Raiden. Power System Transient Stability Simulation Software Prototype

Eugene Mashalov, Joint Stock Company "Scientific and Technical Center of Unified Power System", Ekaterinburg, Russia

***Abstract*—This paper discusses implementation details of the transient stability analysis prototype software based on the implicit integration scheme. The main advantages of an implicit integration scheme with separate treatment of differential and algebraic state variables as well as an integration process with discrete event handling are considered. The results of test simulations and their comparison with existing software package are presented.**

***Index Terms*—Transient Stability Analysis, Implicit Integration, Discontinuous DAE**

I. Introduction

TRANSIENT stability simulation is one of the most complicated tasks of power system analysis. Besides methodological complexity, it requires intense computation with significant time consumption. Parallel implementation of simulation algorithms is difficult due to the sequential nature of the solution method, which is based on integration of the differential-algebraic system of equations (DAE), which is hardly separable into weakly connected parts. Thus, even modern computer systems do not provide the opportunity for real-time simulation, unless model details are reduced. Transient stability is dominantly used in offline applications, such as facility development, and rarely used for power system control on a time scale close to real-time.

However, some near-real-time implementations have been introduced in the last decade. One of the most successful is the stability monitoring system, used by the System Operator of Russia. The system evaluates the safe margins of network load over a period of 40–900 s with respect to static and transient stability. These margins are used for automatic or manual system control. The duration of the evaluation cycle is mostly determined by the duration of the transient stability simulation. The aim of reducing the computation time of simulation led to the development of the software prototype "Raiden" based on the implicit integration scheme, instead of the former implementation "RUSTab" based on explicit methods. The new algorithm consists of three components of integration: the method, the scheme, and the process.

II. Transient stability simulation algorithm

Time spent on transient simulation is proportional to the number of numerical integration steps of the system (1):

$$\begin{cases} \dot{y} &= f(x(t), y(t), t) \\ 0 &= g(x(t), t(t), t) \end{cases} \qquad (1)$$
$$x(t_0) = x_0, y(t_0) = y_0$$

where g and f are smooth vector functions and x_0, y_0 are initial conditions. The time of simulation initialization and computation of the initial conditions is much smaller than the time of the (1) solution. The latter depends on the integration method properties and the system dimension N_e. As a rule, solution time $T_{hn} = O(N_e^3)$, since the integration step in one form or another requires the solution of a system of nonlinear algebraic equations. The value of the integration step h_n is determined by the properties of the DAE. The crucial property of (1) with respect to h_n is the stiffness of the DAE, estimated as the ratio of the maximum and minimum values of the real part of the eigenvalues of its linearized matrix.

A. The integration method

In most cases, the transient stability problem (1) is considered stiff, which encourages using implicit integration methods for its solution. The integration method is a vector function $F(x, y)$ which maps values from the previous step y_{n-1} to current step values y_n. Of course, the current step values of $x_n \leftarrow x_{n-1}$ are also required, but there are several approaches to performing this. The general form of the implicit method for differential variables is:

$$F(y_n, y_{n-1}, t) = 0 \qquad (2)$$

The solution of (3) involves the solution of the nonlinear system of equations, which is associated with some technical complications. That was the main reason for the former implementation of transient stability software, which was developed in the 1980s relying on explicit methods. Integration step control bounds solution to:

$$\begin{gathered} ||z(t_n) - z_n(t_n)|| <= \epsilon, \\ z_n(t_n) = [y_n(t_n), x_n(t_n)]^T \\ z(t) = [y(t), x(t)]^T \end{gathered} \qquad (3)$$

where:

$z_n(t_n)$ is the approximated solution;
$z(t)$ is the exact solution;
ϵ is the error tolerance;

The integration step control varies $h_n = t_n - t_{n-1}$ according to (3). For transient phases with rapid changes, the step is reduced. When the transient decays, the step can be increased, which significantly reduces the number of steps required to solve (1).

The implicit method implementation is much more complicated than the explicit one, as it requires a non-linear system

of equations solution. In most cases, this solution is provided by the Newton method. Power system equipment models should be implemented with respect to this requirement and provide the ability to construct blocks of partial derivatives of the Jacobi matrix. The use of the automatic implementation system can overcome this complication [1]. Another problem that arises from nonlinearities is the possibility of multiple solutions at the time instants of discontinuities. However, the problems where (1) has discontinuous f and g are well known and there are various approaches developed. Some use rigorous discontinuity modeling [2], while others use integration restart techniques [3].

