NASA/TM-2020-5008125

Electrical Modeling and Thermal Analysis Toolbox (EMTAT) User's Guide

September 2020

Mark E. Bell, PE
HX5, LLC, Cleveland, Ohio

Jonathan S. Litt
Glenn Research Center, Cleveland, Ohio

National Aeronautics and
Space Administration
Glenn Research Center
Cleveland, Ohio 44135

Acknowledgments

The authors would like to thank Santino Bianco, Jeffryes Chapman, Rosana Maringolo-Baldraco, Peter Matt, George Thomas, and Edmond Wong of NASA Glenn Research Center for their work on EMTAT, testing, and for general technical advice. We would like to thank Thomas Carstens, Gregory Fretti, Francisco Mendez-Sosa, and Austin Lowmiller, through the NASA Faculty Fellows and Internship programs, for their work on EMTAT as well. We would also like to thank Kevin Melcher and Dennis Culley at NASA Glenn Research Center for their support in the development of EMTAT, and the NASA Transformative Aeronautics Concepts Program (TACP)/Transformational Tools and Technologies (TTT) projects for funding this work.

Table of Contents

Figure Index

Table Index

1 Introduction

Control design and analysis usually involve the use of models of the appropriate fidelity: high enough to capture the relevant dynamics but also able to execute in a reasonable time. With increased interest in Electrified Aircraft Propulsion (EAP), there is a growing need for an easily accessible analysis tool that combines rapid model development with faster-than-real-time execution in an industry standard environment. The modeling of EAP systems for control design and analysis involves a multidisciplinary approach accounting for the widely different dynamics of the turbomachinery, the power system, and the thermal interactions involved. The Electrical Modeling and Thermal Analysis Toolbox (EMTAT) is a set of Simulink® libraries designed to simulate a variety of power electronic devices, using both Physics Based and PowerFlow calculations. EMTAT is designed to interface with the Toolbox for the Modeling and Analysis of Thermodynamic Systems (T-MATS)[1] as a complementary set of library blocks, and as such it operates on a turbomachinery timescale. Due to the relatively slow time step of most T-MATS simulations, on the order of milliseconds, the electronic devices can be assumed to be operating at steady state over each time step. Steady state operation allows the electrical performance calculations to be simplified, with all of the high speed transients captured as efficiency losses. Note - this does not replace high speed, high fidelity electronic simulation tools such as SPICE. However, since EMTAT typically simulates 25x faster than real time on modern computers[*], it is more useful for testing control concepts. Steady state operation of electrical components, while smoothing out the high speed electrical transients, still demonstrates system dynamics in operation. Physics Based models also allow for realistic heat outputs and thermal rises to be calculated, with the associated performance impacts.

EMTAT contains electrical component models that can be dragged and dropped into the workspace and graphically connected to other blocks to form a model of a complete electrical system. EMTAT consists of two types of blocks: PowerFlow (or Load Flow) and Physics Based. PowerFlow is a steady-state method of modeling electrical systems. These models are based around lines linking different buses that are connected to generators or loads.[2] PowerFlow modeling provides a number of advantages: the overall system is modular (allowing for components to be added or removed easily), components may be represented by equivalent circuits that match required fidelity, and simulation execution time is much faster than real time (as noted above). Although these types of models come with many attractive attributes, they also assume the model is in steady state. While the Physics Based blocks are capable of modeling specific components due to physical performance parameters, and are in a sense higher fidelity than the PowerFlow blocks, they also operate in steady state from the point of view of the turbomachinery. This is because electrical transients are typically on the scale of microseconds (as opposed to milliseconds for turbomachinery), so the electrical system transient dynamics can be largely ignored since any perturbations will have fully settled out before the mechanical system has moved appreciably. Thus at each integration time step of the turbomachinery, the electrical system is assumed to be in steady state. With this assumption, the integration time step can be much larger than would be the case if the electrical dynamics were specifically modeled, which in turn increases simulation speed. The electrical dynamics are still accounted for as efficiency losses, which allow the heat generated by system operation to be modeled. This steady-state modeling approach is important for the system-level applications for which EMTAT has been designed because it enables faster-that-real-time execution in most cases. Thus it lends itself to simulating missions, hardware-in-the-loop testing, and dynamic analysis of system operation.

EMTAT was developed to work with MATLAB®/Simulink® software (The Mathworks, Inc.) and it was written using Version 2019b. It is intended for use by industry, government, and academia; for that reason, it is open source. All EMTAT equations were developed from public sources and all default maps and constants provided in the EMTAT software package are nonproprietary and available to the public. The software is released under the Apache V2.0 license agreement. At the time of this software release,

[*] The main workstation used for developing and testing this software had a 2.9 GHz quad core processor, 16 GB of RAM, and ran Windows 10 Enterprise.

license terms and conditions were found at http://www.apache.org/licenses/LICENSE-2.0.

2 EMTAT Setup

The EMTAT User's Guide applies to version 1.0 of the EMTAT software package, as developed by NASA Glenn Research Center. The package consists of the EMTAT Library that integrates with Simulink and contains the following sub-libraries: Physics Based Blocks, PowerFlow Blocks, and Solver Blocks. In addition, the EMTAT package contains examples to demonstrate how systems may be created using the Simulink libraries.

2.1 Notation

The notation summarized in Table 1 will be used throughout the EMTAT User's Guide.

