{FR} * F_{FR} - N_S * C_S = 0 \tag{12}$$

Where, N_f represents the number of facility (f). F_{LWR} and F_{FR} represent the fuel demands of the two reactor types, C_f represents the capacity of facility f, and Pu_{LWR} is the plutonium produced by the LWR, and Pu_{FR} is the plutonium required by a fast reactor to operate.

In Eq. (9) the total power demand of the system is handled. This equation is set such that the combined power output of the fleet of LWRs and the output of the fleet of FRs must equal the power demand.

In Eq. (10) the relationship of the plutonium demanded by a fast reactor and that supplied by a single LWR is displayed. This relationship assumes steady state operation to start, however D3ploy is capable of updating these values as the system advanced through time, so that the plutonium demanded by the fast reactor can vary with time. For the system to operate perfectly without undersupply this equation is set to equal 0, the LWRs supply the exact amount of plutonium used by the FRs.

Eq. (11) shows the relationship between the LWR and the front end. It balances the amount of fuel required by the LWR with the fuel produced by the front end.

Eq. (12) shows the balance between the used fuel from the LWR and the FR and the storage capacity of the storage facilities. This assumes that all the waste from separation process that removes plutonium from the used LWR fuel is sent to storage.

Once these equations are produced, they can be converted into a matrix. Using the NumPy (16) linear algebra package this matrix is solved to produce the amount of facilities required at any given point in the simulation. However, this only holds true if the capacities and demands of each individual facility holds constant throughout the simulation. Instead, the matrix is solved each time step, using the change in driving commodity (in this case power demand) between the next time step and the current time step to determine the number of facilities to be deployed. Depending on the ratio of power generation capabilities of the reactors and the demand change in power, this may not be an integer value of facilities. For example, if an additional 100 MWe is required in the next time step, and a reactor can produce 1000 MWe, a full reactor is not required. In this situation D3ploy tracks the number of non-integer facilities required and deploys when that value reaches an integer value.

In D3ploy this works by setting the tracking value of a facility type to 0 at the start of the simulation. As the simulation progresses the amount of each facility required is added to this tracking number. For example, if in time step one D3ploy calculates that 0.6 reactors are required to meet demand. This number is added to the reactor tracking number, making it 0.6. A reactor is deployed in the next time-step because while a full reactor is not required, greater than zero reactors are required to meet the power demand. The tracking number for reactors is then reduced by one (the number of reactors deployed), making it -0.4. On the next time-step (t=2) another 0.6 reactors are required, therefore the tracking becomes 0.2 (-0.4 + 0.6). Again, greater than 0 reactors are required to meet the power demand and therefore a reactor is deployed, and the tracking number is reduced by one, making it -0.8. On the following time step (t=3) the number of reactors required is again 0.6, the reactor tracking number increases to -0.2 and a reactor is not deployed because the tracking number is less than zero.

This process is calculated for all facilities in the fuel cycle that are part of the system of equations for each time step.

There are numerous flaws to this method. First, it does not work for simulations that have large spikes in demand. For example, reactors requesting all their fuel at once on the refueling time-step. This method only works if supplies and demands are spread out per time-step. A simple work around to this is for a reactor to demand its fuel spread out evenly over its cycle length. This applies to all facilities in the fuel cycle.

Another limitation of this method is that currently new technology cannot be added during a simulation. This means that at present this method is incapable of handling transition scenarios where new reactor technology becomes available for deployment later in the simulation.

II. FUTURE WORK

D3ploy is a fully integrated Cyclus module that shifts the decision making of when to deploy fuel cycle facilities from the user to the software. While determining deployment schedules for reactors might be straight forward in fleet-based simulators or in cycles that do not include recycling of used fuel, it is non-trivial for more complex fuel cycles. D3ploy aims to shifts the development of deployment schedules away from human resources and on to computational resources.

This paper details the method by which d3ploy operates within the Cyclus Simulator and lists the methods that have currently been implemented into D3ploy. Previous work (1) has highlighted the capabilities of D3ploy for both the ARMA and ARCH methods. Numerical experiments that compare each of the methods listed in this paper need to be performed. These experiments need to highlight the benefits and shortcomings of each method for difference types of fuel cycle facilities, as well as different demand curves and growth rates.

REFERENCES

1. Flanagan, R. "Using supply and demand curves to determine facility deployment." (2018). Technical Workshop on Fuel Cycle Simulation Records, Paris, France.
2. Huff, Kathryn D., et al. "Fundamental concepts in the Cyclus nuclear fuel cycle simulation framework." Advances in Engineering Software 94 (2016): 46-59.
3. Woodard, D. B.; Matteson, D. S.; Henderson, S. G. Stationarity of generalized autoregressive moving average models. Electron. J. Statist. 5 (2011), 800--828
4. Bollerslev, T. (1986). Generalized autoregressive conditional heteroskedasticity. *Journal of econometrics*, *31*(3), 307-327.
5. Reikard, Gordon. "Predicting solar radiation at high resolutions: A comparison of time series forecasts." Solar Energy 83.3 (2009): 342-349.
6. Diagne, Maimouna, et al. "Review of solar irradiance forecasting methods and a proposition for small-scale insular grids." Renewable and Sustainable Energy Reviews 27 (2013): 65-76.
7. Soman, Saurabh S., et al. "A review of wind power and wind speed forecasting methods with different time horizons." North American power symposium (NAPS), 2010. IEEE, 2010.
8. Taylor, James W., Patrick E. McSharry, and Roberto Buizza. "Wind power density forecasting using ensemble predictions and time series models." IEEE Transactions on Energy Conversion 24.3 (2009): 775.
9. Seabold, Skipper, and Josef Perktold. "Statsmodels: Econometric and statistical modeling with python." *Proceedings of the 9th Python in Science Conference*. Vol. 57. Scipy, 2010.
10. https://github.com/bashtage/arch
11. Cooley, James W., and John W. Tukey, 1965, "An algorithm for the machine calculation of complex Fourier series," *Math. Comput.* 19: 297-301
12. Press, W., Teukolsky, S., Vetterline, W.T., and Flannery, B.P., 2007, Numerical Recipes: The Art of Scientific Computing, ch. 12-13. Cambridge Univ. Press, Cambridge, UK.
13. Jones, Eric, Travis Oliphant, and Pearu Peterson. "{SciPy}: Open source scientific tools for {Python}." (2014).
14. Peixoto, Julio L. "A property of well-formulated polynomial regression models." The American Statistician 44.1 (1990): 26-30.
15. Rohr, Jason R., and Thomas R. Raffel. "Linking global climate and temperature variability to widespread amphibian declines putatively caused by disease." Proceedings of the National Academy of Sciences 107.18 (2010): 8269-8274.
16. Oliphant, Travis E. A guide to NumPy. Vol. 1. USA: Trelgol Publishing, 2006.
17. Gardner Jr, Everette S. "Exponential smoothing: The state of the art." Journal of forecasting 4.1 (1985): 1-28.
18. Hyndman, Rob, et al. Forecasting with exponential smoothing: the state space approach. Springer Science & Business Media, 2008.
19. Hyndman, Rob J., and George Athanasopoulos. Forecasting: principles and practice. OTexts, 2014.
20. Chatfield, Chris, and Mohammad Yar. "Holt-Winters forecasting: some practical issues." Journal of the Royal Statistical Society: Series D (The Statistician) 37.2 (1988): 129-140.
21. Kalekar, Prajakta S. "Time series forecasting using holt-winters exponential smoothing." Kanwal Rekhi School of Information Technology 4329008.13 (2004).