Article

Not peer-reviewed version

An Open-Source Julia Package for RMS Time-Domain Simulations of Power Systems

Thomas Philpott *, Ashish P. Agalgaonkar *, Thomas Brinsmead , Kashem M. Muttaqi

Posted Date: 29 September 2024

doi: 10.20944/preprints202409.2243.v1

Keywords: power systems; RMS simulation; Julia; open-Source

Article

An Open-Source Julia Package for RMS Time-Domain Simulations of Power Systems

Thomas Philpott [1,2,*], **Ashish P. Agalgaonkar** [1,*], **Thomas Brinsmead** [2] **and Kashem M. Muttaqi** [1],

[1] School of Electrical, Computer and Telecommunications Engineering, University of Wollongong, NSW, Australia
[2] Commonwealth Scientific and Industrial Research Organisation (CSIRO), Australia
* Correspondence: tsp266@uowmail.edu.au (T.P.), ashish@uow.edu.au (A.P.A.)

Abstract: This paper presents RMSPowerSims.jl, an open-source Julia package for time-domain simulation of power systems. The package is designed to be used in conjunction with PowerModels.jl, a widely used Julia package for power system optimization. RMSPowerSims.jl provides a framework for the simulation of power systems in the time domain, allowing for the study of transient stability, fault analysis, and other dynamic phenomena. The package is designed to be intuitive and flexible, allowing users to easily define custom models for network components and disturbances. The package has been verified against DIgSILENT PowerFactory and has produced results that are in close agreement with those obtained using PowerFactory. RMSPowerSims.jl is available under an open-source license and can be downloaded from the Julia package registry.

Keywords: power systems; RMS simulation; Julia; open-Source

1. Introduction

Power grids across the globe are presently undergoing a transition away from fossil-fuelled electricity generation towards renewable energy generation (REG). The variable nature of REG, the decentralization of generation, and the introduction of large numbers of inverter-based generators (IBGs) in the form of solar photovoltaic and wind generation mark significant differences from the traditional synchronous generator dominated power systems [1]. As a result, the development of new tools for power system analysis, operation, and planning is critical to maintain reliable and stable operation in modern power systems. For example, electromechanical RMS simulations, which have historically been used to analyze the dynamic behavior of the power grid, are unable to capture some of the fast dynamics of IBGs. As a result, the need for electromagnetic transient (EMT) simulations, which operate with much smaller time-steps, and associated grid models has become more prevalent [1]. This being said, RMS simulations remain applicable in some areas of power system stability analysis due to the computational expense of EMT simulations [2]. This is most prevalent in studies relating to frequency and angular stability, where the reduction of grid inertia resulting from an absence of synchronous is significant. Regarding scheduling and dispatch of generation, new optimal-power-flow (OPF) formulations are needed that can account for the stochastic nature of renewable generation, and the growing impact of distributed energy resources [3,4].

Commercial software packages, such as DIgSILENT PowerFactory [5] and PSS/E [6] are commonly used throughout industry for steady-state and dynamic analysis of power systems, and PSCAD/EMTDC [7] is commonly used for EMT simulation [8]. Open-source software tools are, however, becoming increasingly popular amongst academics and researchers, due to their ability to be modified and extended to suit the needs of the user, and the lack of prohibitively expensive licensing fees. In recent years, the Julia programming language has seen a range of power system related tools developed [9–13]. The reasoning for the selection of Julia for these developments is due to its high performance, comparable to that of low-level programming languages such as C and C++, while maintaining a high-level syntax that is easy for a user to understand. As a result, a suite of state-of-the-art solvers for differential equations, optimization, and linear algebra are available in Julia, making it an ideal language for developing power system analysis tools.

The collection of packages developed by the National Renewable Energy Laboratory (NREL) provide a modeling architecture for modern power grids [9], including a quasi-static simulation package as in [10] and dynamic simulation package suitable for both electromechanical and electromagnetic transient simulation as in [11]. The complexity of this ecosystem does, however, result in a steep learning curve for new users, especially those with little prior programming experience. PowerDynamics.jl provides an alternative option for users who wish to perform dynamic simulations of power systems, however, the implementation of the package models generators, loads and inverters as nodes in a graph, rather than components connected to a node. This is a departure from the traditional power system modeling approach, as is used in DIgSILENT PowerFactory [5] and PSS/E [6]. The approach is suitable for some research applications, and has been shown to be highly performative [12], but is restrictive in that it requires bespoke models if multiple components are connected to a single node. Additionally, for initialization purposes, it is practical to define a node as a slack bus, however, doing so requires that the dynamics of the generator at that slack bus must be omitted from a given model, which is unrealistic and not always acceptable.

The PowerModels.jl package provides a power system modeling framework, with the aim of facilitating easy comparison of OPF formulations, and is the only Julia package known to the authors capable of solving AC OPF problems. Substantial work has been done in implementing large, synthetic models of Australia's National Electricity Market in the PowerModels.jl format, and it would be beneficial to be able to build on these models for dynamic simulation, however this functionality is not supported by PowerModels.jl [14,15]. It is possible to convert from the PowerModels.jl network data dictionary (NDD) format to a format which can be used with software, either commercial or open-source, that is capable of dynamic simulation. In particular, the ability to import and export steady-state network models in MATPOWER's caseformat [16] is commonly supported by open-source power system simulation packages. However, in the experience of the authors, the process of implementing a power system in multiple software programs is time consuming, requires verification to ensure that the models are equivalent, requires maintaining two separate representations of the same system, and often requires an in-depth knowledge of both the source and destination program/format. It is, therefore, preferable to build on an existing model than reconstruct it in a different format. For this reason, this paper presents the development of a Julia based simulation package, known as RMSPowerSims.jl, that is capable of performing dynamic simulations using the existing PowerModels.jl model package from Julia.

