A Research Code for Dynamic Power System Simulation and Analysis

Steve Schaffer, Kevin Wedeward and Steven Ball

Abstract—This paper presents a versatile, Matlab-based simulation and analysis code for electric power systems. The code was created for basic research and designed from a control/dynamical systems perspective. Full advantage was taken of Matlab’s inherent cell arrays, structures, vectorized-calculations, ODE solvers with event handling and complex arithmetic. Design features that make the code easy to use and expand as well as several example analysis tools are outlined.

Keywords—power system simulation and analysis, Matlab

I. INTRODUCTION

Motivated by the desire to study power system behavior, researchers and educators have developed a variety of software applications that perform power system simulation and analysis. Inspiration for many of these applications can be found in the “closed” nature of commercial packages, broad availability of computational resources and popularity of high-level scientific programming languages such as Matlab, Matlab/Simulink, and Modelica. Matlab’s wide acceptance, graphing capabilities, toolboxes and vector-oriented approach have made it a common platform [1], [2], [3]. Simulink’s graphical interface to Matlab prompted its choice when a more intuitive interface was desired [4], [5], and the object-oriented modeling language Modelica was chosen to create a library and graphically-driven interface [6].

The Matlab-based simulation and analysis code summarized in this paper was created to study electric power systems from a dynamic system and control theory perspective. Design goals were to 1) have a versatile and evolving program accessible by researchers at the source code level to quickly and effectively add new model components, and implement and experiment with analysis tools and 2) provide numerical efficiency to allow simulation of moderately large systems on the order of hundreds of busses and thousands of states. This paper reports on the basic features and the various simulation and analysis tools that have been designed and implemented. Section 2 provides an overview of the code and its capabilities and section 3 summarizes example applications of analysis tools.

II. SIMULATION CODE OVERVIEW

To achieve the design goals outlined above, and to accommodate the diverse backgrounds and interests of a research team, the code was written in Matlab, and a large amount of effort was placed into building the core simulation code with standardized and facile data structures, data flow and functional components. The primary elements of the simulation code are shown in figure 1. A Microsoft Excel interface gives the user access to simulation options, model components and analysis tools. The simulation can be used for both static and dynamic simulation of electric power systems. In addition, a mechanism is available for discrete dynamics through which models of protection and logic devices can be incorporated.

The core simulation code is based on multi-machine models described in [7], static and dynamic load models, and the π-equivalent transmission line model [8]. A library of standard dynamic generator and load models are available in the code. Network topologies are quite general and permit any number of generators and/or loads of varying model types to be connected at a given bus.

The resulting model is described by a set of differential algebraic equations (DAE) of the form

\[ \dot{x} = f(x, y; \lambda) \]  

where \( x \) is the internal state vector (generators and loads), \( y \) is the algebraic state vector (voltage magnitude and phase) and \( \lambda, \mu \) represent the set of system parameters of interest. The vector field, \( f \), models the dynamics of the generators and loads. The algebraic equations (1b) represent the fast power balance dynamics on the sparsely connected transmission/distribution network of power lines and busses.

\[ 0 = h(x, y, \mu) \]  

Several Matlab features were employed to obtain numerical efficiency. Matlab’s vector arithmetic was exploited wherever possible and algebraic power balance equations (1b) were implemented in complex form to take advantage of Matlab’s efficient complex arithmetic. The built-in Matlab ODE solver, ode15s, was used to solve the system of equations making full use of its stability as a stiff ODE solver and accuracy control.

The discrete system of protection logic and exogenous system inputs were conveniently implemented using the ode15s sub-function and associated functions to detect and respond to events. Current protection logic capabilities that have been implemented are load sheds, thermal line trips,
voltage and frequency activated generator trips, load tap changing devices, and generator real power and exciter voltage saturation. Time delays associated with these events were implemented and new dynamic events can be added.

The code components and data structures were designed for ease of use and expandability. Microsoft Excel-based input/output gives the user easy access to model inputs, simulation options and also provides the ability to easily introduce new parameters, states and options. The data structures for states, parameters, outputs, internal pointers, and options are easy to use and are created, populated and calculated automatically in one core function. Passing data between functions as well as minimizing storage requirements has been greatly simplified by careful use of global variables. When a new model or state dependent model feature is added, the user makes appropriate additions to the functions associated with generator models, load models, Jacobian and initial conditions. An efficient implementation of the analytic system Jacobian has been implemented which significantly improved the efficiency of the solver, ode15s, and has been useful for several analysis tools. Initial conditions for the dynamic simulation are defined to be the equilibrium solution of (1) and are calculated by a standard loadflow algorithm and the dynamic states are then backed out from the loadflow solution. An algorithm using the dynamic simulation is used to obtain an equilibrium solution when saturated internal generator states occur. System outputs can be calculated, stored and plotted through the ode15s output sub-function.

Several key features were built into the code to facilitate research. A singularly perturbed (SPDE) version of the system equations (1) was implemented to provide the fully dynamic system

\[ \dot{x} = f(x, y; \lambda) \]
\[ \epsilon \dot{y} = h(x, y, \mu) \]

where \( \epsilon << 1 \) is a representation of the separation of fast and slow time scales. One choice of the time evolution of the voltage phasor components is to attribute real and reactive bus powers with bus voltage phase angle and magnitude, respectively. This type of approximation must be used carefully, but a perturbation of this type can be very useful for system analysis [9].