Table 1 - EMTAT User's Guide Notation

Type	Definition
`Courier Font`	MATLAB function, Simulink blocks, or code (i.e., Simulink: `From` block)
Times New Roman Font Italics	Variables (i.e., $f(x)$) or file names, block input/output
Arial Font	File Paths (i.e., TMATS_Library\MEX) or hyperlinks
Platform: **Name** block	Signify what library a block is taken from (i.e., Simulink: `From` block)

2.2 System Overview

EMTAT is a generic set of electrical component models that are packaged as a library that can easily be used with MATLAB/Simulink to build simulation models. Developed to use graphical-based simulation techniques to meet the requirements of industry professionals as well as academics, EMTAT combines generic electrical and controls modeling capability with a numerical iterative solver to create a framework for the creation of complex electrical system simulations. Numerical methods utilized by EMTAT include Newton-Raphson iterative solving techniques along with a Jacobian calculation. EMTAT was designed to interface seamlessly with T-MATS, allowing for the creation of complex thermodynamic and electrical system simulations.

2.3 EMTAT Quick Start Guide

2.3.1 Installing EMTAT

To install EMTAT execute the following steps:

1. Download EMTAT from the GitHub server https://github.com/nasa/EMTAT, select the "Download ZIP" button, and extract the files to a folder that can be accessed by MATLAB, ensuring there are no spaces in the path name. Note that EMTAT was developed using MATLAB v2019b in the Microsoft Windows 10 operating system; it is not guaranteed to work on other operating systems or with other MATLAB versions.
2. Navigate to the directory of the desired released version of EMTAT in the downloaded zip file, *EMTAT_vXX.zip*,[†] in MATLAB.
3. New paths must be added manually to the *pathdef.m* file. Open the "Set Path" window by selecting on the "Set Path" button in the MATLAB toolbar on the HOME tab and add the path (\EMTAT) to the list. Select "Save" to save the path definitions; a message will pop up warning that administrator privileges are required and providing the option to instead save the paths to a new *pathdef.m* file. Save this file to a location on the MATLAB path. Note: this process will save the current path list; before saving, it should be verified that no unintended paths are on the list.

At this point the EMTAT Library manual installation is complete. EMTAT modules may be

[†] In this document, XX is used to represent a wild card character or characters, such as a version number or file name.

found in the EMTAT Library folders and specific components may be copied into your simulation as needed.

2.3.2 Un-installing EMTAT

To remove EMTAT, execute the following steps:

1. If paths were added manually to the *pathdef.m* file during installation, they must be removed manually by re-saving the *pathdef.m* file after deleting the \EMTAT path from the file.
2. Delete the EMTAT folder from your computer.

2.3.3 EMTAT examples

EMTAT includes a number of example simulation models to help a user gain insight into how to use the EMTAT Library to simulate electrical systems. To gain access to these examples, navigate to the Examples folder under EMTAT. To open an example, navigate directly to the folder containing the example and run the *XX_setup.m* file (or by opening the *XX.slx* file, in cases where a setup file does not exist). The *XX_setup.m* script will execute every command necessary to set up that particular example, which may include any of the following: loading variables to the workspace, creating paths, loading busses, and opening the Simulink model.

2.3.4 Block help

Each EMTAT Library block has a help file that explains how to set up each block; therefore, the low level functionality of each block will not be discussed in this document.

3 EMTAT Library Structure

The EMTAT Library contains three sub-libraries: PowerFlow, Physics Based, and Solvers.

- PowerFlow: This sub-library contains electrical blocks based on PowerFlow (load flow) analysis.
- Physics Based: This sub-library contains electrical blocks based on analysis of standard operation equations using physical component values.
- Solvers: This sub-library contains the numerical solvers required to simulate a Physics Based EMTAT system. These blocks are generic and can be used to create iterative solvers for a wide range of systems, even outside of EMTAT.

EMTAT is intended to be an environment for building electrical simulations in Simulink with the goal of dynamic analysis and controls design. The sub-libraries discussed above provide the basic building blocks of a simulation, however, a user may be required to create new blocks and/or modify existing blocks to meet their specific needs. The EMTAT Library has been developed specifically for that purpose.

4 EMTAT Simulation Creation

4.1 Simulation Setup

In many electrical systems, the inputs to each component alone are not enough to determine how the system will respond. For example, the duty cycle and input power to a voltage converter are not unique for a given output voltage, as it also relies on the output current which is determined by the rest of the circuit. In order to resolve this, an electrical power state equation is used to solve for the input current to the converter block. Calculating the power and the actual power are then used to calculate a power error. During simulation, the power error is driven to zero by adjusting the input current value to change the state at which the converter is operating (see Section 4.3 for more details), which is an iterative process.

The EMTAT software package contains blocks that allow for easy creation of an electrical model that requires “outer loop” iteration (over time) and “inner loop” iteration to minimize the error at each time step (Figure 1). The inner loop iterative solver blocks in the “Solver” sub-library apply a Newton-

Raphson method to iteratively solve for independent variables by monitoring dependent variables. For the voltage converter example above, the dependent variable is power error and the independent variable is input current. As with all Newton-Raphson techniques, a Jacobian is calculated by the solver; this Jacobian is essentially a linear map of partial derivatives from the independent variables to the dependent variables, which may change based on the system operating point and the degree of system nonlinearity. In cases where the Newton-Raphson solver fails to converge within a user-specified number of computations due to a non-linearity, the EMTAT solver blocks add robustness by recalculating the Jacobian matrix.

