Automatic Implementation of Equipment Models For Transient Stability Simulation by Symbolic Manipulations

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

***Abstract*—One of the most important tasks in developing transient stability simulation software is to maintain a library of power system equipment models. This paper discusses the process of automatic implementation of custom equipment transient stability models described in graphical and textual forms or their combination. The implementation process is based on symbolic manipulation and does not require programming skills from users.**

***Index Terms*—Power System Transient Stability Simulation, Power System Equipment modeling, Symbolic Manipulations, Computer Algebra Systems, Compilation.**

I. Introduction

SINCE the beginning of the 2000s, the ability of engineers to create their own custom mathematical models of power system equipment has become one of the major requirements for transient stability simulation software. The intensive development of FACTS technology, the elaboration of control systems and technologies, and the growing demands for energy efficiency increase lead to the fact that dozens of new equipment units appear on the market every year. Often, the transient properties of such equipment differ significantly from the properties of the built-in simulation models. The productivity of the approach to keeping the library of models built into the software up to date by the developers seems to be moderate, since the main task of the developers is to develop the software itself. The professional community of software users is immersed in equipment progress trends and is motivated to obtain accurate simulation results. Therefore, if there are convenient and reliable tools for creating custom models, the process of updating the library with higher quality and at lower costs can be provided directly by users. Modern communication technologies allow the exchange of model development results and unlimited access to the current library online.

Despite the fact that for almost all engineering areas in which some kind of simulation is required, there is at least one software package (and most often several options are available), a unified approach to describing user models has not been developed. One of the universal environments that can be considered a de facto standard is Matlab/Simulink [1]. Model development in Simulink is focused on graphical interface in the form of a block diagram. Another widely used environment, Modelica [2], also claims to have the maximum coverage of engineering areas, but offers an approach to model using a special object-oriented language. In parallel with universal modeling environments, specialized software developed for particular areas is quite successfully developing. The choice of such software is obviously conditioned by the fact that it considers the features and established practice of the engineering discipline, and also allows to effectively use the features of the mathematical description of the problem. In most cases, specialized software provides an acceptable level of functionality without the need to create custom models by tuning to the of built-in ones, and allows to focus directly on the analysis of the results.

Specialized simulation software also includes products for simulation and analysis of transient stability. Almost every product has tools for creating custom models, but even in this relatively narrow niche, a unified approach to their presentation has not been formed. In particular, EUROSTAG [3] and PSS®SINCAL [4] use graphical interface, while DigSILENT PowerFactory [5] offers a programming language in textual format. If we consider in more detail the process of implementing a custom model, that is, its transformation from a representation in which it is developed by an engineer into a representation suitable for embedding in the software, it turns out that the graphical and textual approaches to model development at some point in the implementation process should give identical results. Thus, there is no need to focus on one approach or another, and moreover, they can be combined. Users with little development experience tend to prefer graphical interface block-building tools as they are more visual. As models become more complex, it is often more convenient to use a textual representation, since graphical block diagrams of relatively simple algorithmic expressions can be cumbersome. With major changes in the model, efforts will be required to maintain the visibility of its graphical representation. Since model editing is most often done to increase detail by adding new blocks to existing links, a significant part of the graphic image has to be rebuilt. This paper discusses the process of implementing custom models that allows to perform symbolic transformations that are invariant to the representation method of the initial mathematical description of the model and allows to combine graphical and textual approaches to the model development.

II. Requirements for custom models

Before defining the requirements for the custom models implementation, it is necessary to give a brief description of the problem being solved. Transient stability simulation is in fact is the process of solving a differential-algebraic system of equations (DAE) of the form:

$$\left\{\begin{array}{lcl}\dot{y} & = & f(x(t), y(t), t) \\ 0 & = & g(x(t), t(t), t)\end{array}\right. \tag{1}$$
$$x(t_0) = x_0, y(t_0) = y_0$$

where g and f are smooth vector functions. The solution of (1) is formally to find $x(t)$ and $y(t)$ for $t \in [t_0; T_{end}]$. Since the analytical solution of (1) is impossible in most cases, a numerical integration is used which involves replacing differential part of (1) with finite-difference equations and their sequential solution with some integration method. The integration method builds sequence of approximations $z_n(t_n) = [y_n(t_n), x_n(t_n)]^T$, satisfying the conditions:

$$||z(t_n) - z_n(t_n)|| \leq \epsilon \tag{2}$$

where $z(t_n)$ is analytical solution and $z_n(t_n)$ is approximated solution. Since (1) is usually stiff, mostly implicit integration methods are used, requiring the solution of a nonlinear system of equations at each integration step n.

