A METHODOLOGY OF AUTOMATIC MODEL GENERATION USING MATLAB CODE

Technical Report

Submitted to:
The Office of Naval Research

Contract Number: N0014-08-1-0080

Submitted by:
Patrick R. Breslend, M. Steurer

November 2013

Approved for Public Release – Distribution Unlimited

Any opinions, findings, conclusions or recommendations expressed in this publication are those of the author(s) and do not necessarily reflect the views of the Office of Naval Research.
MISSION STATEMENT

The Electric Ship Research and Development Consortium brings together in a single entity the combined programs and resources of leading electric power research institutions to advance near- to mid-term electric ship concepts. The consortium is supported through a grant from the United States Office of Naval Research.
## Table of Contents

1 Executive Summary ................................................................................................................. 1
2 Introduction and Project Goal .................................................................................................. 2
   2.1 Task 2.2 Overview ........................................................................................................... 2
   2.2 Scope of work ................................................................................................................... 2
   2.3 Project Goal ...................................................................................................................... 2
3 Automatic Model Generation using a Script ........................................................................... 3
   3.1 Coding scripts to reduce research time ............................................................................ 4
      3.1.1 Model, document, and report with scripts ............................................................... 4
      3.1.2 Libraries of helpful functions for model building ..................................................... 5
      3.1.3 Proposed methodology for model construction using script .................................. 6
      3.1.4 Common components converted to scripted code .................................................. 7
   3.2 Automated modeling with complex architectures ............................................................ 8
      3.2.1 Adjusting model parameters, components, and designs from script ..................... 8
      3.2.2 Method for signal measuring using script ................................................................. 9
      3.2.3 Reduction in the need for user input ........................................................................ 9
      3.2.4 Automatic reporting and documentation ................................................................. 10
      3.2.5 Converting existing models into script .................................................................... 11
4 Current Limitations ................................................................................................................ 11
5 Future Efforts and Standards ................................................................................................. 12
6 Conclusion and Recommendations ....................................................................................... 12
7 Acknowledgements ............................................................................................................... 12
8 Appendix A: Tutorial ............................................................................................................. 13
   8.1 Read–me–First ................................................................................................................. 13
   8.2 Example on how to build MATLAB models using code ................................................. 13
LIST OF FIGURES

Fig. 1: Tutorial READ-Me-FIRST file used to explain the contents of the tutorial folder........... 13
Fig. 2: Example on how to build MATLAB models using code (pg. 1)........................................ 14
Fig. 3: Example on how to build MATLAB models using code (pg. 2)................................. 15
Fig. 4: Example on how to build MATLAB models using code (pg. 3)................................. 16
Fig. 5: Example on how to build MATLAB models using code (pg. 4)................................. 17
Fig. 6: Example on how to build MATLAB models using code (pg. 5)................................. 18
Fig. 7: Example on how to build MATLAB models using code (pg. 6)................................. 19
Fig. 8: Example on how to build MATLAB models using code (pg. 7)................................. 20
Fig. 9: Example on how to build MATLAB models using code (pg. 8)................................. 21
Fig. 10: Example on how to build MATLAB models using code (pg. 9)............................... 22
Fig. 11: Example on how to build MATLAB models using code (pg. 10)............................ 23
Fig. 12: Example on how to build MATLAB models using code (pg. 11)........................... 24
Fig. 13: Example on how to build MATLAB models using code (pg. 12)............................ 25
LIST OF TABLES

NO TABLE OF FIGURES ENTRIES FOUND.
1 Executive Summary

The purpose of this project was to investigate ways in which large system models can be quickly and easily generated in an automated fashion in MATLAB/Simulink in order to assess the behavior of a wide range of considered variants of a system. This approach is intended to increase the efficiency of constructing system models from sets of existing component models and allow the behavior and performance of alternative designs to be more easily explored. This will allow much more thorough investigations at the system level, as opposed to the study of small sets of point designs which must be manually constructed, potentially allowing system designs to be constructed and evaluated from separate optimization and trade-space exploration algorithms.

The focus of this work is on the development of automatic model generation for a shipboard power system, although the concepts and techniques used to develop such code may be equally applicable in other fields of study. The construction of dynamic and electromagnetic transient simulation models of shipboard power system can be time and labor intensive. The effort in constructing such systems is partially attributable to the overall number of components in the system. The components in the ship system are often repeated throughout the ship model, with instances differing only in the values of model parameters or the components to which they are connected. The research described herein focuses on the development of MATLAB functions used to add instances of common components of shipboard power systems to Simulink system models. The functions are developed with numerous options to allow flexibility in adding component models, but reasonable default values are provided for the optional arguments in order to minimize effort in making use of the functions. MATLAB scripts are written which make use of these functions in order to generate system models in Simulink, which can then be executed from scripts to assess the performance of the system for given scenarios. A modular approach is taken to allow functions for larger subsystems to make use of functions used to add component models. The model instances are parameterized, making use of variables in hierarchical structures in the workspace. This allows model parameters to be easily manipulated from scripts to facilitate parametric studies with the system models. However, the functions return structures with default values. This approach allows components and subsystems to be easily removed or added to system models, allowing for rapid model generation and more advanced system studies.

Since most components have already been modeled, the development of functions that can convert working models into code is the next step towards reducing research and development of more complex/ more dynamic models for automatic generation. Additionally, substantial effort is often expended to properly document a model. Proposed is a new way to build models using code that can also be documented with the press of a button. MATLAB provides a publishing function that allows the researcher to document code and models within the code itself.
2 INTRODUCTION AND PROJECT GOAL

In Task 2.2 strategies and methods are to be developed that will decrease research and development time in model building, user input, and report generation. Efforts have been made to develop a tutorial on how automatic model generation can be used on a MVDC ship system. The end goal of this work is to have all relevant components in a system to be coded in a specific MATLAB command structure that will lead to rapid prototyping. The coding will allow for quick placement of complex components into larger systems to decrease model development time. Simple coding has been shown to produce complex system studies that otherwise would take extensive work to setup.