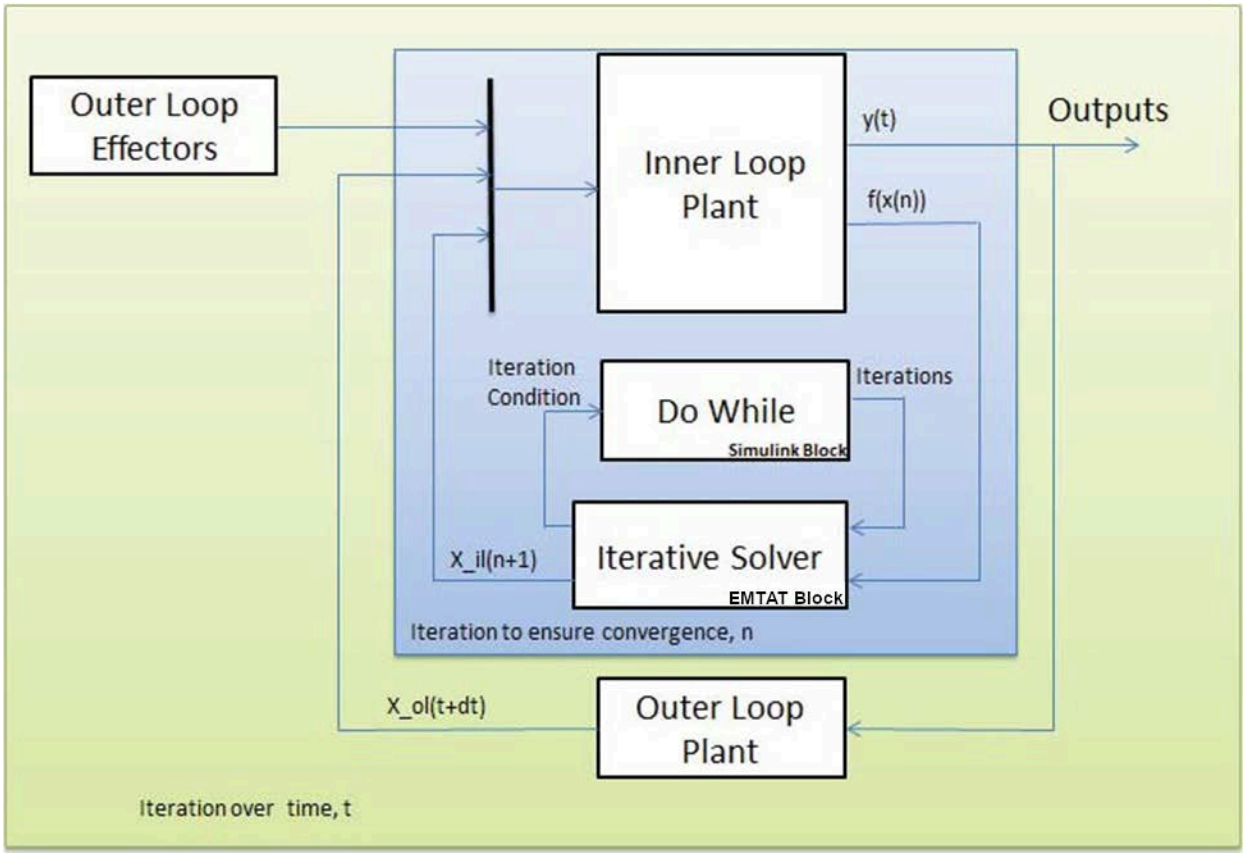

Figure 1 - Sample simulation architecture for a dynamic system.

4.2 EMTAT Formatting

4.2.1 Simulink wiring

EMTAT uses a vectored input/output signal architecture to limit inputs and outputs to those that are absolutely necessary. This means that many of the wires, which are the connections between modeling blocks, represent arrays of numbers that must be structured correctly for the simulation to work properly. The order of elements in these wires is given in the "help" page for each block; alternatively, the wire definitions may be viewed by looking under the mask or by connecting the wire into a bus selector. In general, if an output is not expected to be wired directly to another block, it will be placed in the *Outputs* bus.

4.2.2 Mask format

EMTAT makes extensive use of the subsystem mask tools in Simulink. Each EMTAT block comes with a mask that has been created to allow the user to provide to the block any required parameter or setting that must be defined. This facilitates the modeling process because the user is not required to have detailed knowledge about electrical systems in order use EMTAT. The mask parameters for each EMTAT block

may be viewed by opening that block; an example of an EMTAT block mask can be seen in Figure 2.

At the top of the input mask is the name of the library block followed by a block description. The variables that may be changed in the mask are listed in the parameters section, which may have multiple tabs. Each mask parameter input description is formatted as: "Name_M – Definition [units, if applicable] (matrix size, if applicable)," where the suffix "_M" denotes a variable defined in the mask. Check boxes are used in cases where features can be enabled or disabled; variable names for these items have the form "(VariableName)En_M". At the bottom of the parameter dialog is a "Help" button that will provide more information about the block when selected.