Transient trajectories are not smooth and subject to discontinuities at particular time instants due to switching and limiting of state variables. Thus f and g are not continuous in $t \in [t_0; T_{end}]$ but only piece-wise continuous, as well as their derivatives. This requires a restart integration method at time instants of discontinuities t_d with new initial conditions $z_d(t_d)$). In some cases structure and dimension of (1) also subject to change. Discontinuities are associated with events that can be divided into time events and state events. Time events are unconditional and has a known time of occurrence. They can be applied to solution at scheduled time instants. State events have only conditions of occurrence and their times must be located during solution.

Based on the above characteristics of the problem, to solve system (1), the following minimum set of procedures should be implemented:

- evaluation of initial conditions at t_0;
- evaluation of residuals of equations;
- construction of a submatrix of partial derivatives;
- time location of the state events t_d;
- evaluation of switched (1) initial conditions at the discontinuities t_d.

The custom model is a subsystem of (1), therefore, a set of procedures identical to the set for the general system must be implemented for it.

When developer is asked for creation of equipment model, he starts by analyzing the given block diagram, forms a system of equations, and then implements a program with functions that ensure the interaction of the model with the software core. The result of that work is an integral part of the software. To implement a custom model, a similar process is required with two differences: the developer does not participate in the process, and the result of the work is not included directly in the software, but operates as external module. The latter is usually, a native machine code executable module in the operating system format. (*.dll for Windows or *.so for Unix systems). Implementing custom model functions in native code eliminates technical differences between custom and built-in models and ensures maximum performance.

The generation of native code in itself is the complex task, since it is necessary to take into account the system architecture features. Therefore, when implementing custom models, to generate native code, standard systems for compiling executable modules are used, for which the source text of the custom model program is preliminarily created in one of the general-purpose programming languages. In this case, such a programming language is called intermediate. In practice, C/C++ and Fortran compilers are often used. Automatically generated programs in these languages are the penultimate stage of the implementation of the custom model, preceding the compilation of the executable module in native code.

III. Model representation in AST-form

Consider the model representation in block diagram and textual forms. As an example, the simple model of automatic voltage regulator is used. Textual form of representation is

```
Uf = limited_lag(Usum, Trv, Ufmin, Ufmax)
Usum = Ku * Vc + K1u * derlag(Vc, T1u) +  lag(Vs, Tbch) - K1if * derlag(If,T1if)
Vs = Kf * Tf * derlag(Su,Tf) + K1f * derlag(Su,T1f)
Vc = Vref - Vg + Ig * Xc
```

Fig. 1. Sample AVR model pesudocode representation

shown in Fig. 1 and block diagram in Fig. 2.

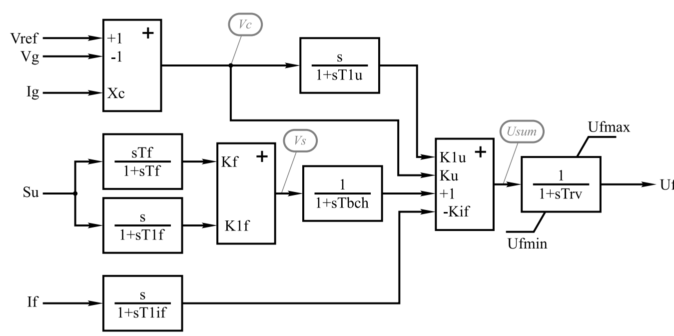

Fig. 2. Sample AVR model block diagram

Both representations use elementary functional blocks: lag, derivative lag and limited lag. Ready-made elementary functional blocks are included in custom model toolbox to simplify model creation and enable efficient implementation by software core instead of letting the user build these blocks from individual differential equations. In addition, the implementation of the elementary blocks in the software core allows to use common procedures for discrete events location and processing and hide these details from the user. If necessary, differential equations can be explicitly specified using the integrator block.

Formally, the block diagram of the model corresponds to a directed graph whose vertices are elementary blocks, and edges are defined by links. Compared to a conventional graph, the order in which the edges in a node are listed matters, because elementary blocks have positional arguments that must be specified in a particular order. This imposes some restrictions on the algorithms used in the implementation of custom models. The block diagram representation of custom models turns out to be easier to use for implementation compared to textual one, since the order of calculations is readily available by traversing graph from outputs to inputs, and validation of links and parameters is performed by the block diagram editor.