2.1 Task 2.2 Overview

The primary goal of this task is to conduct research into techniques for math modeling that will speed up simulation times. Detailed models can require substantial time to run, so an average value model has been proposed to reduce simulation times significantly. Average value models for large ship system simulations will soon reduce simulation times. Simulation times may be reduced but model building still accounts for a large portion of the time devoted during research. The work explained within this document is an exploratory step to develop a methodology to reduce time spent on redundant tasks. The effort to develop a new technique for building and documenting models is not complete but the methodology is enough to display the benefits of a new modeling approach.

2.2 Scope of work

The original plan was to look into design approaches that could remove repetitive actions in model building, and enable automated model generation from external optimization and trade space exploration routines. Often in large models, such as the electric ship system, there are many sections of the model that are copies of other sections with different parameters. Research was also conducted regarding automatic documentation and report building to further reduce research and development times. The research has shown that development time could also be reduced using automatic model updates for parametric and system studies. Moreover, efforts in converting pre-existing models into scripted code for automatic model generation have begun.

The work presented in this report is meant to be broad and help model development become more efficient. The models used for example in this report are representations of possible models used in ship model research. The goal for a library of common components is still under current research and should be available in future ESRDC reports. The authors of this report demonstrated that redundant tasks can be commanded from a script which is still relevant whether or not the models used for example are complete. A tutorial was developed from this example using the proposed methodology of scripted code that builds a complex model, while also documenting the tutorial and proposed structure for model building.

The authors understand that MATLAB provides a Simulink GUI but this user interface can be time consuming in developing multiple test cases efficiently. The authors are well aware that MATLAB has saturated the various research fields making it hard to use anything but MATLAB
software. MATLAB is unique and allows any person in research the tools needed for research regardless of experience. The authors set out to propose efficient model building within Simulink allowing for others to expand further. Again, this is a methodology “A methodology does not set out to provide solutions but offers the theoretical underpinning for understanding which method, set of methods or so called “best practices” can be applied to a specific case.” The authors tried to be as thorough as possible to give examples, which would lead others to develop the idea further. The specific project that this work was intended to benefit went in a direction that was not suitable for further efforts to be spent in developing rapid model generation code. The work done and the idea presented are still valid when applied correctly and are merely considered as tools for building and reporting large models, which contain many instances of common components.

2.3 Project Goal

This project started with an exploration into the inner structure of MATLAB models and what possible methods could be implemented to speed up model generation.

The project goals completed and reported on include:

- Design methodology and development of a script structure for model building
- Electric ship system component example that can be used in a library of other component scripts for rapid model generation.
- Demonstrating automated building of complex system architectures with scripts
- Adjusting model parameters, components, and designs from scripts
- Insert and remove signal monitoring and test setups with script
- The amalgamation of model building, documentation, and reporting into one step
- Examination of possible methods to convert manually assembled models into scripted code
- Developing new techniques that could lead to standards in the future to further reduce model development time

It is obvious to the authors that models can be built with the Simulink GUI using copy, cutting, etc. The main title of this paper alludes to rapid model generation which cannot be done efficiently through the GUI alone. Obviously if one change is to be made to a model then it may be faster to cut, copy, or paste then simulate. Our research area consists of models which took weeks to generate. This is why the work was done to find ways in which MATLAB commands could be used to remove hours of cutting, coping, pasting, and renaming all the small subsystems of each component. Then the authors noticed that copying and pasting did not help organize parameters as they were placed throughout the layers of these complex models. Nor did the GUI help with model reporting because that was being done separately. The authors set out to focus
on developing the report to introduce a new methodology of rapid model building. This method is used for research where model parameters, components, node connections, and measurement points change frequently and doing things via the GUI is time consuming. Hence the need for a list of easy to use commands which can be easily tweaked and re-simulated more efficiently than today’s common practices. The full model code used in this research work is available upon request but consists of too many lines of code to justify appending to this report.

3  Automatic Model Generation Using a Script

This section is a summary of work completed and is not a complete solution to how researchers should model, document, or report their work. This work serves as a recommendation and exposure to techniques that suggest promising reductions in the development times for system models. There are plenty of areas that can be improved upon in the current research techniques. The work presented points to approaches for reducing development times, but the approach may benefit from further refinement. The idea of coding in a hierarchical structure is well known and is the reason why the authors selected this as their example to demonstrate the proposed methodology. There is no doubt from the authors that other code languages could be used for model generation. The language chosen to code with or the way in which one executes the concepts put forth in this paper will not change the point that model building has redundant tasks and could be simplified through strings of common tasks performed by code. Simulink GUI copying and pasting components to expand model complexity will quickly lead to naming confusion and parameters values not being clearly visible without digging deep into a model to find the corresponding component of interest and learning the name given to the parameter or the value assigned. The proposed strategy is a way to code models subsystems from the beginning in order to give parameters a hierarchical structure throughout the model being constructed.

3.1  Coding scripts to reduce research time

Coding scripts to control parametric studies is common practice in research. MATLAB has provided a platform for control of its simulation models via the command line. However, the documentation and examples for this approach are limited. A substantial amount of time was needed to fully understand the impact scripting commands could have on a model. It was found that any mouse click or keystroke in MATLAB can be programmed through the command line, thus leading to the obvious creation of automatic generation of models from scripts. The tutorial provided in Appendix A: Tutorial uses the function generatorSetTutorial to generate a model.