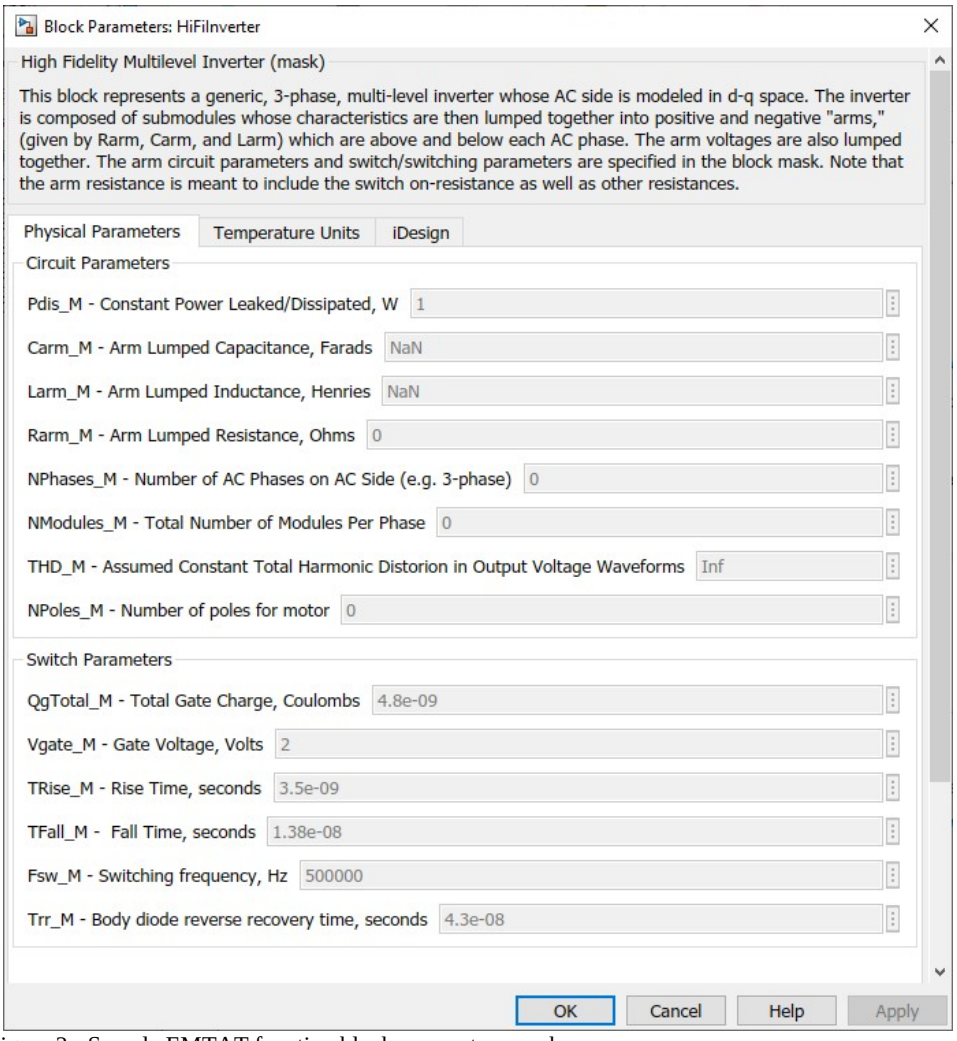

Figure 2 - Sample EMTAT function block parameters mask.

4.3 Solver Setup

The EMTAT solver block sets are based around two major components: an iterative solver and a Jacobian calculator. The iterative solver makes use of the Newton-Raphson method to step a plant toward solution and is described mathematically in Equation (1):

$$\begin{bmatrix} x_1 \\ \vdots \\ x_N \end{bmatrix}_{n+1} = \begin{bmatrix} x_1 \\ \vdots \\ x_N \end{bmatrix}_{n} - \boldsymbol{J}^{-}(x_n) \begin{bmatrix} f_1(\boldsymbol{x}_n) \\ \vdots \\ f_N(\boldsymbol{x}_n) \end{bmatrix}$$

Equation 1

Variables $x, n,$ and N are the independent variable, iteration number, and the number of independent and dependent variables. $\boldsymbol{x}$ is a vector of independent variables. $f(\boldsymbol{x}_n)$ is the dependent variable as a function of the independent variable. $\boldsymbol{J}^{-}(x_n)$ is the inverse of a Jacobian matrix as a function of the vector of independent variables. In this context, the Jacobian is a square matrix of partial derivatives that linearly map between plant inputs and outputs. It is defined by perturbing each plant input from an initial condition, $\boldsymbol{x}_0$,to find the effect on the plant outputs, $f(\boldsymbol{x_n})$. A more precise mathematical description of the Jacobian can be seen in Equation (2).

$$J = \begin{bmatrix} \frac{\partial f_1(\boldsymbol{x}_n)}{\partial x_1} & \cdots & \frac{\partial f_1(\boldsymbol{x}_n)}{\partial x_N} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_N(\boldsymbol{x}_n)}{\partial x_1} & \cdots & \frac{\partial f_N(\boldsymbol{x}_n)}{\partial x_N} \end{bmatrix}$$

Equation 2

EMTAT solver blocks work in two major steps. In the first step, the Jacobian calculator determines a linear map of the plant. Starting at the initial condition, the Jacobian calculator sequentially perturbs each input slightly, recording the results, and calculating the slope of each output as a function of each input. Once this is completed for each input, the Jacobian is built, inverted, and sent to the Newton-Raphson block. In the second step, the Newton-Raphson solver steps toward a solution using the inverse of the Jacobian developed in the first step. This solver method makes the assumption that the plant is locally linear, so it is possible that the solver will not converge to a solution when the system is (highly) nonlinear. The help files for the blocks inside the solver include more information about the solver and how it works. Additionally, if the solver cannot invert the matrix or the perturbations go outside the linear range, the simulation will present the user with an error message detailing the problem at run time.

4.4 EMTAT Block Setup

The “PowerFlow” and “Physics Based” sub-libraries of the EMTAT Library contain the generic blocks that may be combined to create electrical systems. A simple example of an electrical system, used throughout this document, is a Motor Branch, (a Voltage Converter, Voltage Inverter, and a Motor in series, as shown in Figure 3). Descriptions of the electrical system setup in this section assume the reader has some understanding of power electronics. In this figure, the EMTAT blocks are represented by the black bordered boxes, labeled appropriately. The forward outputs of each of the blocks are represented in blue, while the feedback outputs of each block are represented in red. The solver block can solve multiple simultaneous power balances at the same time, providing feedback to each block in turn.