Creating a custom model in a text representation is reduced to describing a system of equations using generally accepted mathematical operators and a set of built-in functions that also include elementary blocks. The system of equations must link the input and output variables. This provides for the possibility of using internal variables for structuring the system or introducing feedback. The description of the system is far from an intermediate language, so its transformation is required to implement the model. The textual representation is not as visual as the block diagram, but with some skill, it greatly speeds up the creation of models. At least text editors today remain the main means of professional software development and there is no trend towards the transition to graphical environments. It should be emphasized that the description of the model in the textual representation cannot be called a program in the traditional sense of the term. The program sets the sequence of actions, while the solution sequence is not defined for th e system of equations in textual representation. However, the implementation of the model must be nothing more than an ordinary imperative program in the form of an executable module, therefore, in the process of transformations, the order of calculations must be determined.

To transform the custom model source data in text format, some general-purpose parsers are widely used. They are the programs that allow to perform lexical and syntactic analysis of the source text according to a given dictionary and form the required data structure for further processing. Almost all programming languages use parsers in one way or another to generate intermediate object code from source code. Existing parsing technologies can also be used to analyze the textual representation of the model's system of equations. In the process of analyzing the original textual representation, the control of compliance with the specified rules and the detection of syntactic errors are performed. As a result of the analysis, a tree structure is generated that allows performing the required transformations by manipulating the tree. To represent the system of equations of the custom model, a structure in the form of the abstract syntax tree (AST) is suitable. The nodes of such a tree are the operators, and the edges define the mutual links of the operators. One of the possible options for converting the textual representation of the AVR model from the example Fig. 2, 1 into the AST is shown in Fig 4.

In general, this structure is not a tree. The use of variables as operands of functional blocks turns the structure into a graph, which, in the presence of feedbacks, cannot be reduced to the form of a tree. If there are several output variables, the structure turns into a forest. The traditional use of the term ”tree” is due to the fact that the structure is hierarchical and determines the order of links of nodes. The AST shown in the Fig 4. Cycles on system variables are shown by dotted

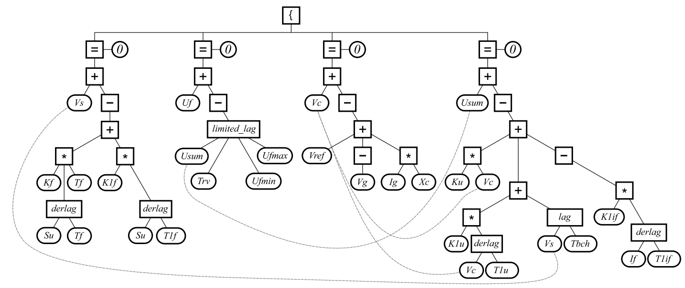

Fig. 3. Sample AVR model AST-tree

lines. If the edges of the connections of variables are not taken into account, then the graph can be considered directed from the lower nodes to the upper ones. The edges of the variable connections do not have a given direction, but it can be determined provided if there are no feedbacks.

The root node of the tree marked as $\boxed{\{}$ denotes a system of equations. Nodes marked as $\boxed{=}$ represent individual equations. An equation node has two child nodes: a left a right. In AST, equations of the form $f(x) = 0$ are accepted, so if a non-zero expression appears on the right side in the process of transformation, it is transferred to the left side of the equation with a negative sign. Each equation node can resolve one of the model variables.

The form of AST for a given system of equations is not unique. For ASTs there is an identity rule, similar to the rule for mathematical expressions. Two ASTs are identical if the values of the root nodes are equal on the entire set of values of the variables included in them in their domains of definition. While maintaining identity, it is possible to perform AST transformations using subtree manipulations, achieving certain properties from the resulting AST. All manipulations of this kind are performed at the symbol level, that is, with variables whose values are unknown. Identity is preserved through the use of the rules of algebra and logic in the transformations. The AST structure is convenient for these transformations from an algorithmic point of view.

Taking as intermediate variables V_c, V_s and U_{sum} at the figure points marked in the block diagram of the model in Fig. 2, by traversing the graph from outputs to inputs, it is easy to obtain an AST identical to Fig. 4. One can also get the AST identical to shown in Fig. 4 by transforming an arbitrary version of the AST built from block diagram. Thus, the AST can be used as an intermediate structure that is invariant with respect to the representation of the custom model. To build it, it is necessary to combine the analysis of the model graph with the analysis of text expressions. Due to this, the possibilities of describing the custom model in block diagram form can be significantly expanded. In particular, for the inputs and outputs of elementary blocks, mathematical expressions can be additionally specified in the textual representation. At the same time, the block diagram of the model remains compact and clear, and the user has almost unlimited possibilities for expanding the functionality of the model. The construction of an AST with such a combined approach begins with a traversal of the model graph, with a transition to the analysis of mathematical expressions. The results of parsing expressions are included in the graph AST as subtrees.