The motivations for the development of RMSPowerSims.jl are two-fold. Firstly it provides RMS simulation capability with the aid of an existing PowerModels.jl model. Secondly, the scope of the energy transition requires a new generation of power system engineers to be trained in the use of power system analysis tools, and to be capable of addressing the research gaps. Currently, there is a gap in the freely available open-source tools for power system education when it comes to dynamic simulation. MATPOWER is commonly utilized in tertiary education programs for steady-state analysis and power flow studies, however, does not provide a toolbox with dynamic simulation capabilities. For this reason, RMSPowerSims.jl has been designed. Also, throughout the design process, we have aimed to make RMSPowerSims.jl as intuitive as possible, without making compromises to performance or accuracy.

The remainder of this paper is organized as follows: Section 2 discusses the overall structure of the new RMSPowerSims.jl package, Section 3 examines the process of running a simulation and provides a detailed overview implementation of the package, Section 4 presents a verification process of the package using the 39 Bus New England System [17] and DIgSILENTs PowerFactory, and Section 5 provides conclusions and outlines future work.

2. RMSPowerSims.jl

This section provides details relating to the modeling structures of RMSPowerSims. The high-level control flow, and the data structures used to define the network model are discussed in Section 2.1.

The hierarchical software object type structure used to define individual power system components is discussed in 2.2. Details of some specific component models that have been implemented are provided in Section 2.3, and disturbance simulation is discussed in Section 2.4. The package documentation [18] provides specific details relating to: the data model; the parameter definitions and underlying equations for the component models; disturbance implementation; function definitions; and the simulation process.

2.1. High-Level Package Structure

To provide an overview of the intended use of the package, the procedure of preparing a model and running a simulation is shown in Figure 1.

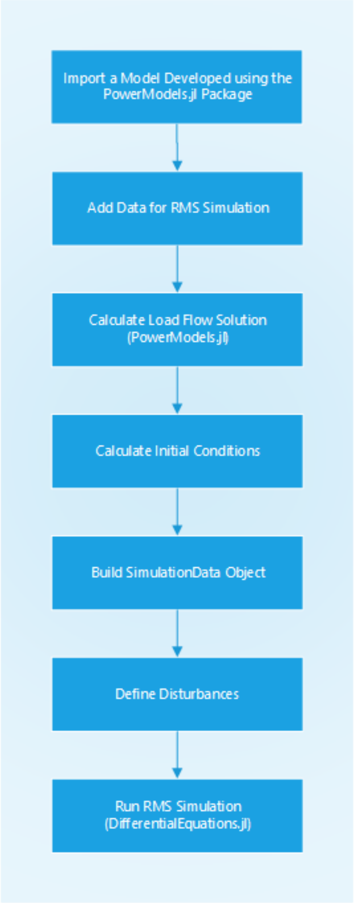

Import a Model Developed using the PowerModels.jl Package
Add Data for RMS Simulation
Calculate Load Flow Solution (PowerModels.jl)
Calculate Initial Conditions
Build SimulationData Object
Define Disturbances
Run RMS Simulation (DifferentialEquations.jl)

Figure 1. Intended procedure for running a simulation using RMSPowerSims.jl.

The primary data structure used for storing network data is an extension of the PowerModels.jl NDD format [13]. The NDD is an augmented version of the MATPOWER case file format [16], with the significant difference being the addition of dedicated fields for loads and shunt components. Any existing valid, Julia based PowerModels.jl model must contain all the necessary information for steady-state analysis. For time-domain simulation, additional parameters and modeling details are required that specify the dynamical behavior of the components of the network. RMSPowerSims.jl has been configured in such a way that RMSPowerSims.jl NDD is an extension of the PowerModels.jl NDD.

The Julia package DifferentialEquations.jl [19] is used to formulate the differential-algebraic equation (DAE) problem that characterizes the power-systems dynamics. Passing the entire PowerModels.jl NDD to the differential equation solver would be inefficient, as only a subset of the data contained in the dictionary is required to define the dynamic model of the system. Accordingly, a bespoke data structure is defined that contains only the data required for time-domain simulation, referred to as a *PowerSystemSimulation* object. At this point it is worth briefly outlining how a differential-algebraic equation problem is formulated in DifferentialEquations.jl, as a cursory understanding is necessary to explain the structure of the *PowerSystemSimulation* object and its components. A general DAE problem passed to the DifferentialEquations.jl package can be expressed as a function in the form

$$out = DAE\_function(u, du, p, t) \tag{1}$$

where u is a state vector containing the state and algebraic variables of the system, du is a vector containing the derivatives of the state variables, p is an object containing the non-time-varying parameters of the system, t is the time, and out is a vector containing the residuals of the differential and algebraic equations. The body of code in the $DAE\_function$ is where the equations that define an arbitrary dynamical system are implemented. Since a power system is an example of a DAE type system, it is possible to directly implement the equations that define the system in this form. However, due to the size and complexity of modern power systems, it is more practical to define the equations in a modular form, with each component of the system representing a functional module that has its own parameters and equations. These function modules can then all be called from a single main function that assembles the equations of the entire system. Algorithm 1 shows a pseudo-code representation of how this process is implemented in RMSPowerSims.jl, where u', du' and out' are the subsets of the u, du and out vectors that are necessary for implementation of the component model, and $model$ is a custom typed object that contains the parameters of the component model. This custom typing is important as it allows for the use of Julia's type-based multiple dispatch capabilities. Multiple dispatch in Julia allows for the definition of multiple methods for a given function, with the method that is called being determined by the types of the arguments passed to the function [20]. In the context of RMSPowerSims.jl, this allows the solver to identify which moduar $DAE\_function$ should be applied, depending of the type of the $model$ parameter. This type is defined within a hierachical typing structure, with the supertype defined as *ComponentModel*. The type hierachy and the existing subtypes defined are discussed further in Section 2.2.