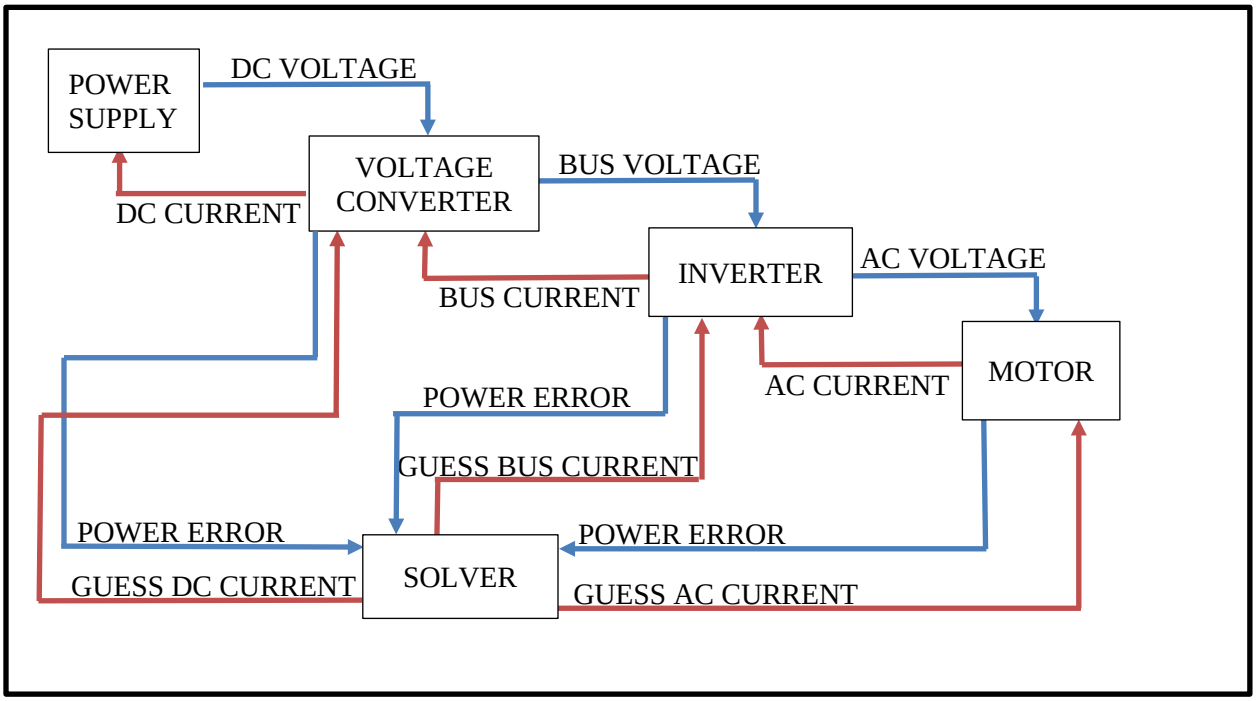

Figure 3 - Motor Branch diagram.

4.4.1 Independent and dependent variables

In a model of an electrical system, current from device to device is typically unknown and must be solved for using numerical methods. EMTAT solver blocks can be used to drive the power errors (the difference between input power and internally calculated power) to zero by changing the input electrical current of any given device. When setting up the simulation, the number of independent variables must match the number of dependent variables. Example variable definitions for the PowerFlow version of the Motor Branch can be seen in Table 2 and the Physics Based version in Table 3.

Table 2—PowerFlow motor branch solver independent and dependent variables

Dependent variables	**Independent variables**
Power Error (converter)	Current into converter
Power Error (inverter)	Current into inverter
Power Error (motor)	Voltage output

Table 3—Physics Based motor branch solver independent and dependent variables

Dependent variables	**Independent variables**
Power Error (converter)	Current into converter
Power Error (inverter)	Current into inverter
DQ Space Voltage Error (motor, 2 axis vector)	DQ Space Current into Motor (2 axis vector)

It should be noted that although certain inputs may be known in some simulations, the user should not set the known independent variables to constants and remove dependent variables from the solver. Minor changes in the vector of independent variables may drive convergence to a point that does not exist, which will result in the simulation crashing. It is more prudent to simply use the known value(s) in the initial conditions of the independent variables and allow the solver to use the dependent variables to solve for it. If the initial conditions are indeed the steady-state solutions, the final value of the independent variables will remain close to, or the same as, the initial values.

4.4.2 iDesign

The Physics Based library blocks rely on internal component level physical parameters to calculate the performance equations. Those values may not be intuitively understood by most users. iDesign is a feature of these blocks that calculates these internal component values based on device operation

equations, system level parameters, or extrapolation from existing reference components. This feature requires certain design or steady-state variables to be known and entered into their masks to work. Once this information has been defined, the iDesign calculation can be run and it will directly update the mask parameters for a given component. The specific required system-level parameters vary from block to block; the help menu for each of these blocks should be consulted to determine what is required. See Figure 4 for an example of the iDesign block compared to the variables input mask.