3.1.1 Model, document, and report with scripts

Scripts are useful tools in reducing time spent modeling, documenting, and reporting. The top level script is comprised of a few lines of code that calls for the model to be built, simulated, and recorded using their respective function calls. The top level build code will be responsible for creating a new Simulink system and populating it with the top level subsystems. Each preceding subsystem for a model is responsible for calling the functions for each of its subsequent systems. The subsystem function calls will propagate through the layers of the model until all subsystems are built. This means that each subsystem in the model has a script file to populate its system.
This is very important because it allows for select branches of a model to be modular, automated, and duplicated. The subsystem script will also search its system for all blocks and their respective port handles. The port handles will then be used to assign line connections between the blocks of a system. The model will now have all of its blocks properly connected as intended. This automated way of connecting lines is considerably faster than manually connecting each of the blocks. The system at this point should be completely built and waiting for a parameter update to set up a specific case.

Scripting the build of a model is beneficial to the researcher and other users who look through their work. It is common practice for engineers to comment their code to ensure complete understanding of a script. Commenting throughout each script is suggested, and acts as a complete documentation of the model. MATLAB has a publishing function which creates a document of the code. The style used to comment the code will determine the layout of the document. The publish function can make a document of the scripts code, pictures, and results all while having the ability to give detailed explanation throughout the published document. These documents can be automatically generated when the model code is called. Documenting is very important to research and it is normally worked on long after the model has been built and tested. This means that some information is bound to be lost in the documentation process and can make it much harder for other researchers to fully understand the model presented in technical reports. If this technique is applied, research documentation will become more uniform and set a standard for collaboration between researchers.

The use of scripts to develop reports is similar to the documentation steps. The report script will call the documentation scripts for each component while adding in sections with explanations from a system level perspective. It would be beneficial to the researcher to develop these reporting scripts while consecutively constructing the building scripts. The researcher could automate a substantial section of their reports through the details provided in building of the model.

3.1.2 Libraries of helpful functions for model building

In developing the scripting approach proposed it was quickly realized that common snippets of code were being used repeatedly and could be condensed into simple function calls that would return the same results. This was helpful mainly for locating certain sizes or positions of blocks within a subsystem.

3.1.2.1 Establish a simplified coordinate system

One obstacle to building a model from code is developing a system for arranging each block into the system and knowing where the next one should be placed. MATLAB has a (x, y) coordinate system in place that sets the origin at the top left corner of each system at (0, 0). The system from the top left corner increases in the positive x and –y direction. MATLAB then defines each of its blocks by the diagonal points, top left and bottom right, of the block as [x1, y1, x2, y2]. This makes placement of blocks into a system difficult because you need to know the diagonal points of each block and how to decide the diagonal points of the next block.
The solution proposed was to develop a grid of the system that would help describe where each block would be placed. The idea is that the grid would simplify the position placement by making the grid lines in increments of 60. At each grid intersection a grid position was placed. For example the positions (0, 0), (1, 1), and (2, 3) have physical positions (60, 60), (120, 120), and (180, 240) respectively. With the grid system defined the problem now was to define how each block would utilize the grid to simplify model building. The decision was made to simplify placing the position of blocks not by the diagonal points but by their center. This would mean that a block positioned on the grid at (1, 1) would have its center at (120, 120). A default size for the blocks was made to fit the grid system. The blocks default size is 30x30 to fit in the grid system. The grid and block system may look complicated but it was simplified by the functions location.m, origin.m, and posReference.m. The function location is used to convert a position in the grid coordinate system to a position in the MATLAB coordinate system. The function origin is a quick way for one to locate the center of a block. The function posReference is a way to set a blocks position to a certain grid spacing away from a reference point see Appendix A: Tutorial.

If a block with default width and height is placed on the grid at position (0, 0) then the location function will return a MATLAB block position [45, 45, 75, 75]. This will allow the script to be written for example, \( pos = \text{location}([1, 1]) \), where \( pos \) will have the value [105, 105, 135, 135]. This function has options in place to set the width and height of the block if different from the defaults. The code would then be written \( pos = \text{location}([1, 1], 'w', 60, 'h', 120) \) and \( pos \) would be equal to [90, 60, 150, 180]. It should be apparent that placing blocks is as simple as picking one point on the grid and the code does the work of locating the actual coordinates.

When building a system its more intuitive to place blocks in reference to another block already placed in the system. This allows the code to systematically place blocks in the order that they would appear in the system naturally. For example a turbine might be followed by a generator in a ship system. The generator block could then be placed a grid position away from the turbine. The \( pos \) of the generator would be calculated as \( pos = \text{location}([2, 0], 'ref', 'sysName/turbine') \). If the turbine was placed at \( \text{location}([1, 1]) \) then the generator would have the \( pos \) equal to [235, 105, 255, 135] which has the same \( pos \) value if \( \text{location}([3, 1]) \) was called.

All of this work led to the final proposed approach to the issue of placing blocks in a system. The first block of a system will be defined as the block with grid position closest to location (0, 0). All succeeding blocks will be placed in reference to the preceding block position. This allows the model to expand, as new blocks are added into a system, by shifting the preceding blocks accordingly.

### 3.1.3 Proposed methodology for model construction using script

A model will consist of a top most layer which holds the main system blocks and can be seen as the overall architecture of the model. The subsequent systems can be found within each system in the top layer creating a second layer of blocks in the model hierarchy. This structure continues to create lower layers with more blocks until the model completely describes the system. A structure was needed to begin building models from code that would be consistent from layer to layer.
The script for each system is actually considered as a function since commands need to be sent in and responses need to be returned. The possible commands that could be sent into the function may not be known. A known method for coding such functions was implemented allowing for a change in the amount of inputs to a function. This allows the systems to be updated in the future with more optional command inputs that can propagate through the layers. The system is also designed to output its parameters into an output structure that is named when the function is called. The output structure is a roadmap of how the modeled system is constructed with all the settings of the components within.