Algorithm 1 Power System Equations

```
function power_system_equations!($out$, $du$, $u$, $p$, $t$)
    for each $component\_model$ in $power\_system\_model$ do
        $out'$ = DAE_Function($du'$, $u'$, $model$, $t$)
    end for
end function
```

With an understanding of how the equations of the system are implemented in the RMSPowerSims.jl package, it is now possible to discuss the structure of the *PowerSystemSimulation* object passed to the solver, and its components. A dictionary tree representation of the *PowerSystemSimulation* object is shown in Listing 1. For brevity, the diagram does not show the full structure, only those parts which are necessary for the explanation of the structure. The actual implementation can be found in the package documentation [18].

As can be seen in Listing 1, the *PowerSystemSimulation* object has a tiered structure and contains several other custom data objects. The top level of the structure contains a *PowerSystemModel* object, the initial conditions of the simulation, and a list of any disturbances that occur during the simulation. Disturbance handling and calculation of initial conditions are discussed in Sections 2.4 and 3, respectively. The design philosophy is that the *PowerSystemModel* object is intended to contain all of the data required to define the dynamic model of the system, independent of any the state of the system. In contrast, the *PowerSystemSimulation* object contains only those details related to a given simulation and corresponds specific state and trajectory of the system. The *PowerSystemModel* object is defined with two distinct lists. The variable list contains the unqiely defined names of all state and algebraic variables used to define the model, including the names of any derivative quantities required to implement the component models. The component list contains the *ComponentModel* objects and the pointers to the variables required for simulation of each component. There is a one to many relation between the component list and the variable list

```
PowerSystemSimulation
|-- PowerSystemModel
|    |-- Component List
|    |    |-- ComponentModel 1 & Pointers
|    |    |-- ComponentModel 2 & Pointers
|    |    |-- ...
|    |    `-- ComponentModel n & Pointers
|    `-- List of State & Algebraic Variables
|-- Initial Conditions
`-- Disturbances
     |-- Disturbance 1
     |-- Disturbance 2
     |-- ...
     `-- Disturbance n
```

Listing 1: Simulated dirtree within listing

2.2. Component Models and Typing

A hierarchical typing system is used to classify, and differentiate among, the various components of the model. The models for each component sit at the bottom of this hierarchy, and are implemented as what is referred to in the Julia programming language as a concrete type i.e., a type that can be instantiated. It is this lowest level of the hierarchy that is used to identify the correct method of a given RMSPowerSims.jl function to apply to a given component, using Julia's multiple-dispatch.

During the process of solving the differential equations, a system is fully defined by instances of concrete-typed component models and their corresponding *DAE_functions*. That is to say that all components in the system are treated in the same manner by the solver and hierachical type structure of the *PowerSystemModel* is irrelevant. The need for supertypes arises when configuring *PowerSystemSimulation* object and calculating the initial conditions. The hierarchy of supertypes defined in the package is shown in Figure 2. As can be seen in Figure 2, all component models are sub-typed from the *ComponentModel* type. The next level of subtypes classifies components as either a generator, controller, node, or load model. Conveniently, this classification aligns with the definitions already used in the PowerModels.jl NDD, with the exception of controller models, which are not included in PowerModels.jl. The reasoning for this classification, however, arises from the structure of the network equations. The power balance form of the network equations for bus i are expressed as

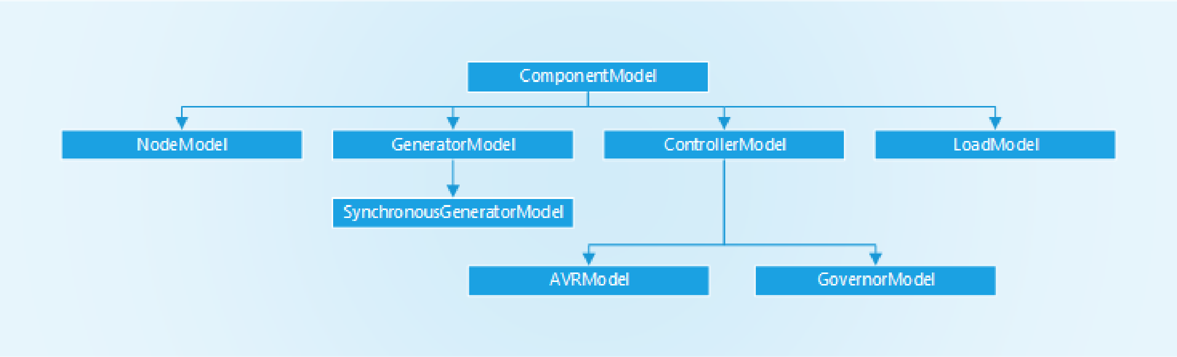


Figure 2. Structure of component model types in RMSPowerSims.jl.

$$0 = \sum_{k=1}^{n_g} P_{gk} - \sum_{k=1}^{n_l} P_{dk} - \sum_{k=1}^{n_b} |V_i||V_k||Y_{ik}|cos(\theta_i - \theta_k - \alpha_{ik}) \tag{2}$$

$$0 = \sum_{k=1}^{n_g} Q_{gk} - \sum_{k=1}^{n_l} Q_{dk} - \sum_{k=1}^{n_b} |V_i||V_k||Y_{ik}|sin(\theta_i - \theta_k - \alpha_{ik}) \tag{3}$$

where V is the voltage magnitude, θ is the phase angle, P_g and Q_g are the real and reactive power injections of the generators, P_d and Q_d are the real and reactive power injections of the loads, Y is the magnitude of the admittance between buses i and k, and α is the phase angle of the admittance between buses i and k, n_b is the number of buses, and n_g and n_l are, respectively, vectors containing the indices of the generators and loads connected to bus i. The interpretation of these equations is that the dynamics of the generators and loads in the system are coupled through the power injection terms in the network equations. The definition of the generator and load supertypes is motivated by the need to ensure that the power injection terms P_g, Q_g, P_d and Q_d are defined, otherwise this coupling would not be possible. As such, it is enforced by the RMSPowerSims.jl package that all generators must include variables P_g and Q_g, all loads must include variables P_d and Q_d, and that all power injection variables must be expressed on a common per-unit base. P_d and Q_d are defined as having positive values for positive demand to be consistent with the conventions used in PowerModels.jl. The definition of seperate supertypes for generators and loads allows for this difference in polarity, and also provides an intuitive distinction between the two types of components.