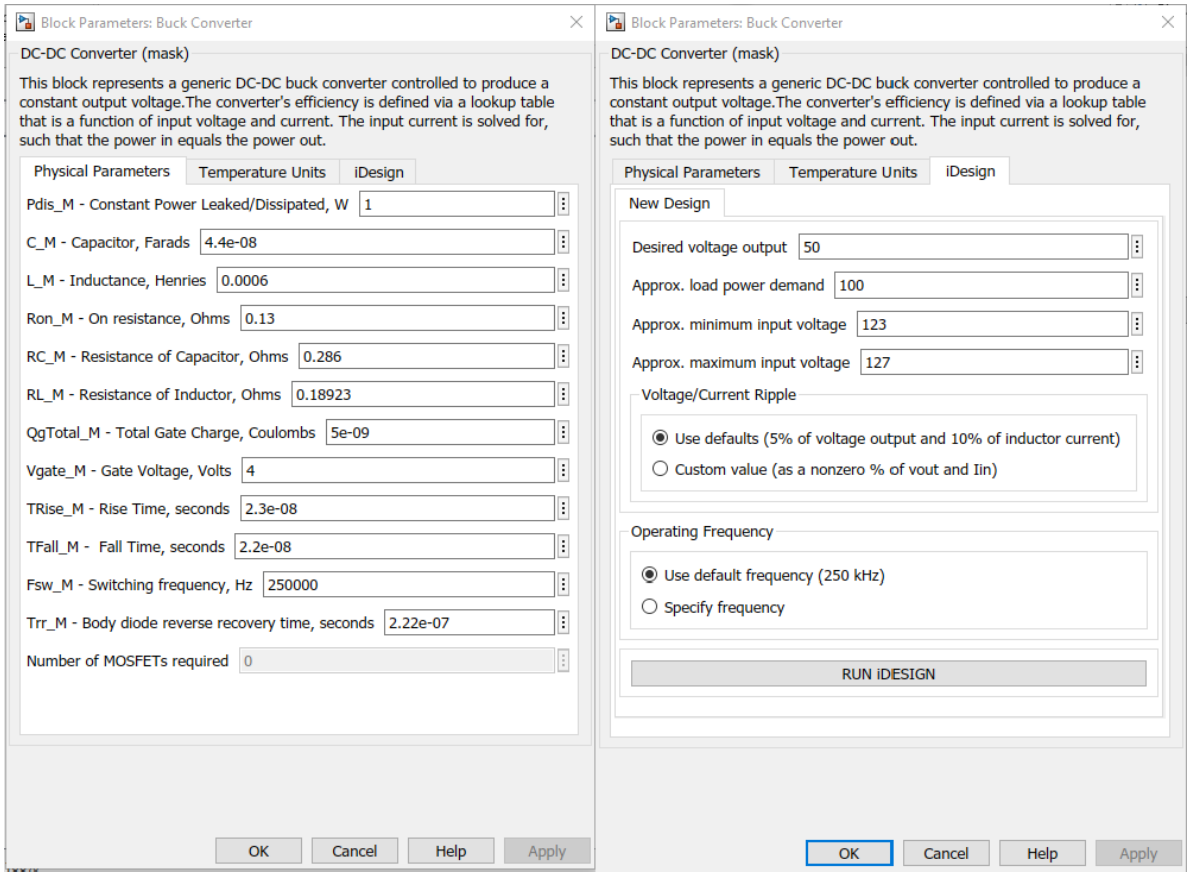

Figure 4 - Example EMTAT Physics Based Buck Converter input mask showing the Physical Parameters and iDesign tabs

4.5 Customization

Although EMTAT blocks are designed to be as general as possible, a user may need to customize an existing function within EMTAT or create an entirely new function to meet the needs of a specific application. The user may make modifications as they see fit and is encouraged to share any meaningful updates with the community at large. In order to make changes to an existing block without altering the original block, the user must break the link between the library and the block in question. This can be done by right clicking on the block in a model and selecting Library Link → Break Link.

5 Troubleshooting EMTAT

5.1 Convergence "Errors"

When running EMTAT blocks in a loop with solver blocks, the simulation will first act to converge the plant using the integrated Simulink solver. Failures in convergence often result in Simulink notification of errors in blocks that seemingly have nothing to do with the solver, such as integrators or state-space blocks, which makes troubleshooting the error more difficult. This is why, if an error occurs when running a model, the value of dependent variables ($f(x)$) routed to the solver should be checked first. If

any of the values are inf, NaN, or outside the specified iteration condition limits for the solver, the simulation is not converging. It should be noted that in some cases EMTAT blocks will issue errors. The reason for these errors will be clearly explained in the error message, and they will not be reviewed in this document.

The root cause of a convergence issue may not always be obvious. Outlined here are a few ways to troubleshoot a suspect simulation:

1. **Double-check block inputs:** Check for errors in the maps and constants used in the simulation.

2. **Check solver inputs:** Improper independent variable initial conditions, perturbation size, etc. may cause the solver to get stuck in a local minimum that does not result in system convergence.

3. **Check independent/dependent variable matching:** Verify that all system independent variables and dependent variables are accounted for in the simulation/solver.

4. **Component testing:** If data is available (from another example of the system) at the input and output of each component, the simulation can be broken down to the component-level, feeding the inputs observed from the other model to the inputs of each block. This can help isolate a specific block that is not modeled correctly.

5. **Steady-state testing:** If the system is intended to be run dynamically, try running it in steady-state first to identify if a problem with the solver parameters is contributing to the convergence problem.

6. **Run without solver:** Take out all the solvers and run the simulation with a constant input to each independent variable; if there is another example of the system that the model can be compared against, doing so can help find errors.

7. **Analyze the Jacobian:** It may be useful to look at the Jacobian (*J*), which shows the relationship between independent variables and dependent variables in the system, as the solver tries to converge to a solution. This value is located within the *S_Data* bus output from the solver blocks and can be displayed using the Simulink: Display block. Given an understanding of which independent variables should affect which dependent variables, the values of the Jacobian may be used to determine a possible source of error in the model.

8. **iDesign:** The iDesign tools use system variables to calculate realistic internal component values, with the caveat that these values may not match those of a given real system.

5.2 Crashing

There may be times when an EMTAT simulation will crash and issue a system error, causing MATLAB to close unexpectedly. These internal errors typically arise due to errors in the execution of compiled code (generally when the code attempts to access memory locations that it doesn't have access to). This can occur when:

- Attempting to access elements of arrays outside of the memory range (e.g., element 21 of a 15 element array).
- Using pointers incorrectly.
- Passing arrays to other functions by reference.

One such error has been observed when a length-1 vector or array is provided for a mask parameter used

for table lookup. If the simulation crashes, one of the first steps should be to check that the mask parameters for each EMTAT module have proper length. It should be noted this is just one possible reason for the crash, although a common one.