The first part of each component script will have an opening section that assigns all input arguments. The path to which layer the component will be placed and the specific location are the only commands needed to place components into a model. The other available commands are all optional and can be added as needed. The optional input arguments are given defaults to allow the models to be constructed without the need to set specific parameters during development. These defaults can be adjusted in the code but the parameter can also be updated before simulation by controlling the parameter structure that is available from outside the model.

The next section of the script is where the components layer is populated with all the components it contains. An example of this would be a function called \textit{generationSet} that populates the generation set layer with all subsequent layers including exciter, prime mover, and generator layers. This section of the code is best organized in the order in which components are placed in the model from left to right and top to bottom. This is beneficial when it comes to connecting blocks in later sections.

After all subsequent components are placed in the layer the script will collect port handles. The port handles are the locations for the ports of each component. The port handles are then used in the next section of the script to add connection lines between the components. Adding lines to a model can be set to auto-routing which find the best path for the line to connect. One issue that was found is that not all blocks in MATLAB have the same type of line connections which can be seen in Fig. 12.

The last section of the script is used for returning the output structure of the layer and the option to store identifiers like the type of layer or some description of the layer. The extra identifiers were put there as an option for the script to output more details about the component layer. This was not explored any further but would be an area for future work. Some possible scenarios where this would help are when models are passed along to other researchers not familiar with the individual component layer. Users could be given extra information about the optional commands that can be called or ways in which the model should be used to get desired outputs.

\subsection*{3.1.4 Common components converted to scripted code}

The majority of complex electrical systems are normally comprised of a few common components. This means that models of these complex systems can be simplified into script that will build these common components when needed. If a group of components form a common subsystem then that subsystem could also be scripted into a function call. For example the MVDC system for an electric ship may have four generation points in its system. Each generation point has common modeling components within its subsystem e.g. exciter, turbine,
generator, etc. If each component’s subsystem is scripted then the next level in the system can be scripted to call each component. Now the entire model can be built from one function call at the top most level.

### 3.2 Automated modeling with complex architectures

The model has been simplified to an automated structure of scripts which call each other in the generation of a model. The complexity of a model is bundled up into one function call that automatically runs through the model space populating it with all its systems and subsystems. This turns each system, no matter how complex, into a simple command that can be duplicated as needed.

#### 3.2.1 Adjusting model parameters, components, and designs from script

The proposed approach allows the user to adjust the model from a global perspective without having to delve into each subsequent system. The idea is that if a lower level parameter, component, or design needs to be changed then the user can make that change from the top level of a system. The researcher might want to make a change to a generator’s phase winding resistance within a complete ship model. The researcher can code the model building script to make the necessary changes on the parameters value without opening the model and each subsequent subsystem to reach the parameter input in the user interface. Others have proposed such scripts but it was the authors intention that this feature be added to the methodology for rapid model generation because it allows the coder to build without naming each component and subsystem by hand. This minimizes effort needed to change a model between simulations.

The parameters of a model are output from the lowest level all the way up to the top level of a model as a structure. Each parameter is defined by its placement in the system’s layers. The notation for parameters of a specific component could be simplified to the structure `model.topLevel.subsystem1.subsystem1_1.component.parameter`. The parameter list is exported when the model is built and can be accessed from the workspace. When the model is simulated the parameters in the workspace propagate through the model to update any changes. This parameter list can be exported as a script to allow the user to perform advanced parametric studies.

The components of a model might also need to be changed in between simulations. For example a system study could be run on the ship where different models for the exciter could be switched out with a simple option added to the component system call. This ultimately will allow the system components to be designed and named differently for the user to implement where needed. The researcher could switch out different generator models in an automated script to produce plots which would compare their effect on the overall system. The current method to implement such a study takes hours of setting up and two separate models for comparison need to be edited and saved. This also allows the researcher to update all the scripts that are using a specific component and change them all out for another component to see the resultant change in simulation results.

Changing components in a model manually can be somewhat simple but changing the layout of a system between simulations is not realizable in current research techniques. The proposed
approach methodology allows the researcher to change sections of code in the script with new code which would change the layout of the model. The script can then be set in an automated fashion that would allow the user to rebuild entire sections of models while running cases and storing results. If this process were automated it would open up new system studies that previously were conceived in a piecemeal fashion that took extensive amounts of development time. This simple example illustrates the types of automation that are now available through simple scripts that call the functions for the researcher. There no longer is a need for the researcher to manually navigate through complex models to make changes. This will reduce errors in collaboration since the list of changes made to a system are written in code and can be easily recorded and thus seen by others using the model.

3.2.2 Method for signal measuring using script

When running simulations on a model the researcher is interested only in specific signals, variables, outputs, inputs, or a combination of them. If this is the case then the researcher will have to add blocks such as scopes, toWorkspace, toFile, etc. The steps needed to locate these blocks in the library, place them in the system, connect the lines, and name the value being measured is tedious and redundant. A few functions have been made to show that these steps could also be automated in a simple method. The researcher will call the function and provide it with subSystemName/blockName, port to measure, and desired signal name. The result is a measured port that will be stored under the signal name provided. The researcher can copy and paste these calls and change the signal ports to be measured for the current simulation. Signal measurements could also be scripted as a list of specific setups to perform different studies depending on what is being researched about the model.

The signal measurement functions contain defaults as well as the ability to have options set. For example, the inputs and outputs of a system are of interest therefore a call for all ports to be measured is set. The default call will measure the first output of the component. A scenario can be developed to automate and lower the time needed to measure a specific component throughout the model. The researcher would search for the specific name throughout the script and add the code needed to measure each component.