B. The integration scheme

The DAE (1) solution implies discretization into a pure algebraic system of equations using a chosen integration method. Since the resulting system is nonlinear, an iterative solution method is used. The simplest method is fixed point iteration. This method is usable, but, in many cases, does not provide convergence due to its limitations. The application of the Newton method requires Jacobian matrix formation, which greatly increases the equipment models development complexity. In addition to the system of equations for each model, it is also required to form a block of the Jacobi matrix. However, Newton's method makes it possible to reliably obtain a solution, has good convergence, and, moreover, can be used not only to solve (1) but also to determine the initial conditions at $t = 0$ and at the time points of discrete changes t_d. The weak side of Newton's method is its sensitivity to the quality of the initial guess. This problem is partly solved by using the initial estimation from the previous integration step. An additional factor that improves the convergence of Newton's method can be the scheme of the integration method. A well-known integration approach is the predictor-corrector scheme. With values of the available solution z_{n-1} at the time t_{n-1}, the method can predict the values of z_n using extrapolation. The resulting prediction is used as an initial estimation for solving the equivalent (1) system, in which the differential equations are discretized in a form that depends on the chosen integration method. During the solution, the prediction is corrected so that the values satisfy the given system of equations. This scheme is effectively implemented using multi-step integration methods, which imply saving the data of several completed steps to perform the next step. To solve DAE, this approach is implemented in the Gear's method [4]. This method involves the use of the BDF integration method, both for differential and algebraic equations, and provides a simultaneous solution to (1). Such an integration scheme is fully implicit and preserves integration method stability properties.

A variant of the Gear's method is implemented in the well-known software EUROSTAG [5]. Since the methods of the BDF family have the property of damping high-frequency oscillations with an increase in the integration step, the Gear's method exhibits "hyperstability" and erroneously treats unstable cases as stable. The proposed modification, which consists of applying the Adams method to differential variables, eliminates this drawback since the 2nd order Adams method is A-stable. The use of the BDF for algebraic variables makes it possible to maintain a higher integration step compared to the Adams method with the same overall stability properties of the integration scheme.

All linear multistep methods of order q, despite significant differences in their properties, can be expressed in Nordsieck form [6]. That vector has a dimension of $1 \times q + 1$. Since the integration method must be A-stable order is constrained $(q \leq 2)$ and the maximum dimension of the Nordsieck vector is fixed at 1×3.

At the n-th integration step, Nordsieck vector components of differential and algebraic variables are:

$$Y_n = \left[y_n, h_n\dot{y}_n, \tfrac{h_n^2}{2}\ddot{y}\right], \quad X_n = \left[x_n, h_n\dot{x}_n, \tfrac{h_n^2}{2}\ddot{x}\right] \tag{4}$$

The Nordsieck vector is merely a form to express the Taylor's expansion of x_n and y_n at t_n. If this vector for t_{n-1} is known, the prediction at t_n is:

$$Y_n^0 = Y_{n-1}A, \quad X_n^0 = X_{n-1}A \tag{5}$$

where:

Y_n^0 is the differential variables vector extrapolated to t_n;
X_n^0 is the algebraic variables vector extrapolated to t_n;
A is the lower triangular Pascal matrix $q+1 \times q+1$.

The corrected Nordsieck vector at t_n is:

$$Y_n = Y_n^0 + l^Y e_{yn}, \quad X_n = X_n^0 + l^X e_{xn}, \tag{6}$$

where:

l^Y is the row vector of integration method for differential variables;
l^X is the row vector of integration method for algebraic variables;
e_{yn}, e_{xn} are the error vectors with respect to the predicted vectors;

The vectors e_{yn}, e_{xn} are normalized:

$$l_1^Y = l_1^X = 1 \tag{7}$$

The corrector equations (6) can be written by components:

$$\begin{array}{rcccl} y_n & = & y_n^0 & + & l_0^Y e_{yn} \\ h_n\dot{y}_n & = & h_n\dot{y}_n^0 & + & l_1^Y e_{yn} \\ \frac{h_n^2}{2}\ddot{y}_n & = & \frac{h_n^2}{2}\ddot{y}_n^0 & + & l_2^Y e_{yn} \\ x_n & = & x_n^0 & + & l_0^X e_{xn} \\ h_n\dot{x}_n & = & h_n\dot{x}_n^0 & + & l_1^X e_{xn} \\ \frac{h_n^2}{2}\ddot{x}_n & = & \frac{h_n^2}{2}\ddot{x}_n^0 & + & l_2^X e_{xn} \end{array} \tag{8}$$

From the $\dot{y}_n$ equation from (8):

$$\begin{array}{l} h_n\dot{y}_n = h_n f(x_n, y_n, t_n) \\ h_n\dot{y}_n^0 + l_1^Y e_{yn} - h_n f(x_n, y_n, t_n) = 0 \end{array} \tag{9}$$