6 EMTAT Tutorials

The EMTAT software package is a set of blocks made in the MATLAB/Simulink environment that can be used to model any electrical system. In order to help the user better understand how the various blocks in the library can be assembled, the software package contains multiple examples. This tutorial will describe their development and highlight important points that the user should keep in mind when developing simulations. As a user works through the tutorial, it may be useful to have the corresponding example from the EMTAT Library loaded for comparison.

6.1 EMTAT Example: BoostConverter_Example

This section will describe how to use EMTAT blocks to create a simple boost converter system. This example is included with the EMTAT software download (*BoostConverter_Example.slx*). The example is a steady state simulation of the Physics Based Boost Converter block. A boost converter system is typically a stage in a larger electrical system simulation that provides a stable voltage reference for an electrical bus. In broad terms, it takes an input voltage from a power supply, boosts the voltage higher to a specified reference point, receives a current demand from the power sinks on the bus, and then calculates the current demand on the power supply. Here are the necessary steps taken to re-create this example:

Step 1. Create the plant

In this example, the plant has four main external inputs: input voltage, output current demand, duty cycle, and ambient temperature. The input voltage is from the power supply feeding the block. The output current demand is fed back from the blocks or bus downstream. The duty cycle is from a controller or solver. The ambient temperature is either fixed or fed back from a temperature block (although we are not using temperature feedback in this simulation). The plant has three main outputs: output voltage, input current demand, and power lost to heating. The output voltage would be fed into other components downstream as the controlled bus voltage. The input current demand would be fed back to the power supply feeding the boost converter. The power lost to heating would be fed through a temperature block to calculate the internal temperatures for performance adjustments.

The block requires an external integrator to drive the steady state error to zero, as well as to prevent an algebraic loop. The solver takes the power error from the boost converter block as a dependent input and calculates a guess input current to the block as an independent output. To simulate the effects of a controller, we also use the solver to compare the output voltage to a set point voltage, which will allow the solver to drive the duty cycle of the boost converter. In a typical system the user might use a PID controller to adjust the duty cycle, but for simplicity sake and to focus on the block itself, we used the solver instead.

Step 2. Select the solver

The system has only constant input voltage and output current demands, so it is not time-dependent and a steady-state solver block will be used. (TMATS: SS NR Solver w JacobianCalc, borrowed from the T-MATS library.)

Step 3. Connect the blocks

The outputs of the function blocks depend only on the inputs to the function blocks. The solver block has been designed to accept a vector input and produce a vector output, requiring that the outputs of the function blocks be multiplexed together and then wired to the input of the solver block, $f(x)$. The output of the solver, X, can be routed directly to the input of each function block, as those blocks accept a vector input (see Figure 6 later in this section, where displays were added to

enable the user to observe results).

Step 4. Set up the solver

Open the solver. The mask parameters need to be tuned per the application, which the iDesign tool can assist with. The length of the vectors *SJac_Per_M* and *SNR_IC_M* must each be equal to the number of independent variables or dependent variables in the system (which should be the equal; in this case, 2). Open on the Boost Converter block in the Simulink window to view the mask inputs for this particular case.

Step 5. Configure the simulation

To avoid encountering errors when running the simulation, the solver options should be set up prior to doing so through the "Configuration Parameters" window, accessed from the "Simulation" option on the Simulink menu bar. Navigate to the "Solver" pane and select the fixed-step, discrete solver as shown in Figure 5.