The proposed solution to measuring signals can also be done on components not in the standard Simulink library e.g. Simscape. These functions use a different method to locate the block ports for measurement, and automated or coded methods are still unknown.

3.2.3 Reduction in the need for user input

The proposed methodology is an organized approach for researchers to take full control of their models while decreasing the complexity of the work needed to design, develop, and execute a study. The ways in which a user can reduce manual input to the model has been expressed in earlier sections. A model will eventually hit a limit in the minimum simulation time needed. So other areas that effect research time need to be considered to reduce overall research and development time. Scripted modeling approach allows for the user input of traditional research to be done without the user. To simplify it in words “every possible action that can be made in MATLAB can be written as a line of code to be executed in a script”. It is up to the researcher to
amalgamate manual and automated control of their models to produce the most time efficient approach.

3.2.4 Automatic reporting and documentation

The single most important part of a researchers work is to report their findings for the advancement of science as a whole. Reporting research results is much easier if the approach is well documented and the descriptions of the salient features are apparent. The current methodology to produce reports normally results in rushed work to meet deadlines which ultimately leads to possible loss of important information. The proposed approach simplifies the documentation steps and develops a systematic approach to producing proper detailed reports. The researcher will need to change their current method and adopt this modeling approach which could be a difficult transition at first but possibly may become the preferred approach.

The first step is for the researcher to choose a system to be converted into scripts as described in earlier sections. MATLAB allows the user to publish code in a desired format such as Word, pdf, html, etc. This is not just a simple printout of the code but a way for the user to comment sections of the code for detailed documentation of each section of the code. The published document is well formatted while including code, comments, and outputs with a title and table of contents.

Since documentation is so important a researcher needs to document each and every block, component, subsystem, and overall system as they construct and develop the model. This is important because current modeling work is not described in detail and is difficult to follow for others using their models. The documentation would be throughout the code compelling the researcher to explain naming conventions, inputs, outputs and the overall design methodology. This will prove to be the most beneficial improvement to modeling techniques which in turn reduces documentation time.

If the researcher keeps this approach throughout the code then a standard documentation style will be developed that provides an easy to read and understandable product. Developing a systematic way to comment the code will allow for template code and ways to automate the documentation. It is possible for the code to output documents that automatically comment the main details such as the variables, parameters, inputs, and outputs of the system reducing documentation time for each simulation conducted. This is ideally used to display the results from each simulation run into a readable report for distribution.

The researcher can also insert into their code images, links, figures, tables, etc. to aid in documentation explanation. This may seem like a lot of added work for the researcher but this will reduce the construction and description of the final report. The final report can appendix the documentation files generated by the code and the researcher can use the report to actually describe the results. This approach really helps when a specific model is used to run many different simulations. The researcher will have already documented his work previously and will only need to report the new test and the new results. The production of reports will ultimately become formatted in a script that is commented and performs simulations and results automatically. The overall idea is that documentation and report times can be drastically reduced by approaching them from the scripting methodology proposed.
In the tutorial presented a twin shaft gas turbine, exciter, and generator along with their internal components were modeled using script to create an overall generation set model. The code used to generate the model also created the tutorial provided in the Appendix A: Tutorial. The tutorial given in the Appendix was generated from the model’s code using the report generator in MATLAB. The MathWorks website uses this method to generate MATLAB component tutorial/help from MATLAB component’s commented code.

The top level script in component generation code consists of just a few lines of code to build the whole model and can be duplicated quickly to produce a more complex ship system. The researcher has the ability to leave information imbedded in the code that can document and develop user manuals for using the models. This would eventually lead to libraries of model scripts that could be open sourced to collaborating researchers to further advance science while reducing overall development times across the profession.

3.2.5 Converting existing models into script

The overlying issue while reading this report and trying to grasp this new methodology is that it has an initial learning curve that would deter most researchers. The solution to this issue is to develop functions which convert existing models to code for the researcher. This has been proven in the tutorial that a system can be picked apart and used to construct a string of the code needed to build that model. The variables, signals, parameters, sizes, connections etc. can be found in a model. If all the information for a model can be found then the model can be reconstructed by lists of functions which reconstruct the model to be an exact copy. There is little to no need for the researcher to manually comment the generated code. The bulk of the commenting can be done automatically through the $model \rightarrow code$ functions. The model conversion code has not been explored in depth because the applications for such code are extensive. It would require some substantial coding to be robust enough to handle all the existing styles of models built manually. The most promising way to accomplish such a task was recognized while opening a model file (.mdl) in a text document. The file viewed in a text editor is found to contain each and everything in a model. It seems possible that this file could be loaded in a script which decodes and produces a string that is in the wanted script format for model building.

4 Known Limitations

The limitations of this work exist in the lack of a complete detailed description of guidelines or a standard which would unify all this newly proposed methodology. It has been shown in the Appendix A: Tutorial that the methodology presented can and will reduce research and development time across all research areas of science which utilize MATLAB. There should be further research efforts made to develop a hard guideline for how a model should be built, documented, and reported so it can be coded for automation. The current code was developed to prove the concept that research time can be reduced in many areas but it is neither complete nor robust. The overall task was to find areas of research and development times that could be reduced. The limitation will continue to be the code, until a unified set of guidelines are developed.
Other limitations that need to be explored are ways in which this new technique of combining modeling, documentation, and reporting can be done in other simulation software. Again this is limited by the guidelines for model documentation. The code will always be able to meet the guidelines set, yet coding models that do not follow a standard will limit the effectiveness of such a methodology.