It is worth noting that it is also possible to couple the generators and loads through current injections, where the nodal algebraic equations represent real and reactive current, rather than real and reactive power [21]. This formulation is mathematically equivalent, and often viewed to be programattically simpler, however, the power-balance form has been chosen for as it is more intuitive and removes the need for additional equations to account for power injections. Extension of the package to include current injections, to allow alternative definitions of generators and loads, is in the scope of future work for the package.

The *ControllerModel* supertype is defined to allow for the inclusion of external generator controllers, such as automatic voltage regulators (AVRs) and governors for synchronous machines. A given instance of a *ControllerModel* object is implemented as a functional module that is distinct from the functional module of the generator it is connected to. It is during the process of configuring the *PowerSystemSimulation* object that the pointers are configured so that both the controller and the generator module point to the coupling variables. In the formulation of RMSPowerSims.jl presented in this paper, a *ControllerModel* can only be connected to a *GeneratorModel*, and not to a *LoadModel*. This is a logical restriction for a traditional power system, where generators are typically controllable while loads are not. However, considering the increasing prevalence of demand-side management, the authors note the utility of extending the package to allow for modeling of controllable loads, and as such is in the scope of future work.

The lower-level supertype *SynchrounousGeneratorModel* is defined as subtypes of *GeneratorModel*, and the lower-level supertypes *AVRModel* and *GovernorModel* are defined as subtypes of *ControllerModel*. This is to allow synchronous generators to be modeled either with or without AVR and turbine-governor models. The parameters that are controlled by the AVR and governor, excitation voltage and mechanical torque, respectively, are necessary for modeling of a synchronous generator, however it is common to model them as constant in a simplified model. To allow for this, during the process of configuring the *PowerSystemSimulation* object the package checks whether a controller of type *AVRModel* or *GovernorModel* is defined, and if not, a *ConstantExcitation* or *ConstantMechanicalPower* model is automatically added.

2.3. Implemented Component Models

Component models implemented for the IEEET1 type excitation system [22], the TGOV1 thermal governor [23] remain largely unchanged from their source material. The only differences are the omission of the over/underexcitation limiter inputs from the IEEET1, as those systems have yet not been implemented in the developed package, and the omission of the turbine damping branch in the TGOV1, as this is commonly neglected. Component models for constant excitation and constant mechanical power inputs to a synchronous machine model are also available, however, these are not discussed here as their implementation is trivial.

A static ZIP load model has been included, with the governing equations as below

$$P_d = P_{d0}(K_{pz}(\frac{V}{V_{nom}})^2 + K_{pi}(\frac{V}{V_{nom}}) + K_{qc}) \tag{4}$$

$$Q_d = Q_{d0}(K_{qz}(\frac{V}{V_{nom}})^2 + K_{qi}(\frac{V}{V_{nom}}) + K_{qc}) \tag{5}$$

Where V is the voltage magnitude of the connected bus, and P_{d0}, Q_{d0} and V_0 are the initial values of the active power injection, reactive power injection, and voltage magnitude of the connected bus, respectively. The initial values are parsed from the load flow solution. The parameters K_{pz}, K_{pi}, K_{pc}, K_{qi}, K_{qz}, and K_{qc} are the ZIP load parameters, and must satisfy the condition $K_{pz} + K_{pi} + K_{pc} = 1$ and $K_{qz} + K_{qi} + K_{qc} = 1$.

A sixth-order synchronous machine model based on the model presented in [21] is implemented. Some modifications are made to account for the dependence of electrical torque on rotor speed, as can be seen in the speed ratio terms in (11), (12) and (13). This dependence is necessary for simulations of small systems where rotor speed deviates significantly from synchronous speed during a disturbance [24]. If it is required that the effects of rotor speed variation be neglected, these ratios are simply replaced by one.

The differential equations of the model, as it is implemented in the package, are

$$\dot{E}_q T'_{do} = -E_q - (X_d - X'_d)\left(I_d - \left(\frac{X'_d - X''_d}{(X'_d - X_l)^2}\right)(\psi_{1d} + (X'_d - X_l)I_d - Eq)\right) + E_{fd} \tag{6}$$

$$\dot{E}_d T'_{qo} = -E_d + (X_q - X'_q)\left(I_q - \left(\frac{X'_q - X''_q}{(X'_q - X_l)^2}\right)(\psi_{2q} + (X'_q - X_l)I_q + E_d)\right) \tag{7}$$

$$\dot{\psi}_{1d} T''_{do} = -\psi_{1d} + E_q - (X'_d - X_l)I_d \tag{8}$$

$$\dot{\psi}_{2q} T''_{qo} = -\psi_{2q} - E_d - (X'_q - X_l)I_q \tag{9}$$

$$\dot{\delta} = 2\pi f_{\text{nom}}(\omega - \omega_{\text{ref}}) \tag{10}$$

$$\dot{\omega}\left(\frac{2H}{\omega_s}\right) = T_m - \left(\frac{\omega_s}{\omega}\right)\left(V_d I_d + V_q I_q + R_s\left(I_d^2 + I_q^2\right)\right) \tag{11}$$