Using (7) define:

$$\begin{array}{c} F_Y(e_{xn}, e_{yn}, t) = h_n\dot{y}_n^0 + e_{yn} - h_n f(x_n, y_n, t_n) \\ F_X(e_{xn}, e_{yn}, t) = g(x_n, y_n^0 + l_0^Y e_{yn}, t_n) \end{array} \tag{10}$$

from which the following system can be obtained:

$$\begin{cases} F_Y(e_{xn}, e_{yn}, t) &= 0 \\ F_X(e_{xn}, e_{yn}, t) &= 0 \end{cases} \tag{11}$$

with Jacobi matrix (n step index is omitted for simplicity):

$$J = \begin{bmatrix} \frac{\partial F_Y(e_x,e_y,t)}{\partial e_y} + \frac{\partial F_Y(e_x,e_y,t)}{\partial e_x} \\ \frac{\partial F_X(e_x,e_y,t)}{\partial e_y} + \frac{\partial F_X(e_x,e_y,t)}{\partial e_x} \end{bmatrix} \tag{12}$$

The combination of (10) and (12) gives:

$$J = \begin{bmatrix} I - h_n l_0^Y \frac{\partial f(x,y,t)}{\partial y} - h_n l_0^X \frac{\partial f(x,y,t)}{\partial x} \\ l_0^Y \frac{\partial g(x,y,t)}{\partial y} + l_0^X \frac{\partial g(x,y,t)}{\partial x} \end{bmatrix} \tag{13}$$

Note that the Jacobi matrix does not become singular with $h_n \to 0$, which allows it to be used to solve DAE for initial conditions y_d, x_d at instants of discontinuities t_d. In addition, step reduction does not cause numerical problems, which, for example, the integration scheme [7] is subject to.

The system of equations (11) can be solved iteratively. Variables at iteration m are:

$$\begin{cases} y_n^m = y_n^0 + l_0^Y e_{yn}^m \\ x_n^m = x_n^0 + l_0^X e_{xn}^m \\ e_{xn}^0 = e_{yn}^0 = 0 \end{cases} \tag{14}$$

The iterative process of solving (11) with respect to e_x and e_y is performed according to the recursive formula (n step index is omitted for simplicity):

$$\begin{bmatrix} e_y^{m+1} \\ e_x^{m+1} \end{bmatrix} + \begin{bmatrix} e_y^m \\ e_x^m \end{bmatrix} + (J^m)^{-1} + B^m \tag{15}$$

where:

$$B^m = \begin{bmatrix} h_n f(x^m, y^m, t) - h\dot{y}^0 - e_y^m \\ -g(x^m, y^m, t_n) \end{bmatrix} \tag{16}$$

and:

$$J^m = \begin{bmatrix} I - h_n l_0^Y \frac{\partial f(x^m,y^m,t)}{\partial y} - h_n l_0^X \frac{\partial f(x^m,y^m,t)}{\partial x} \\ l_0^Y \frac{\partial g(x^m,y^m,t)}{\partial y} + l_0^X \frac{\partial g(x^m,y^m,t)}{\partial x} \end{bmatrix} \tag{17}$$

When the process (15) converges, the Nordsiek vectors X_n and Y_n at the t_n are updated using (6).

C. Local truncation error control

The solution of (15) is the difference between predicted and corrected values of the state variables. It is used for local truncation error estimation. The next integration step and integration method optimal order can be further determined using error estimation. Local truncation error d_n at the step n:

$$d_n = C_{q+1} q! l_q e_n \tag{18}$$

where C_{q+1} is the integration method constant.

Since state variables may have very different magnitudes, it is common to use weighted differences for error estimation:

$$W_{in} = Rtol_i |Z_{i,n-1}| + Atol_i \tag{19}$$

where:

$Atol_i$ is the absolute tolerance of i-th state variable;
$Rtol_i$ is the relative tolerance of i-th state variable.

The condition (20) is used to accept step n.

$$||d_n|| = \sqrt{\frac{1}{N} \sum_{i=1}^{N} \left(\frac{d_{i,n}}{W_{i,n}} \right)^2} \leq 1 \tag{20}$$

where N is the number of state variables. If (20) holds, the step and the order of the integration method are adjusted for the next step. The step is rejected and repeated otherwise with a smaller h.

The choice of the norm type for d estimation has a strong influence on the quality of the simulation. The default option is l^2-norm, which is acceptable both for the quality of the solution and step size control. As the (1) is stiff, differential equations with small time constants tend to produce large differences between predicted and corrected values. The l^2-norm averages these differences and allows step size to be larger. However, this averaging property degrades the quality of simulation for models with large dimensions due to the "masking" of unacceptable errors in some state variables. In some cases, these errors have a crucial influence on the stability simulations and may lead to erroneous estimation of unstable cases as stable. Using l^∞-norm overcomes this problem but leads to an unacceptable time step decrease. A hybrid norm was proposed in [8], which combines properties of l^2- and l^∞-norms. In this paper, a similar approach based on the weighting of norms is applied.

The operations of integration step and order change induce extra computation costs and should not be used at every step. Step size and order control are based on the strategy described in [9], where these aspects are considered in relation to the pure differential system of equations. Since the problem being solved is differential-algebraic and stiff, additional measures for convergence control of the corrector and excessive step change suppression are implemented on the basis of [7].