Simulation of a given model can be run in either the DAE or the SPDE mode and trajectory sensitivities of the system states can be calculated for any set of parameters [10]. Additionally, minimum and/or maximum saturation values can be set for any state, and a dynamic controller can be defined on any system state given any system input. For static analysis, a standard loadflow analysis can be performed as well as a simple bifurcation analysis for system parameters. Further descriptions of research applications that utilize the code are given in the next section.

### III. SAMPLE RESEARCH APPLICATIONS

To show the abilities and functionality of the simulation and analysis code, example applications are shown using the IEEE 118-Bus test system (http://www.ee.washington.edu/research/pstca/). Data files obtained from the website contain static network, generation, and load parameters. Typical dynamic parameters were used where required for dynamic generator and load models. Example applications demonstrated here are not meant to be all inclusive, but are primarily intended to establish the flexibility of the code.

#### A. System Linearization

It is often useful (to determine small-signal stability, for example) to linearize a complex nonlinear system. Linearization of the nonlinear system \( \dot{x} = f(x) \) around the operating point \( \left( x_0, y_0 \right) \) can be expressed in the form \( \dot{x} = Ax \), where \( A = \frac{\partial f}{\partial x}(x_0) \). The matrix \( A \) is also known as the Jacobian of \( f \). Making use of the programmed function \( ep\_jacobian \), the Jacobian of the nonlinear power system model at some \( (t, x) \) is obtained by typing \( A = ep\_jacobian(t, x) \) in the Matlab command window. This functionality allows the user to directly compute the stability of the linear system based on the eigenvalues of \( A \). Also, having access to the Jacobian allows the user to easily code a simulation of the linear representation of the system. Already having access to the inputs, states, and parameters, the user need only write a Matlab function to define and solve the linear system.

Linear feedback control in the form of \( u = -Kx \) and also linear observers in the form of \( \dot{x}_{obs} = Ax + Bu + H(y - Cx) \) have been implemented utilizing the code. Figure 2 shows an example of a linear observer where voltage magnitudes are estimated based on limited observation (\( \approx 50\% \)) of other bus voltage magnitudes.

![Fig. 2. Results for linear observer.](image)

#### B. Lie Brackets

Local accessibility of the system state with respect to a given input can be investigated through calculation of Lie brackets [11]. A numerical algorithm was implemented to approximate \( k^{th} \)-order Lie brackets for the system (1) based on the recursive application of analytic evaluation and finite difference approximations of directional derivatives of appropriate vector fields. The algorithm for approximating the \( k^{th} \) Lie bracket, \( G_k(x) \equiv [g_k, [g_{k-1}, \ldots, g_1]](x) \), at a point, \( x \), with respect the the vector fields \( g_j \) was easily
implemented using the available design tools in the code. The algorithm is based on the recursive definition of the $j^{th}$ Lie bracket

$$G_j(z) = [g_j, G_{j-1}](z) \cong d_{g_j}G_{j-1}(z) - d_{G_{j-1}}g_j(z)$$

where, suppressing the index $j$, we define

$$d_gG(z) = \begin{cases} \frac{1}{J_{G_j}(z)} g_j(z) & j > 2 \\ \frac{1}{J_{G_j}(z)} \left( G_j(z + \varepsilon g_j(z)) - G_j(z - \varepsilon g_j(z)) \right) & j = 2 \end{cases}$$

with $d_{G_{j-1}}g_j(z)$ defined similarly, and the intermediate vector fields, such as $G_j(z + \varepsilon g_j(z))$, defined recursively at the points $z \leftarrow z + \varepsilon g_j(z)$. Note that the analytic Jacobian matrices, $J_{g_j}$, are available from the Jacobian function.

C. Trajectory Sensitivities

As proposed by [10] there are many applications for trajectory sensitivities in electric power systems. Built into the code is the capability of calculating both steady-state and dynamic trajectory sensitivities. For the DAE model, the state sensitivities, $(x_\lambda, y_\lambda)$, with respect to a parameter, $\lambda$, in the internal dynamic equations are defined as the solution of

$$\dot{x}_\lambda = f_x(x, y) x_\lambda + f_y(x, y) y_\lambda + f_\lambda(x, y)$$

$$0 = g_x(x, y) x_\lambda + g_y(x, y) y_\lambda$$

(3a)

(3b)

where, $(x, y)$ is the solution of (1), and $f_x$, $f_y$, $g_x$, $g_y$ and $f_\lambda$ denote the appropriate Jacobian matrices. Consistent initial conditions for the sensitives are calculated after each system event. The sensitivities can be obtained from a singular perturbation of equation (3b) where the bus voltage magnitude and phase angle sensitivities are related as in (2). One unique way of visualizing steady-state sensitivities is the use of a three dimensional landscape in which the sensitivities of multiple state/parameter combinations exist on the same plot. Figure 3 shows an example of this, where the sensitivity of the bus voltage magnitudes $V$ to the generator set-point parameter $V_{ref}$ are plotted for every bus/generator combination on the same figure.

D. Feedback Linearization

Input-output feedback linearization as a methodology for controlling a limited number of states in models of electric power grids as described by [12] has also been implemented. Results shown by figure 4 are obtained using the complex power injection at bus 30 as an input and the bus voltage magnitude and phasor angle at bus 10 as the outputs to be controlled. The results indicate that the process of partial-state feedback linearization can be successfully implemented on realistic-sized power systems through the simulation and analysis code.

Fig. 3. Sensitivity landscape.

Fig. 4. Feedback linearization control of bus phase angle.

REFERENCES