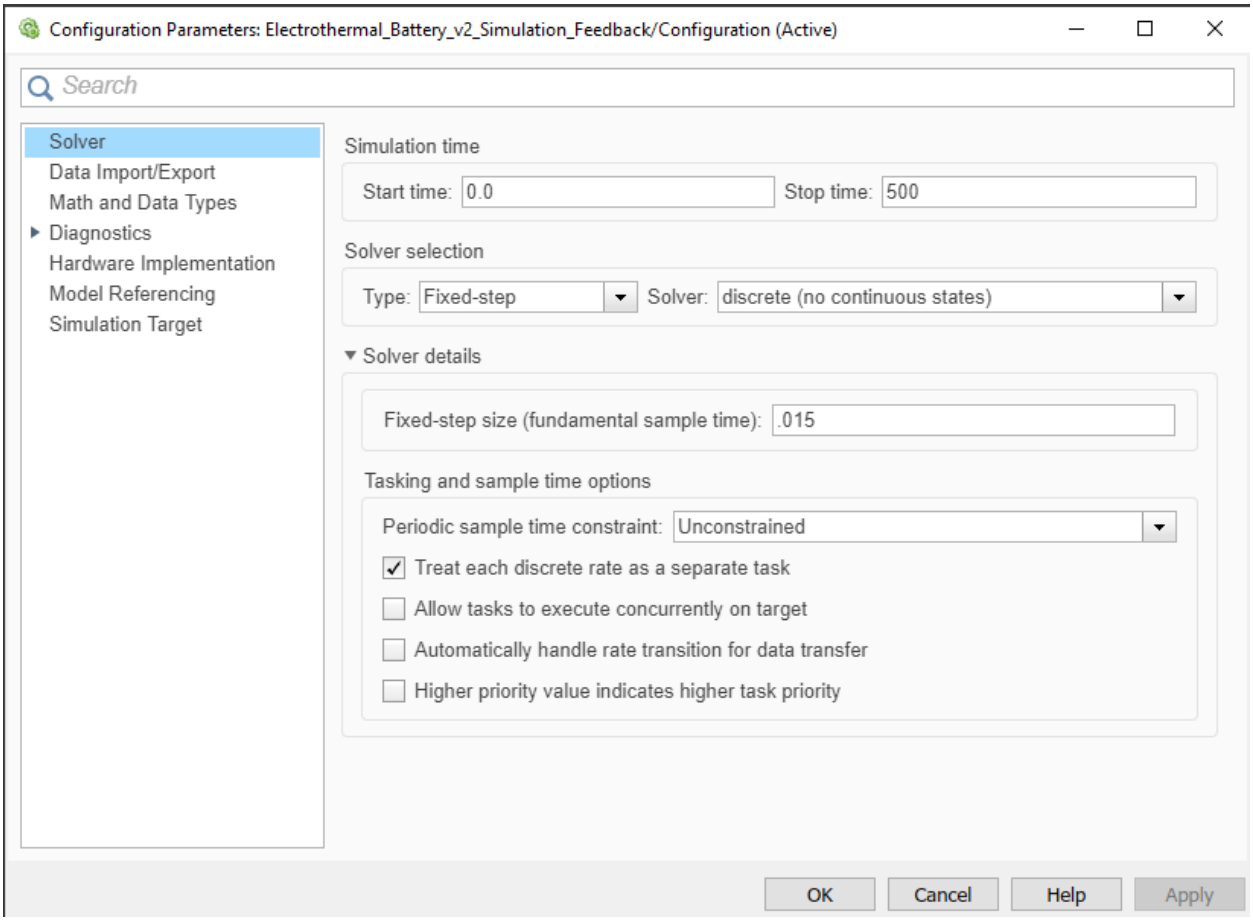

Figure 5 - Model configuration parameters setup.

Step 6. Run the simulation

At this point the system can be run. Select the green play button at the top of the Simulink model. If the simulation solver options have been properly set up, the simulation will run successfully.

Step 7. Execution is complete, interpret results

The displays will be populated with the solution to the equations (see Figure 6). The green tagged elements are fixed constants in this simulation, the purple tagged elements are the combined output bus. The orange tagged elements are internal calculations, the blue are outputs of the block, and the red are feedback inputs from the solver. The element *Converged* in the S_*Data* output vector shows whether the model has converged. This element will be 1 if the convergence criteria have been met but will be 0 if the solver inputs, *f(x)*, do not approach zero.

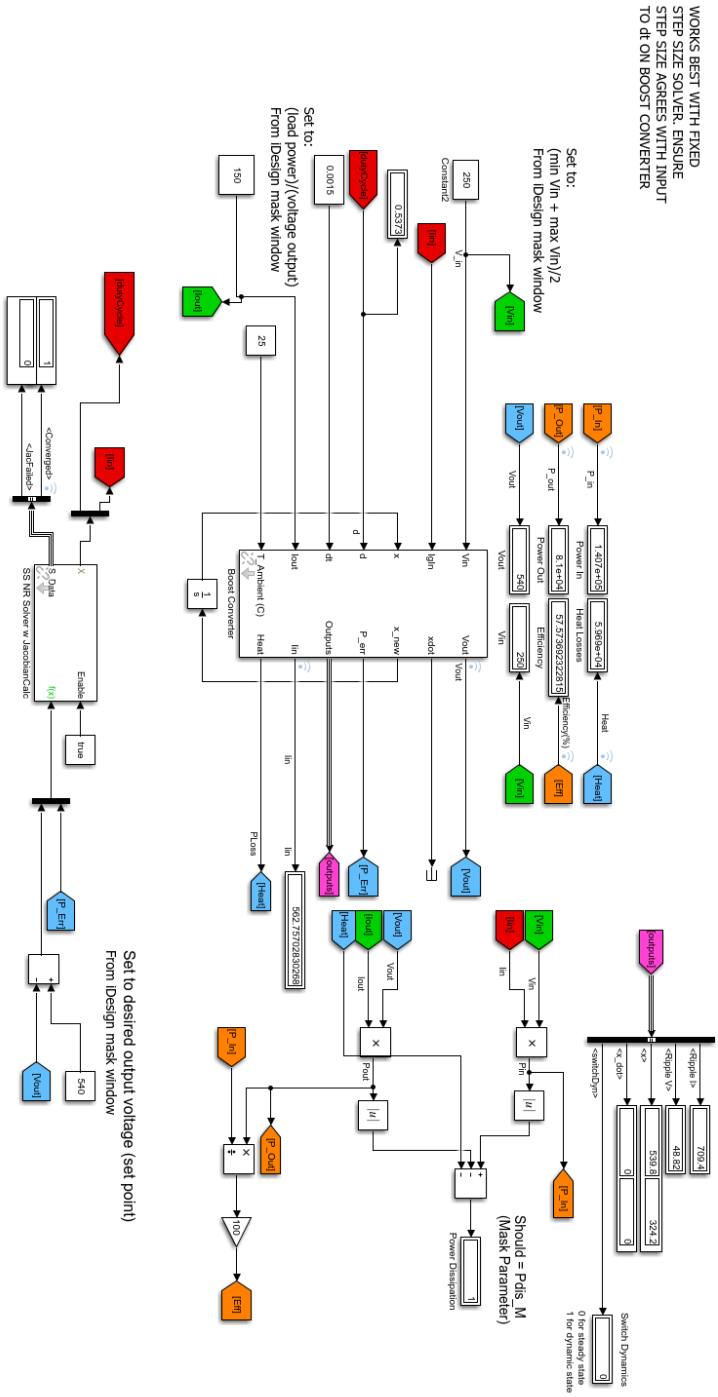

Figure 6 - Run Boost Converter Example.

[1]Chapman, J. W, et al, Toolbox for the Modeling and Analysis of Thermodynamic Systems (T-MATS) User's Guide, NASA/TM-2014-216638, 2014
[2]Gross, C.A., Power System Analysis, John Wiley & Sons, New York, NY, 1986