D. Adams integration method ringing suppression

The implicit 2nd-order Adams method, also known as the trapezoidal rule, has the property of "ringing" with the increase of the integration step. Ringing appears as oscillations of state variables at transient decay phases where the integration step is relatively large. Ringing does not affect the quality of simulations considerably, but prevents the growth of the integration step, despite the fact that the physical process tends to a steady state.

The 2nd-order Adams method (21) applied to the model equation $\dot{y} = \lambda y, y(0) = y_0$ gives expression (22).

$$y_n = y_{n-1} + \frac{h}{2} (\dot{y}_{n-1} + \dot{y}_n) \tag{21}$$

$$y_n = y_0 \left(\frac{2 + h\lambda}{2 - h\lambda} \right)^n \tag{22}$$

$$\lim_{h \to \infty} \left(\frac{2 + h\lambda}{2 - h\lambda} \right) = -1 \tag{23}$$

As the integration step increase the (23) holds, which leads to the state variable sign change $(-1)^n$ at every integration

step. To suppress ringing, it is necessary to eliminate the influence of the property (23) to the solution. There are several approaches to this problem. For the chosen integration scheme, good results are obtained by replacing $\dot{y}_n$ by the value of $\dot{y}_n^{BDF}$, computed by (24) after obtaining the solution of (21) as proposed in [10].

$$\dot{y}_n^{BDF} = \frac{\alpha^2 y_{n-2} - (1+\alpha)^2 y_{n-1} + (1+2\alpha) y_n}{h_n (1+\alpha)} \qquad \alpha = \frac{h_n}{h_{n-1}} \tag{24}$$

At the step $n+1$ the expression (21) would use $\dot{y}_n^{BDF}$ instead of the value $\dot{y}_n$ computed at the step n.

$$y_{n+1} = y_n + \frac{h}{2}\left(\dot{y}_n^{BDF} + \dot{y}_{n+1}\right) \tag{25}$$

Since replacement $\dot{y}_n^{BDF} \to \dot{y}_n$ changes integration method stability properties, it is applied only when ringing is detected. The extra conditions to apply this artificial damping are $h_n > 0.1$ s and a cool-down period of several steps before the next damping. The effect of damping is shown in Fig. 1 and Fig. 2.

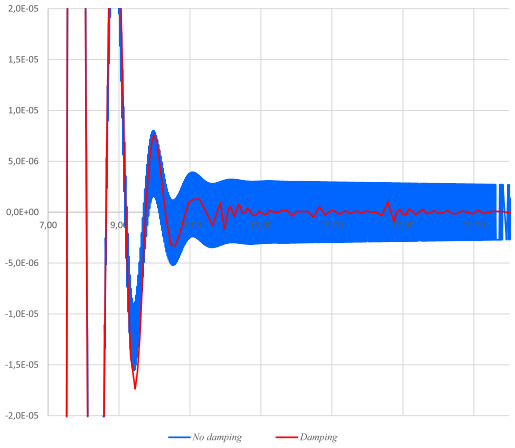

Fig. 1. The ringing suppression effect of 2nd-order Adams damping on state variable evolutions

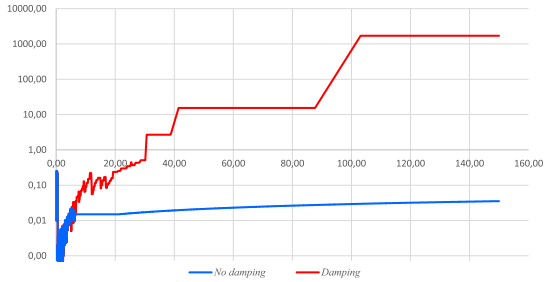

Fig. 2. The ringing suppression effect of 2nd-order Adams damping on integration step size evolutions (log scale)

E. The integration process

The integration method and scheme considered above assume that the (1) is continuous. When simulating practical transients, this condition is not met since some of the f and g are not smooth over their entire domain due to constraints or are discrete. In addition, to simulate discrete events, some of the equations of the system have to be modified. In some cases, the dimension of the system also changes. Problems in which system (1) is not continuous are the separate problem class for DSAR – Differential Switched Algebraic and State Reset equations [11]. An obvious approach for solving such problems is integrating a sequence of continuous DAEs.

The problem of DSAR treatment consists of the arrangement of a transition from one continuous DAE to the next one at a time instant defined with consistent initial conditions. The last problem is not trivial, since the nonlinearity and the presence of discrete functions make multiple solutions possible. Due to the additional information available in the history of solution and specified in the form of rules in the models [1], the ambiguity can be resolved by a formal choice of the most plausible combination of state variables.

Thus, a full-fledged algorithm for transient stability simulation should implement the integration process, which governs the integration scheme and solves the following problems:

1) Discrete events management.
2) Discontinuities handling.
3) Calculation of initial conditions and restart of integration
4) State events time instants location.