5 Future Work

The future efforts should be to tackle the current limitations that have been described. A guideline for building models should be researched and developed. If all subsystems of models followed a similar structure to the tutorial model then this could be the first steps in developing a standard that everyone could follow. Other efforts could be made to design better model building functions to further simplify the work. Subsections of scripts could be simplified into new functions to condense the codes overall length and reduce redundant explanations in the documentation. Substantial parts of this work can be directly implemented now to help reduce time between simulations and documentation. The effort that would have the biggest impact on the future implementation of this methodology is the work in model to code conversion functions. If this was completed then all models in existence could be converted to script for use in automation in future models.

6 Conclusion and Recommendations

The work has come to a stopping point and has been left for future researchers to explore and develop into a final tool. The takeaway message here is that current research is inefficient and the tools capable of fixing this are outlined in the report. Hopefully, this report can serve as a reminder that as the complexity of models increases so does the development time. There is no reason that a set of guidelines could not be developed which help to streamline modeling, documenting, and reporting. Until then this work can be used as is with the hope of implementation in future rapid model generation efforts.

7 Acknowledgements

This work could not have been completed without the help and guidance of J. Langston, C. Dresner, M. Steurer and the power systems group at Florida State University.
8 APPENDIX A: TUTORIAL

A tutorial has been created to walk researchers through scripted model building of a twin shaft gas turbine. The turbine was chosen for the tutorial because it contains many subsystems and is a common component found in student’s research.

8.1 Read–me–First

Inside the Tutorial folder there is a Read-me-first document which explains how to use the tutorial. It can be seen below in Fig. 1.

8.2 Example on how to build MATLAB models using code

The generatorSetTutorial.pdf below in Fig. 2 through Fig. 13 is designed as a walkthrough step by step of how the model is built from each line of code. Note the .pdf was created from commenting the model code and using the publishing button on MATLAB’s toolbar.
Example on how to build MATLAB Models using Code

Table of Contents
Generator Set Tutorial Function: ................................................................. 2
Width option ......................................................................................... 2
Height option ....................................................................................... 2
SubSystem block Generator Set .............................................................. 3
Populate Subsystem Generator Set ......................................................... 3
Adding Notes or Titles to a Model ............................................................ 10
Get Port Handles for Simscape Blocks .................................................. 10
Add lines to connect blocks .................................................................. 11
Return default values as part of a structure ........................................... 12

Authors: Patrick Eresland & Chris Dresner
Contact: Patrick.Eresland@gmail.com or cwendser@gmail.com

In this example a model called 'Tutorial' was built and the function generatorSetTutorial was called to populate the model. The Tutorial folder given with this example contains the necessary functions needed for this example to run.

• Open MATLAB.

• File -> Set Path -> Click Add with Subfolders (Browse for the Tutorial folder given) -> Press OK -> Save and close.

• Set Current Folder to Tutorial as shown below

Below is the MATLAB expression used to edit publish configurations:

```matlab
% Code needed to set up generatorSetTutorial example.
modelName = 'Tutorial';
topLevel(modelName);
genset = generatorSetTutorial([modelName '/Generator Set'], [1,1]);
tutorialOpen(modelName);
```

Fig. 2: Example on how to build MATLAB models using code (pg. 1)
Generator Set Tutorial Function:

The input variable to this function is `varargin` (variable argument in). The variable `varargin` allows any number of arguments to a function. For further reference on using `varargin` type (=> help varargin) to the command line.

```matlab
function os = generatorSetTutorial(varargin)

The output `os` is set to be a structure and will contain all the default variables for the model.

```varargin``` = struct();
```

path is the full path for block destination including the name of block, path is the first argument of the input `varargin`

```varargin``` = varargin{1};
```

loc is the specific [x,y] position from a point where the SubSystem will be added to the model. loc is the second argument of the input `varargin`

```varargin``` = varargin{2};
```

x = loc(1);
y = loc(2);
```

`opts`, is a structure made of option pairs that can be set by the input `varargin`. In this function there are three option pairs ref, w, and h.

```matlab
opts = struct('ref', '', 'w', '', 'h', '');
```

`opts` is updated with all input option pairs given in the input `varargin`

```matlab
opts = updateOptions(opts, varargin, 2);
```

Width option

Ex: ```matlab
>> generatorSetTutorial(path, [x,y], 'w', 60);
```

```matlab
if isempty(opts.w)
w = 120; % Default width for subsystem
else
w = opts.w; % Optional width for subsystem
end
```

Height option

Ex: ```matlab
>> generatorSetTutorial(path, [x,y], 'h', 30);
```

```matlab
if isempty(opts.h)
h = 60; % Default height for subsystem
else
h = opts.h; % Optional height for subsystem
end
```

Fig. 3: Example on how to build MATLAB models using code (pg. 2)
SubSystem block Generator Set

If statement used to determine if the location \([x,y]\) given is in respect to the origin or in reference to a previous block's position.

Ex:>> generatorSetTutorial(path, [x, y], 'ref', [path '/previousBlk']);

if (isempty(opts.ref)) % default reference is the origin
    pos = location([x, y], 'w', 'w', 'h', 'h');
    add_block('built-in/Subsystem', path, 'MakeNameUnique', 'on',
    'position', pos);
else % references a previous block
    pos = location([x, y], 'ref', opts.ref, 'w', 'w', 'h', 'h');
    add_block('built-in/Subsystem', path, 'MakeNameUnique', 'on',
    'position', pos);
end

The Generator Set SubSystem has been added to the model 'Tutorial' shown below. The subsystem is empty and the next section will populate the Generator Set.

---

Populate Subsystem Generator Set

This section is where all the blocks/subsystems within the generator set are called to be added to the model. Below each block's type and name are commented before each subsequent code to help break up the lines.