The state variables E_q, E_d, ψ_{1d}, ψ_{2q}, δ, and ω represent the flux in the excitation, the flux in the three damper windings, the rotor angle, and the rotor speed, respectively. I_d, I_q, V_d and V_q are the direct and quadrature components of the stator current and voltage, respectively. E_{fd} and T_m are the excitation voltage and mechanical torque, respectively, and depend on the connected control system models. ω_{ref} is the reference speed of the system, which is typically the speed of the reference machine, although a center-of-inertia reference is also possible. All other terms in (6) through (11) represent static parameters of either the machine or the system. The relevant details can be found in [21].

doi:10.20944/preprints202409.2243.v1

The stator voltage equations are

$$0 = V\sin(\delta-\theta) + R_s I_d + \left(\frac{\omega}{\omega_s}\right)\left(-X''_q I_q - \left(\frac{X''_q - X_l}{X'_q - X_l}\right)E_d + \left(\frac{X'_q - X''_q}{X'_q - X_l}\right)\psi_{2q}\right) \tag{12}$$

$$0 = V\cos(\delta-\theta) + R_s I_q + \left(\frac{\omega}{\omega_s}\right)\left(X''_d I_d - \left(\frac{X''_d - X_l}{X'_d - X_l}\right)E_q - \left(\frac{X'_d - X''_d}{X'_d - X_l}\right)\psi_{1d}\right) \tag{13}$$

The generators power injections to the network are

$$P_g = I_d V\sin(\delta-\theta) + I_q V\cos(\delta-\theta) \tag{14}$$

$$Q_g = I_d V\cos(\delta-\theta) - I_q V\sin(\delta-\theta) \tag{15}$$

2.4. Disturbances

As with modeling of power system components, the modeling of disturbances utilizes custom typing to leverage Julia's multiple dispatch capabilities. Only a single supertype is defined in this instance, named *Disturbance*. The implementation of a disturbance in RMSPowerSims.jl consists of two parts: a uniquely typed disturbance object containing the parameters required to implement the disturbance, and a *perturb_model!* function that modifies the *PowerSystemModel* object.

For disturbances that do not cause significant discontinuities in the system variables, such as a small step change in load, the *perturb_model!* function can be implemented directly on the *PowerSystemModel* object used by the solver. Attempting this for disturbances that result in larger discontinuities, such as short-circuit faults, often results in the solver failing to converge. In these instances, the solver must be restarted from the time of the disturbance. The initial conditions for the new simulation are calculated from the state of the system prior to the disturbance using the following assumptions:

- The values of state variables do not change from the time of the disturbance to the time immediately following the disturbance.
- The values of algebraic variables are allowed to change instantaneously.
- The values of the derivatives of the state variables are allowed to change instantaneously.

In the current implementation of the developed package, this recalculation is performed as a two step process. Firstly, the values of the algebraic variables are recalculated using the updated *PowerSystemModel* object. This calculation is handled by one of Julia's non-linear equation solver packages [25], with the definition of equations once again being performed using type-based function overloading. A key difference between the *algebraic_equations!* functions called during this process and the *DAE_functions* used during the simulation is that the number of equations is reduced to only those required to define the algebraic variables. The second step is to recalculate the derivatives of the state variables. It is not necessary to use a non-linear solver for this process, as enough variables are defined at this stage for direct calculation of the derivatives for each component.

It is worth noting that the two-step process for recalculating the system state is not the most programmatically succinct method. An alternative that was considered is to re-use the equations defined in the *DAE_functions* and simply fix the values of the state variables, treating both the algebraic variables and the derivatives of the state variables as variable quantities passed to the solver. This allows for simultaneous calculation of the algebraic variables and the derivatives of the state variables and reduces the number of function definitions required for each component model. The reason for not selecting this approach is that it requires the solver to know which variables are state variables and which are algebraic variables. In most cases this is simple, and can be parsed from the component models data file, however, it becomes more complex for variables that can be treated as either, depending on the simulation context. The sole example of this that has arisen in development so far is the excitation voltage of a synchronous machine, which can change depending on the presence of an excitation system. This alternative method also increases the order of the system that must be passed to the non-linear solver, which in turn increases the time taken to solve. In any case, the solution of the alternative method is identical, however, the implementation of the alternative method is an open question that may be revisited in future development.

3. Simulation Process

This section will present a short example of how to run a simulation using RMSPowerSims.jl, in order to give an overview of what occurs during each function call. Listing 2 shows a code snippet that demonstrates how to run a simulation using RMSPowerSims.jl. The *load_model* function called in line 2 loads a PowerModels.jl NDD, which will be assumed in this example to already have been configured with the details needed for time-domain simulation.

```
 1  # Load network model
 2  load_model("example_network")
 3
 4  # Build PowerSystemSimulation object
 5  power_system_simulation = prepare_simulation(net)
 6
 7  # Set disturbances
 8  faulted_bus_index = 3
 9  (t_fault, t_clear) = (0.5, 0.55)
10  power_system_simulation.disturbances = Disturbance[
11      BusFault(faulted_bus_index, t_fault),
12      ClearBusFault(faulted_bus_index, t_clear, restart_simulation=true),
13  ]
14
15  # Run simulation
16  tspan = (0.0, 10.0)
17  soln = run_RMS_simulation(
18      power_system_simulation,
19      tspan;
20      solver_settings=Dict(
21          "reltol" => 1e-9,
22          "abstol" => 1e-8,
23          "maxiters" => 10000,
24          "dtmax" => 0.01,
25      )
26  )
27
28  # Add simulation results to network model
29  add_simulation_results!(net, soln)
30
31  # Plot results
32  plot_res(net, "gen", 1, "Pg")
```

Listing 2: Example of running a simulation using RMSPowerSims.jl.

The *prepare_simulation* function call builds the *PowerSystemSimulation* object from the PowerModels.jl NDD. It is at this point that the initial conditions for the simulation are calculated. PowerModels.jl is used to solve the load flow problem required as a starting point for the initialization process, however, in principle, other alternative solvers for the static powerflow solvers would be suitable. Following this the initial values of all state and algebraic variables are calculated by the functions defined for each *ComponentModel*. It is worth noting that the RMSPowerSims.jl function defined for calculation of initial conditions assumes a steady-state operating point, and as such is not suitable for starting simulations from a transient state. The *PowerSystemModel* is then generated by iterating over each power system component in the NDD.