The discrete event is a command to modify (1) at an arbitrary time instant t_d. Command execution requires at least a restart of the integration. In general, some modifications of (1) structure are also needed. The new consistent initial conditions at t_d must be calculated to resume integration.

Discrete events can be divided into two groups: time events and state events [3]. The events of the first group occur at the moments of time that are known before simulation. These include simulation stop time, application of disturbances or triggered time delays. State events occur when some conditions are met. They are connected with threshold elements, limiters, comparators, relays and so on. Their time of occurrence must be determined during the simulation.

Discrete event processing is independent of the group it is related to, if t_d is known. This instant must be located for state events. Assume integration method steps from t_{n-1} to t_n and $t_{n-1} < t_d < t_n$. Discrete event processing must be handled in the following sequence:

1) At t_{n-1} choose $h_d = t_d - t_{n-1}$ and step to t_d. Let $t_n = t_d$.
2) Estimate local truncation error of the step.If tolerance is violated, reject the step, return to t_{n-1} and choose a new step size according to the error.
3) At t_d store simulation results. Since the discrete event considered instant, the time before the event is denoted by t_{d-} and the time after the event is t_{d+}. Strictly $t_{d+} > t_d > t_{d-}, t_{d+} - t_{d-} \to 0$, but to preserve integration method numerical stability in practice $t_d = t_{d-} = t_{d+} - 0.1 h_{min}$, where h_min the is smallest integration step allowed.
4) Modify f and g according to discrete event and solve system (11) with $h_n = 0$. The latter implies $y_n = y(t_{d-}) = y(t_{d+}) = const$. The solution of (11) by an

iterative method may be difficult due to a bad initial guess. Therefore, the linear method is used to improve the initial guess and the Newton scheme uses step control, as opposed to the Newton scheme used for integration step (15).

5) In case (11) solved successfully, store simulation results at t_{d+}.
6) Restart the integration scheme from initial conditions x_n, y_n obtained from (11) solution. Restart sets the integration method order to 1st and $h_n = h_{min}$.

Since a state event instant must be located, special zero-crossing functions are used to model the conditions of a state event in the form (26).

$$z_i(t) > Ref_i(t) \tag{26}$$

where $Ref_i(t)$ is zero-crossing function of the i-th state variable.

Usually zero crossing functions are considered as algebraic equations in (1) in the form (27).

$$c_j(t) = z_i(t) - Ref_i(t) \tag{27}$$

which sign change $c_j(t_{d-})c_j(t_{d+}) < 0$ indicates t_d location. This condition can be checked at the final stage of the integration step. For zero-crossing functions, whose signs have been changed $t_d \in [t_{n-1}, t_n]$ should be located with accuracy up to h_{min}. Since the integration scheme is based on the multistep integration method at t_n a Nordsieck vector is available for all state variables (including algebraic ones) up to 2nd order (28).

$$\left[c_{jn}, h_e\dot{c}_{jn}, \frac{h_e^2}{2}\ddot{c}_{jn}\right] \tag{28}$$

Using (28) one can solve equation (29) with respect to h_e

$$c_{jn} + h_e\dot{c}_{jn} + \frac{h_e^2}{2}\ddot{c}_{jn} = 0 \tag{29}$$

and calculate $t_e = t_n - h_e$. After the step size correction, the sign change check of (27) is repeated until at least one of the functions changes its sign. Practical simulation demonstrated the possibility of incorrect t_e location due to multiple changes of the sign of (27) at interval $[t_{n-1}, t_n]$. In this case, an additional zero-crossing function in the form $dc_j(t) = \dot{c}_j(t)$, whose sign changes at the extrema of (27), significantly reduces the probability of missing t_d. This approach is proposed in [3] and it does not require to locate t_e for this condition. It is sufficient to detect a sign change of $dc_j(t)$ and then use the bisection method to search for sign change of‘ (27) in the interval $[t_{n-1}, t_n]$.

Explicit use of zero-crossing functions is necessary in software implementations that use standard DAE solvers. In this case, these functions are part of an interface for representing events in the system being solved. In this paper, the custom solver is used, and therefore there is no need to consider zero-crossing functions explicitly. Since there are Nordsieck vectors for variables (26), the solution of (29) can be obtained with the difference between the components of these vectors. Due to this, the dimension of the system being solved does not change to represent the zero-crossing functions. In addition, about 80% of the constraints in the system (1) have $Ref_j = const$, which further reduce the cost of t_e location.

The separate class of discrete state events is time delays. A typical example is a relay that operates when a certain condition is met with delay ΔT_R. Processing models with a time delay requires the control of two related state events - the instants of pick-up and drop, if any. At the pick-up instant t_e, a new time event is generated for $t_e + \Delta T_R$. If the relay drops in $[t_e, t_e + \Delta T_R]$, this event is canceled.