% subsys - Prime Mover
os.primeMover = primeMover([path '/Prime Mover'], [0, 0])

os =

    primeMover: [1x1 struct]

The Prime Mover block displays another example of adding a subsystem block just like the Generator Set function. Within the primeMover function you will find similar code for building and populating a

---

Fig. 4: Example on how to build MATLAB models using code (pg. 3)
Example on how to build MATLAB models using code

subsystem. To view this code type(>> open primeMover;) in the command window.

Shown above is the resulting os.primeMover structure that will be passed to the top level of the model. It contains variables set to default values and can be changed by accessing the model structure in the workspace.

- When adding a subsystem block we always start with os and the name of the structure we wish for it to return. In this case we will use os.primeMover.
- The output structure os.primeMover is the same name as the function we wish to call, which is primeMover.
- The parameters that must be passed to the function is the path, the name of the block and the location.

```matlab
% Gain - kl
pos = location([1.5,1], 'ref', [path '/Prime Mover']);
add_block('built-in/Gain', [path '/kl'], 'MakeNameUnique', 'on', ...
    'position', pos, 'Gain', 'is.kl', 'sampletime', 'is Ts');
```

The first input to the add_block function is the library path. The Gain block can be found in the 'built-in' library which contains the most commonly used simulink blocks. In the add_block function for gain - kl two parameters were given variables for their value(Gain,'is.kl','sampletime','is Ts'). 'variableName' for input structure will be used to set default values during the build process and are of the variables to be changed from the workspace. At the end of the code all 'variableNames' are given their default value for the initial build. To change the default values the output structure must be updated before running the simulation.

```matlab
% Simulink-PS Converter - sps1
pos = location([1,0], 'ref', [path '/kl']);
add_block('Simulink-PSConverter', [path '/sps1'], ...
    'MakeNameUnique', 'on', 'position', pos, 'ShowName', 'off');
```

Fig. 5: Example on how to build MATLAB models using code (pg. 4)
Example on how to build MATLAB Models using Code

The `simulink`-PS Converter is essential to any Simscape/Simulink model. Notice that the `simulink` utility is not the standard built-in library. For this command to work the `psd` utility library must be loaded. When this tutorial model was built using `tutorialBuild`, a function called `topLevel` was called. This is where all the necessary libraries are loaded. The `topLevel` code can be seen below.

```matlab
function topLevel(varargin)

modelName=varargin(1);
if nargin(2);
    opts=struct('Option1','Option2');
end

% Close system if opened
if bdIsLoaded(modelName)==1
    bdClose(modelName);
end

% Open System and give it a name
new_system(modelName); % create a new system with the name given
open_system(modelName); % Open the system you named

% Open all libraries needed for the blocks to be called
load_system('fi_lib'); % Simscape Electrical Library
load_system('nesi_utility'); % Simscape utility Library
load_system('simulink'); % Simulink Library
load_system('powerlib'); % PowerSystems Library
load_system('PSQb_lib'); % Simscape Libs
end

% Connection Port - w
pos = location([1,0], 'ref', [path '/mpsl/'], 'h', 15);
add_block('powerlib/Elements/Connection Port', [path '/w'],
    'MakeNameUnique', 'on', 'position', pos, 'orientation', 'left',
    'Side', 'Right');

% Goto - goto_w
```

Fig. 6: Example on how to build MATLAB models using code (pg. 5)
Example on how to build MATLAB models using code

```matlab
pos = location([1.5,1], 'ref', [path '/k1'], 'w', 60, 'h', 15);
add_block('Simulink/Signal Routing/Goto', [path '/goto_w'], ...
    'MakeNameUnique', 'on', 'position', pos, 'GotoTag', 'w', ...
    'IconDisplay', 'tag', 'TagVisibility', 'global', 'ShowName', 'off');
```

% Simulink-PS Converter - sps2
```
pos = location([3,0], 'ref', [path '/Prime Mover']);
add_block('mbs1_utility/Simulink-PS Converter', [path '/sps2'], ...
    'MakeNameUnique', 'on', 'position', pos, 'ShowName', 'off');
```

% QD Axis Synchronous Machine
```
pos = location([2.5,-0.8], 'ref', [path '/sps2'], 'w', 140, 'h', 120);
add_block('QD_lib/QD Axis Synchronous Machine', ...
    [path '/QD Axis Synchronous Machine'], 'MakeNameUnique', 'on', ...
    'position', pos, 'BackgroundColor', 'LightBlue');
```

Fig. 7: Example on how to build MATLAB models using code (pg. 6)
Example on how to build MATLAB Models using Code

Fig. 8: Example on how to build MATLAB models using code (pg. 7)
Example on how to build MATLAB models using code

Fig. 9: Example on how to build MATLAB models using code (pg. 8)
Fig. 10: Example on how to build MATLAB models using code (pg. 9)
Adding Notes or Titles to a Model

Adding a Title to the model will help with documentation and description of models/subSystems.

% Note - Generator Set
add_block('built-in/Note', [path '/Generator Set:'], 'Position', [525, 71],...
'HorizontalAlignment', 'left', 'FontName', 'arial', 'FontSize', 13,...
'FontWeight', 'bold');

% Note - Generator Set Contents
add_block('built-in/Note', ...
sprintf([path '/Synchronous Machine\nPrime mover\nExciter']),...
'Position', [525, 91], 'HorizontalAlignment', 'left', ...
'FontName', 'arial', 'FontSize', 16);

Shown below is the subsystem Generator Set with title and description text added from the previous two add_block calls. Next we will locate all the port handles necessary for line connections.