Any disturbances that are to be applied during the simulation are added directly to the *PowerSystemSimulation* object, as is shown in lines 7-13 of Listing 2. The specific disturbances shown in the listing are a short-circuit fault at bus 3 of the network, and the clearance of the fault at the same bus. Note that the *restart_simulation* flag is set to true for the *ClearBusFault* disturbance. This is often necessary for clearance of faults as the solver has trouble converging to a solution from a state of near zero voltages.

The call of the *run_RMS_simulation* function executes the time-domain simulation. If any of the disturbances require a restart of the simulation, the simulation will be run in multiple stages, with the initial condition of each stage being calculated from the final state of the previous stage. To avoid duplicate time values, subsequent stages are started after a small time interval, t_{gap}, from the previous stage. For simulations in which multiple disturbances occur, this results in the simulated time interval between disturbances being smaller than the difference between the times they are set to occur by an amount of t_{gap}. It is important to minimize t_{gap}, as significant variations in simulation results have been observed if it is too large. An interval in the order of 10^{-5} has been found to provide acceptable results.

Finally, lines 28-32 of Listing 2 show some simple data processing. The function *add_simulation_results*! parses the solution objects returned by the differential equation solver and adds them to the PowerModels.jl NDD. This simplifies the process of accessing simulation results. The *plot_res* function is a simple plotting function that uses the Plots.jl [26] package to plot the results of the simulation.

4. Verification

In this section, the time-domain simulation results produced by RMSPowerSims.jl will be compared to those produced by DIgSILENT PowerFactory. A version of the PowerFactory model of the 39 Bus New England System [17,27], with modified AVR models and salient-pole generators replaced by round-rotor generators, is used for the simulation. The network diagram of the system is shown in Figure 3. The system contains ten synchronous generators, nine of which are equipped with a TGOV1 thermal governor and IEEET1 type AVR model. All generators are modeled as round rotor machines using PowerFactory's Standard Model [28].

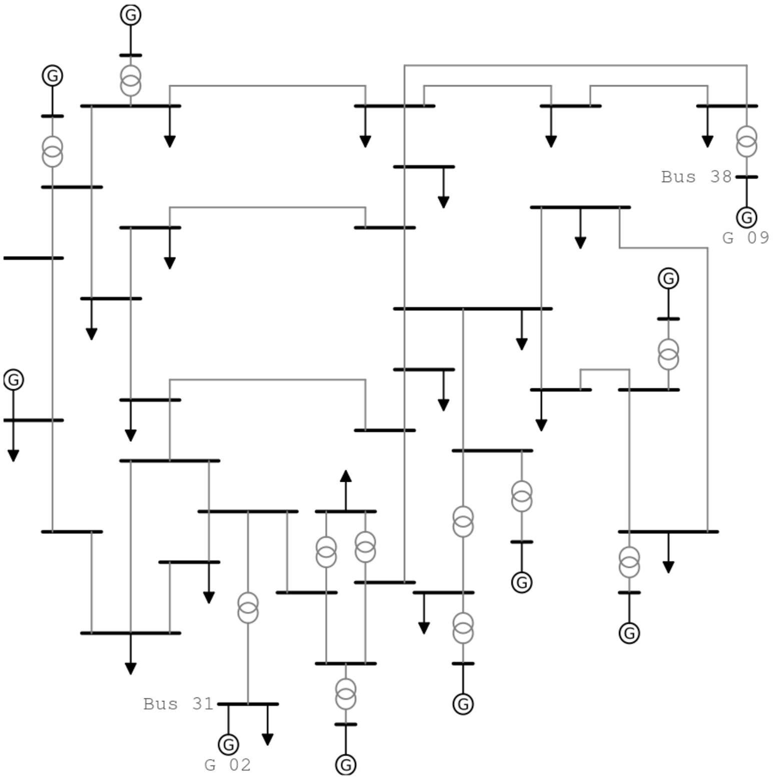
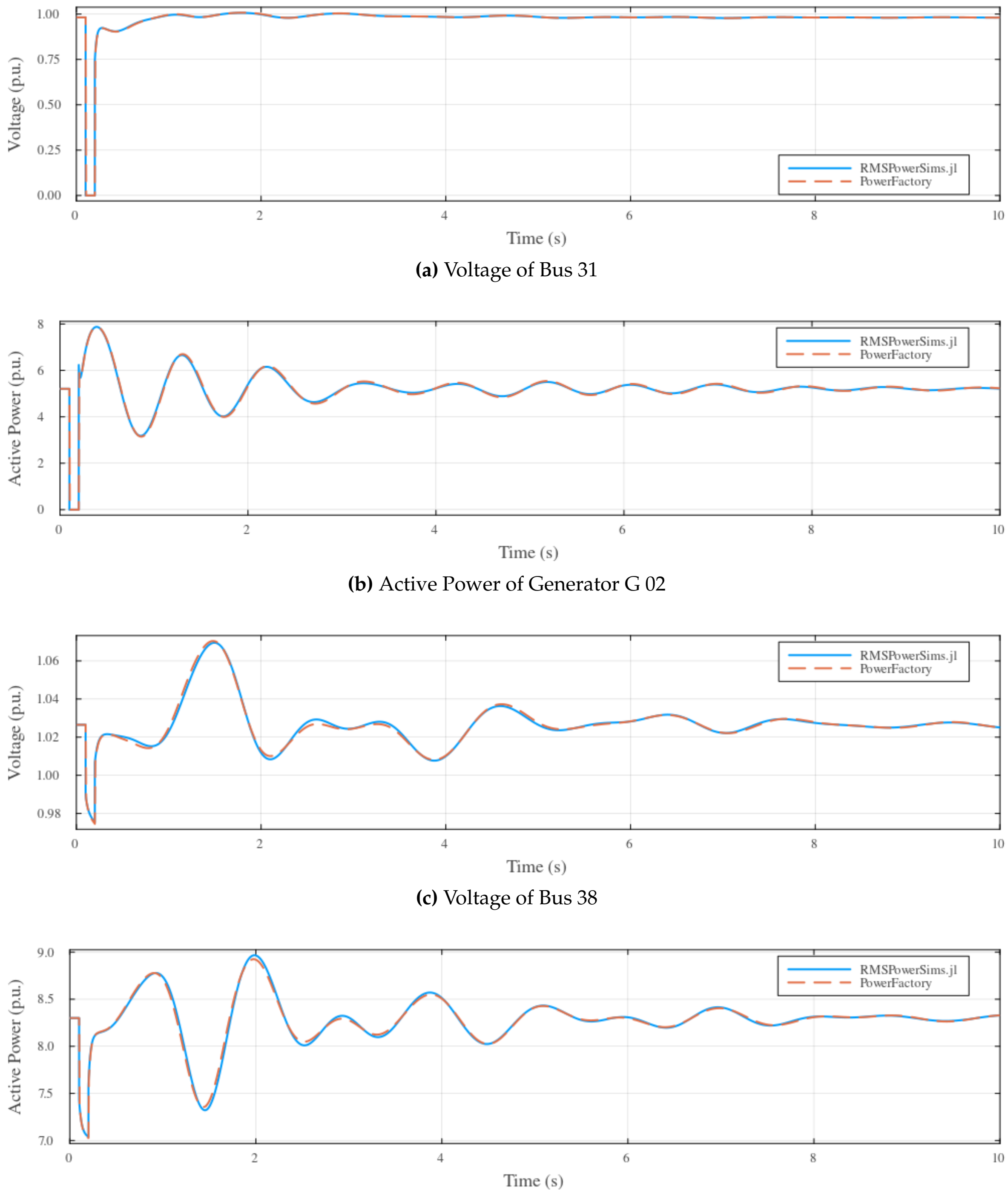