The interesting feature of the delayed relay events is that they can be used as supplemental criteria to stop transient simulation before the designed duration time is reached. In many simulation scenarios, a transient is judged as stable when it has sufficient decay. In other words, the simulation continues until the ”steady state” is reached. If there are delayed events on hold, they can significantly change the transient when applied to the model. Therefore, when transient sufficiently decayed it is necessary to check delayed events and if there are none, it simulation can be safely finished. Neglecting of the events related to the protection of equipment can lead to an erroneous assessment of the stability [12]. Unstable transient detection is much easier in most cases. Besides numerical indicators of instabilities, some practical angle-based criteria can be used to detect loss of synchronous operation and abort simulation to save time.

Some types of state events may have more complex zero-crossing functions than (26). Such an event is, for example, the position of the out-of-sync relay protection impedance point within a designated area, which cannot be represented by the single variable function. In addition, some constraints are defined in the form of piecewise functions, which requires the introduction of a state event and (1) solely for the purpose of switching to a new segment of the constraint due to derivative discontinuity. For such complex state events, a general location procedure has not yet been developed and the search for t_e bisection method is used.

The integration process described above is a classic DSAR solution. The contemporary approach to this problem is shifting towards a rigorous theory of solving systems of differential equations with a nonsmooth right hand side [2]. There is a positive experience of using this approach in the problem of transient simulation [13] with application to DAE. This research field is very promising for future work.

III. Software prototype implementation and simulation results

A. Implementation details

The software based on the approaches described above is implemented in C++. It was designed to run on either the Windows or Linux operating systems. The prototype called ”Raiden” has a limited equipment model library, which includes general models necessary for basic simulations. However, the library can readily be extended with special tools for automatic model implementation [1]. The current library contents are listed in the Table I.

The initial conditions of the transient simulation are determined automatically with the embedded power flow solver,

TABLE I
EQUIPMENT MODELS DESCRIPTION AND FEATURES

Model	Description
Network	Network model in the form of $YU = I$.
Load	Active and reactive power frequency dependent load model in the form of piece-wise load response curve.
Generator	Infinite bus, simplified motion equation model, simplified model in the form of EMFs, full Park's model with 2 or 3 damping windings and refinement of model parameters [14].
Excitation	Basic exciter model, AVR with PID and stabilization circuits, discrete excitation control.

which implements the modified Newton method with initial guesses selection from Seidell or Continuous Newton methods [15], [16].

Raiden can use source data in a proprietary format based on JSON and can import data files in the format of the former transient simulator RUSTab and, to a limited extent, EUROSTAG formats. It is possible to save and load "snapshots" of the state variables and service data of the model at an arbitrary simulation time t_s, which makes it possible to perform subsequent calculations on the same model from the time t_s without the need to re-integrate the interval $[0; t_s)$. By default, during the simulation, all state variables' time series are saved. To reduce the storage space required for results, the data compression method is used [17].

B. Simulation results

Two test cases of different dimensions were used to compare the proposed software with the current implementation of the transient simulator RUSTab. The latter is relatively old software based on explicit methods, but it is known for its stability and simulation quality. The test cases description is shown in Table II.

TABLE II
TEST CASES DESCRIPTION AND ELEMENTS COUNT

Model element	Case 1	Case 2
Buses	887	7387
Branches	1254	9279
Generators	151	624
Loads	730	6757
State variables	8139	38244

A scenario with a duration of 150 s was simulated, including a three-phase short circuit with a duration of 0.1 s, followed by a line trip and its subsequent automatic reclosure. The transient almost completely decays at $t > 45$ s. Simulations were performed using RUSTab and the proposed software. For the RUSTab accuracy on differential variables was set to 10^{-5}, while RUSTab cannot check accuracy on algebraic variables. For the Raiden $Rtol = Atol = 10^{-4}$ were used. A comparison of simulation results of the most volatile state variable is shown in Fig. 3.

In the Fig. 4 comparison of integration step sizes if shown. The Fig. 5 shows zoomed version of the RUSTab step size to depict its step size control results. The implicit method allows much larger step sizes when transient decays due to its superior numerical stability.