Get Port Handles for Simscape Blocks

From our experience we have found three block types (simulink, simscape, and simulink/simscape) so far. To connect lines to Simscape ports or blocks the portHandle is needed. The code below are the calls to find all the necessary port handles and store them to a variable.

spscPH = get_param([path '/sps2'], 'PortHandles');
spsc1PH = get_param([path '/sps1'], 'PortHandles');
wPH = get_param([path '/w'], 'PortHandles');
QDampPH = get_param([path '/QD Axis Synchronous Machine'], 'PortHandles');
psc2PH = get_param([path '/psc2'], 'PortHandles');

Fig. 11: Example on how to build MATLAB models using code (pg. 10)
Example on how to build MATLAB models using code

```matlab
psc7PH = get_param([ path '/pss2', ],'PortHandles');
psc08PH = get_param([ path '/pss3', ],'PortHandles');
pssc5PH = get_param([ path '/pss6', ],'PortHandles');
pssc10PH = get_param([ path '/pss4', ],'PortHandles');
pssc11PH = get_param([ path '/pss5', ],'PortHandles');
gnd1PH = get_param([ path '/Gnd1', ],'PortHandles');
gnd2PH = get_param([ path '/Gnd2', ],'PortHandles');
Dgen1PH = get_param([ path '/D_gen1', ],'PortHandles');
Qgen1PH = get_param([ path '/Q_gen1', ],'PortHandles');
VSS1PH = get_param([ path '/Voltage Source (SL)1', ],'PortHandles');
scPH = get_param([ path '/Solver Configuration', ],'PortHandles');
excPH = get_param([ path '/Exciter', ],'PortHandles');
```

For Simulink ports, no port handles are needed. Simply refer to its port by using 'blkName/portNum'.

Add lines to connect blocks

Simulink blocks use output ports to import commands for line connections.

```matlab
add_line (path, 'Prime Mover/1', 'kl/1', 'autorouting', 'on');
add_line (path, 'Prime Mover/1', 'sps2/1', 'autorouting', 'on');
add_line (path, 'kl/1', 'goto_w/1', 'autorouting', 'on');
add_line (path, 'kl/1', 'spsl/1', 'autorouting', 'on');
add_line (path, 'pss5/1', 'goto_Ikd_gen/1', 'autorouting', 'on');
add_line (path, 'pss4/1', 'goto_Ikq_gen/1', 'autorouting', 'on');
add_line (path, 'pss3/1', 'goto_Iq_gen/1', 'autorouting', 'on');
add_line (path, 'pss2/1', 'goto_Id_gen/1', 'autorouting', 'on');
add_line (path, 'pss1/1', 'goto_te_gen/1', 'autorouting', 'on');
add_line (path, 'pss6/1', 'goto_Td_gen/1', 'autorouting', 'on');
add_line (path, 'Exciter/1', 'Voltage Source (SL)1/1', 'autorouting', 'on');
```

Simscape and Sim Power Systems blocks connect using straight connections for their lines. Some of these blocks can have imports or outputs as well and will use the Simulink line code for connecting to these ports.

```matlab
add_line (path, pssc1PH.RConn(1), wPH.RConn(1), 'autorouting', 'on');
add_line (path, psscPH.RConn(1), QDasmPH.LConn(6), 'autorouting', 'on');
add_line (path, QDasmPH.LConn(1), pssc2PH.LConn(1), 'autorouting', 'on');
add_line (path, QDasmPH.LConn(2), pssc7PH.LConn(1), 'autorouting', 'on');
add_line (path, QDasmPH.LConn(3), pssc8PH.LConn(1), 'autorouting', 'on');
add_line (path, QDasmPH.LConn(4), pssc10PH.LConn(1), 'autorouting', 'on');
add_line (path, QDasmPH.LConn(5), pssc11PH.LConn(1), 'autorouting', 'on');
add_line (path, QDasmPH.RConn(1), pssc9PH.LConn(1), 'autorouting', 'on');
add_line (path, QDasmPH.RConn(2), Dgen1PH.RConn(1), 'autorouting', 'on');
add_line (path, QDasmPH.RConn(2), excPH.LConn(1), 'autorouting', 'on');
add_line (path, QDasmPH.RConn(3), excPH.LConn(2), 'autorouting', 'on');
add_line (path, QDasmPH.RConn(3), Qgen1PH.RConn(1), 'autorouting', 'on');
add_line (path, QDasmPH.RConn(4), gnd2PH.LConn(1), 'autorouting', 'on');
add_line (path, QDasmPH.RConn(5), VSS1PH.RConn(1), 'autorouting', 'on');
add_line (path, QDasmPH.RConn(6), scPH.RConn(1), 'autorouting', 'on');
add_line (path, QDasmPH.RConn(6), gnd1PH.LConn(1), 'autorouting', 'on');
```

---

Fig. 12: Example on how to build MATLAB models using code (pg. 11)
Return default values as part of a structure

This section is where the variable structure of this system is given defaults for the model. This structure can then be changed for specific testing environments and send back through the model as an input structure.

```matlab
os.Ts = -1;
os.k1 = 2;
os.TYPE = 'Generator Set';
os.DESC = {};
end
```

The resulting structure can be seen above consisting of a primeMover, voltageSource, and exciter structure within the genset layer. Also the gain block parameters can be accessed (Ts & k1). We have added a section to each structure layer called TYPE and DESC. This is for further coding we might want to search an entire model for all components of TYPE, 'Generator Set'.

This code was used to produce one subsystem full of blocks, line connections, and other subsystems etc. The subsystem block above called Generator Set can be added to any model with one line of code.

```matlab
os.genset = generatorSetTutorial([modelName] '/Generator Set');,[xy]);
```

Published with MATLAB® 7.12

Fig. 13: Example on how to build MATLAB models using code (pg. 12)