Figure 3. Network diagram of the 39 Bus New England System.

The results discussed in this section relate to a simulation in which a bolted short-circuit fault occurs at Bus 31 at $t = 0.1s$, and is cleared at $t = 0.2s$. RMSPowerSims.jl is set to recalculate the state of the system both at the time of the fault, and at the time of clearance. Figure 4 shows comparisons of the time series results produced by RMSPowerSims.jl and PowerFactory. As can be seen in Figure 4(a) and Figure 4(b), both the voltage at the faulted bus and the active power response of the generator connected to faulted bus, G 02, match the results produced by PowerFactory quite closely. Figures 4(c) and 4(d) show the voltage of Bus 9 and the active power response of the generator connected to Bus 38, G 09, which is distant from the fault. The results here are also shown to be in close agreement with those produced by PowerFactory.



(a) Voltage of Bus 31

(b) Active Power of Generator G 02

(c) Voltage of Bus 38

(d) Active Power of Generator G 09

Figure 4. Time series results for short circuit simulation.

It is worth noting that while the verification processes of other software packages commonly use metrics such as the root-mean-square error of the compared signals, we have simply opted for visual inspection in this case. The reasoning for this is that the synchronous generator models used in RMSPowerSims.jl and PowerFactory are not strictly identical [29], and as such, the results produced by the two packages are not expected to match identically. Any metric based comparison would, therefore, be misleading as it is not possible to attribute which differences are due to the generator model differences, and which are the implementation of the simulation package.

5. Conclusions

This paper has presented an RMS simulation capable Julia package, that is intended to be compatible with the PowerModels.jl ecosystem. The overall structure of the package has been discussed, as well as a brief overview of the simulation process. A verification process has been performed using the 39 Bus New England System to compare the responses acquired using the developed RMSPowerSims.jl pachage and DIgSILENT PowerFactory. The results clearly indicate that RMSPowerSims.jl is capable of producing accurate time-domain simulation results.

The immediate future work for the package includes the implementation of additional synchronous generator control models and renewable energy generation models. This will increase the range of systems that can be simulated and facilitate studies of modern power systems for all the researchers.

Author Contributions: Conceptualization, T.P.; methodology, T.P.; software, T.P.; validation, T.P.; formal analysis, T.P.; investigation, T.P.; resources, A.P.A, T.B. and K.M.M.; data curation, T.P.; writing—original draft preparation, T.P.; writing—review and editing, T.P., A.P.A, T.B. and K.M.M.; visualization, T.P.; supervision, A.P.A, T.B. and K.M.M.; project administration, T.P, A.P.A, T.B. and K.M.M.. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the Australian Government Research Training Program Scholarship and the Commonwealth Scientific and Industrial Research Organisation (CSIRO).

Data Availability Statement: The original data presented in the study are openly available in RMSPowerSims.jl at https://github.com/tphilpott2/RMSPowerSims.jl.

Conflicts of Interest: The authors declare no conflicts of interest.