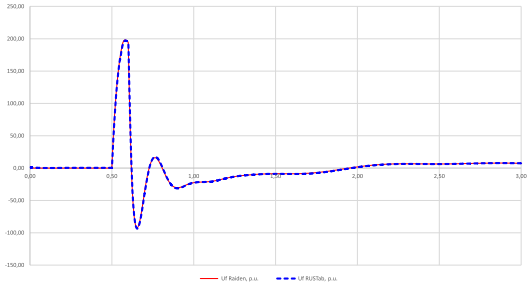

Fig. 3. Case 1 simulation results of AVR outputs

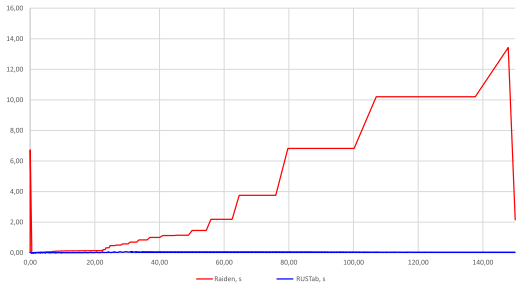

Fig. 4. Case 1 integration steps

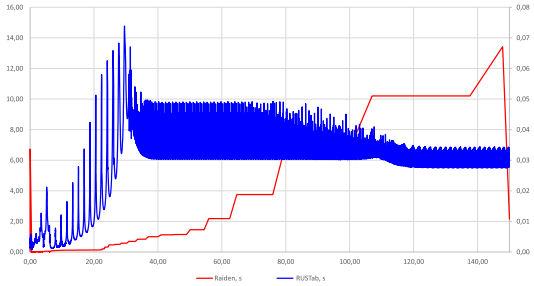

Fig. 5. Case 1 integration steps (zoomed)

The times of simulations are shown in the Table III. It should be noted that the Raiden does not implement any parallel computing features, while RUSTab uses multithreading for vectorizing integration operations.

An additional simulation with a higher tolerance $Atol = 10^{-2}$ was compared with the original simulation. The comparison of these results is shown in Fig. 6 and the step size comparison is shown in Fig. 7.

TABLE III
TIME OF SIMULATION AND STEP COUNT FOR THE CASE 1

	RUSTab 10^{-5}	Raiden 10^{-4}	Raiden 10^{-2}
Time, s	77.9	4.0	3.7
Steps	10628	2503	1842

The difference between simulation with $Atol = 10^{-2}$ and simulation with $Atol = 10^{-4}$ within given tolerance bounds

while computation time decreased by 25%. Further increases in tolerance may lead to a reduction in simulation time and this effect can be used to perform rough estimations of stability. However, it should be noted that this can also lead to an inaccurate location of the state events and significant phase deviations when simulating long-term transients.

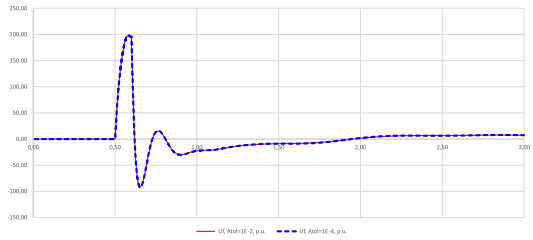

Fig. 6. Case 1 simulation results with different tolerances

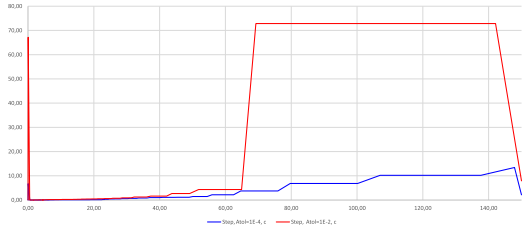

Fig. 7. Case 1 integration step with different tolerances

The second test case dimension is connected to a higher level of switchgear included in the model. Such a models have become more common in recent years. About 30% of the branches in the network are breakers. The network dimension of this model is significantly reduced after topological processing. In this test case, the number of nodes for which equations were generated is 5141, which is approximately 70% of the original number of buses.

A scenario with a duration of 150 s was simulated, including a single-phase short circuit with a duration of 0.08 s, a line trip, and subsequent breaker failure with a delay of 0.255 s. The active phase of the transient decays at $t > 15$ s, but oscillations with a small amplitude persist until the end of the simulation. The comparison of simulation results is shown in Fig. 8.

TABLE IV
TIME OF SIMULATION AND STEP COUNT FOR THE CASE 2

	RUSTab 10^{-5}	Raiden 10^{-4}	Raiden 10^{-2}
Time, s	4685.6	279.1	98.6
Steps	79395	11293	3909

Case 2 was subjected to another simulation with a higher tolerance $Atol = 10^{-2}$. The comparison with original simulation is shown in Fig. 9. There are slight differences between simulation results, caused, perhaps, by stronger BDF damping in the simulation with higher tolerance. When tolerance is increased, the integration scheme can increase step size, which reduces step number and simulation time in this case by 65%.

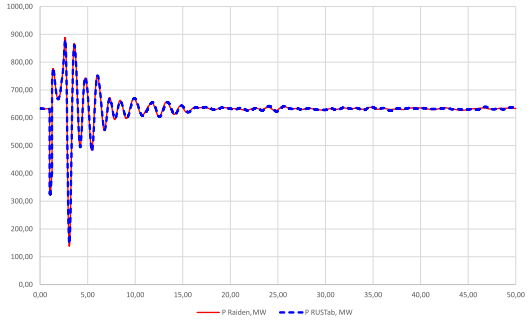

Fig. 8. Case 2 simulation results of the generator output

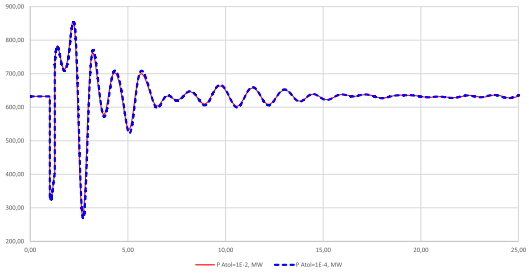

Fig. 9. Case 2 simulation results with different tolerances

IV. Conclusion

References

[1] E. Mashalov, "Automatic implementation of equipment models for transient stability simulation by symbolic manipulations," *Unpublished*, 2022. [Online]. Available: https://rgdoi.net/10.13140/RG.2.2.29110.78404

[2] A. F. Filippov, *Differential Equations with Discontinuous Righthand Sides*, F. M. Arscott, Ed. Springer Netherlands, 1988. [Online]. Available: https://doi.org/10.1007/978-94-015-7793-9

[3] F. E. Cellier and E. Kofman, *Continuous System Simulation*. Springer, New York, 2006.

[4] C. Gear, "Simultaneous numerical solution of differential-algebraic equations," *IEEE Transactions on Circuit Theory*, vol. 18, no. 1, pp. 89–95, 1971.

[5] J. Astic, A. Bihain, and M. Jerosolimski, "The mixed Adams-BDF variable step size algorithm to simulate transient and long term phenomena in power systems," *IEEE Transactions on Power Systems*, vol. 9, no. 2, pp. 929–935, 1994.

[6] E. Hairer, S. P. Nørsett, and G. Wanner, *Solving Ordinary Differential Equations I*. Springer Berlin, Heidelberg, 1993.

[7] P. Linda, "A Description of DASSL: A Differential/Algebraic System Solver, SAND82-8637," Sandia Labs, Tech. Rep., 1982.

[8] "Deliverable d4.1 of the pegase. algorithmic requirements for simulation of large network extreme scenarios," CRSA, RTE, TE, TU/e, Tech. Rep., 2011. [Online]. Available: http://fp7-pegase.com

[9] K. Radhakrishnan and A. C. Hindmarsh, "Description and use of LSODE, the livemore solver for ordinary differential equations," Office of Scientific and Technical Information (OSTI), Tech. Rep., Dec. 1993. [Online]. Available: https://doi.org/10.2172/15013302

[10] J. A. Lee, J. Nam, and M. Pasquali, "A new stabilization of adaptive step trapezoid rule based on finite difference interrupts," *SIAM Journal on Scientific Computing*, vol. 37, no. 2, pp. A725–A746, Jan. 2015. [Online]. Available: https://doi.org/10.1137/140966915

[11] I. Hiskens and P. Sokolowski, "Systematic modeling and symbolically assisted simulation of power systems," *IEEE Transactions on Power Systems*, vol. 16, no. 2, pp. 229–234, 2001.

[12] T. V. Cutsem, M. Glavic, W. Rosehart, C. Canizares, M. Kanatas, L. Lima, F. Milano, L. Papangelis, R. A. Ramos, J. A. dos Santos, B. Tamimi, G. Taranto, and C. Vournas, "Test systems for voltage stability studies," *IEEE Transactions on Power Systems*,

vol. 35, no. 5, pp. 4078–4087, Sep. 2020. [Online]. Available: https://doi.org/10.1109/tpwrs.2020.2976834

[13] M. A. A. Murad, B. Hayes, and F. Milano, "Application of filippov theory to the IEEE standard 421.5-2016 anti-windup PI controller," in *2019 IEEE Milan PowerTech*. IEEE, Jun. 2019. [Online]. Available: https://doi.org/10.1109/ptc.2019.8810738

[14] I. Canay, "Modelling of alternating-current machines having multiple rotor circuits," *IEEE Transactions on Energy Conversion*, vol. 8, no. 2, pp. 280–296, 1993.

[15] F. Milano, "Continuous newton's method for power flow analysis," *IEEE Transactions on Power Systems*, vol. 24, no. 1, pp. 50–57, 2009.

[16] J. A. L. F. J. MarcosTostado-Véliz, Manuel A.Matos, "An improved version of the continuous newton's method for efficiently solving the power-flow in ill-conditioned systems," *International Journal of Electrical Power and Energy Systems*, vol. 124, p. 106389, Jan. 2021. [Online]. Available: https://doi.org/10.1016/j.ijepes.2020.106389

[17] E. Mashalov, "Power system transient stability simulation results compression method," *Unpublished*, 2022. [Online]. Available: https://rgdoi.net/10.13140/RG.2.2.10208.97289

V. Biography Section

Eugene Mashalov Graduated from Electrotechnical Faculty, Ekaterinburg, Urals State Polytechnical University, 1997. Received Ph.D degree in 2000 from the same university. Works for JSC "Scientific and Technical Center of Unified Power System", Ekaterinburg, Russia.
mashalov@gmail.com
www.inorxl.com