Abbreviations

The following abbreviations are used in this manuscript:

AC	Alternating Current
AVR	Automatic Voltage Regulator
DAE	Differential-Algebraic Equation
EMT	Electromagnetic Transient
IBG	Inverter-Based Generation
NDD	Network Data Dictionary
OPF	Optimal Power Flow
REG	Renewable Energy Generation
RMS	Root Mean Square

References

1. Renewable Integration Study Stage 1. Report, Australian Energy Market Operator, 2020.
2. Hatziargyriou, N.; Milanovic, J.; Rahmann, C.; Ajjarapu, V.; Canizares, C.; Erlich, I.; Hill, D.; Hiskens, I.; Kamwa, I.; Pal, B.; Pourbeik, P.; Sanchez-Gasca, J.; Stankovic, A.; Van Cutsem, T.; Vittal, V.; Vournas, C. Definition and Classification of Power System Stability - Revisited & Extended. *IEEE transactions on power systems* **2021**, *36*, 3271–3281.
3. Capitanescu, F. Critical review of recent advances and further developments needed in AC optimal power flow. *Electric power systems research* **2016**, *136*, 57–68. doi:10.1016/j.epsr.2016.02.008.
4. Roald, L.; Andersson, G. Chance-Constrained AC Optimal Power Flow: Reformulations and Efficient Algorithms. *IEEE transactions on power systems* **2018**, *33*, 2906–2918. doi:10.1109/TPWRS.2017.2745410.
5. DIgSILENT PowerFactory Website. https://www.digsilent.de/en/powerfactory.html, accessed on 2024-08-18.
6. PSS/E Website. https://www.siemens.com/global/en/products/energy/grid-software/planning/pss-software/pss-e.html, accessed on 2024-08-18.
7. PSCAD PowerFactory Website. https://www.pscad.com/, accessed on 2024-08-18.
8. A Review of Power System Modelling Platforms and Capabilities. Report, The Institution of Engineering and Technology, 2015.

9. Lara, J.D.; Barrows, C.; Thom, D.; Krishnamurthy, D.; Callaway, D. PowerSystems.jl — A power system data management package for large scale modeling. *SoftwareX* **2021**, *15*, 100747. doi:10.1016/j.softx.2021.100747.
10. NREL-Sienna. PowerSimulations.jl. https://github.com/NREL-Sienna/PowerSimulations.jl, accessed on 2024-08-18.
11. Lara, J.D.; Henriquez-Auba, R.; Bossart, M.; Callaway, D.S.; Barrows, C. PowerSimulationsDynamics.jl – An Open Source Modeling Package for Modern Power Systems with Inverter-Based Resources. *arXiv (Cornell University)* **2024**. doi:10.48550/arxiv.2308.02921.
12. Plietzsch, A.; Kogler, R.; Auer, S.; Merino, J.; Gil-de Muro, A.; Liße, J.; Vogel, C.; Hellmann, F. PowerDynamics.jl– An experimentally validated open-source package for the dynamical analysis of power grids. *SoftwareX* **2022**, *17*, 100861.
13. Coffrin, C.; Russell, B.; Sundar, K.; Ng, Y.; Lubin, M. PowerModels.jl: An Open-Source Framework for Exploring Power Flow Formulations. *arXiv (Cornell University)* **2018**. doi:10.48550/arxiv.1711.01728.
14. Heidari, R.; Amos, M.; Geth, F. An Open Optimal Power Flow Model for the Australian National Electricity Market **2023**. doi:10.48550/arxiv.2306.08176.
15. Philpott, T.; Agalgaonkar, A.P.; Muttaqi, K.M.; Brinsmead, T.; Ergun, H. Development of High Renewable Penetration Test Cases for Dynamic Network Simulations using a Synthetic Model of South-East Australia. 2023 IEEE International Conference on Energy Technologies for Future Grids (ETFG), pp. 1–6. doi:10.1109/ETFG55873.2023.10408228.
16. Zimmerman, R.D.; Murillo-Sánchez, C.E.; Thomas, R.J. MATPOWER: Steady-State Operations, Planning, and Analysis Tools for Power Systems Research and Education. *IEEE transactions on power systems* **2011**, *26*, 12–19. doi:10.1109/TPWRS.2010.2051168.
17. Athay, T.; Podmore, R.; Virmani, S. A Practical Method for the Direct Analysis of Transient Stability. *IEEE transactions on power apparatus and systems* **1979**, *PAS-98*, 573–584. doi:10.1109/TPAS.1979.319407.
18. RMSPowerSims.jl. https://github.com/tphilpott2/RMSPowerSims.jl, accessed on 2024-08-18.
19. Christopher, R.; Qing, N. DifferentialEquations.jl – A Performant and Feature-Rich Ecosystem for Solving Differential Equations in Julia. *Journal of open research software* **2017**, *5*. doi:10.5334/jors.151.
20. Julia Documentation, Methods. https://docs.julialang.org/en/v1/manual/methods/, accessed on 2024-08-18.
21. Sauer, P.W.; Pai, M.A. *Power System Dynamics and Stability*; Prentice Hall, 1998.
22. Exciter IEEET1. https://www.powerworld.com/WebHelp/Content/TransientModels_HTML/Exciter%20IEEET1.htm, accessed on 2024-08-18.
23. Governor TGOV1 and TGOV1D. https://www.powerworld.com/WebHelp/Content/TransientModels_HTML/Governor%20TGOV1%20and%20TGOV1D.htm, accessed on 2024-08-18.
24. IEEE Standard 1110-2002 (Revision of IEEE Std 1110-1992) - IEEE Guide for Synchronous Generator Modeling Practices and Applications in Power System Stability Analyses, 2002.
25. NLsolve.jl. https://github.com/JuliaNLSolvers/NLsolve.jl, accessed on 2024-08-18.
26. Christ, S.; Schwabeneder, D.; Rackauckas, C.; Borregaard, M.K.; Breloff, T. Plots.jl – a user extendable plotting API for the julia programming language **2023**. doi:https://doi.org/10.5334/jors.431.
27. DIgSILENT GmbH. *39 Bus New England System, DIgSILENT PowerFactory*, 2020.
28. DIgSILENT GmbH. *PowerFactory 2022, Technical Reference, Synchronous Machine*, 2022.
29. Canay, I.M. Causes of Discrepancies on Calculation of Rotor Quantities and Exact Equivalent Diagrams of the Synchronous Machine. *IEEE transactions on power apparatus and systems* **1969**, *PAS-88*, 1114–1120. doi:10.1109/TPAS.1969.292512.

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.