After transformation the representation of the model into the form of the AST, the main stage of the model implementation begins, which includes the following operations:

- classification of variables;
- validation of the initial system of equations;
- simplification and optimization of mathematical expressions;
- generation of expressions for calculating partial derivatives;
- determination of the order of solving the system of equations by variables causalization.

IV. Custom model symbolic transformation

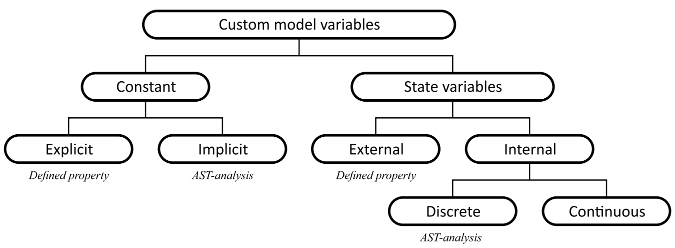

Fig. 4. Classification of the custom model variables

TABLE I
State variable count for basic transient stability equipment models

Model	Description	Count
Bus	Voltage angle and magnitude, filtered voltage slip	4
Load	Active and reactive power	2
Generator	Simplified generator model with active and reactive power, voltage and current on dq-axes, angle, slip, transient and excitation EMFs	10
Exciter	Voltage at bus, output	3
AVR	Excitation EMF, voltage, slip and excitation current filtered derivatives, output, filtered slip	12
DEC	Discrete excitation control: 2 relays, 2 triggers, voltage magnitude	7

To get an idea of the amount of data to be stored consider the number of state variable for basic transient stability equipment models in the table I. The equipment models used are not highly detailed, but overall practical model dimension gives 6624 state variables. For 60s simulation with output step 10^{-2}s and double precision representation of results the required amount of memory will be 303.2MB. Of course output can be limited to a selected set of state variables to reduce storage requirements, but this selection require extra care for various cases of contingencies.

The storage required for resulting time series depends on model dimension, simulation duration and accuracy, as well as transient response features. State variables values are computed on each DAE integration step h, but the results go to output with much slower rate than integration step $H_p >> h$. When integration uses $h < H_p$, intermediate instants in $[t; t + H_p]$ are not stored, except instants with discrete events, such as switching, limiting and contingencies. At discrete instants results are stored before the event in t_d^- and after the event in t_d^+, $0 < t_d^+ - t_d^- < h_min$. Such a representation is much more convenient for the analysis of discrete events, since it preserves discontinuity, but it also increases the amount of data stored, and it grows in proportion to the frequency of discrete events.

When transient decays, state variables amplitude oscillations become smaller, which increases integration method extrapolation accuracy and allows to increase h appropriately. When $h > H_p$ result output will use h instead of H_p. If transient decay is slow or transient is unstable its simulation results will require much more storage, as integration step is unlikely to reach H_p.

Low-precision transient simulation with limited error tolerances requires less memory since h can usually be greater than H_p, but the quality of the simulation cannot be considered as a storage reduction factor.

V. Transient stability results features and general purpose compression algorithms

Consider the simulation results time series as subject to data compression:

1) Simulation results are sequential time series. In most cases they are approximations to piecewise smooth functions, which are DAE solutions. The difference of two successive values are small, suggesting high data redundancy.
2) All time series values are associated with common time values. It can be used to store the only time data for all series.
3) Some state variables are discrete or change infrequently. This is also true for state variables that have reached their limits. The values of these state variables remain fixed for a relatively long period of simulation time or vary within a small range of integration accuracy.
4) When the transient decays, most state variables tend to steady state with negligible oscillations.
5) Any state variable can be subject to discontinuity due to discrete event. It is assumed that discrete events frequency is limited in most cases.

When data compression is needed one of many available commercial or free software libraries can be used. Most of data compression algorithms are general purpose, and do not consider data features. Data compression is based on redundancy detection and removal [**?**], [**?**]. Some methods use preconditioning to improve further compression ratios [**?**] or dictionaries with most frequently occurring data sequences. All of these algorithms assume that the data to be compressed is fully available or available by relatively large successive blocks. Decompression assumes lossless restoration of the original data from the compressed and also operates on full data set. This leads to memory requirements to fit data to be processed.

Transient stability simulation results may contain data for tens of thousands of state variables. Depending on the engineering problem, a large subset of them may need to be worked on, but it is unlikely that a complete set of data will be required. Thus, it is desirable to be able to decompress only a subset of the results for selected state variables. In order to achieve this capability from the general purpose algorithms, it is necessary to compress the data for each state variable separately, which will lead to an increased CPU, memory and storage overhead.

The above peculiarities of general purpose algorithms make them less efficient for simulation results compression. General purpose algorithms should be adapted to work on time series. The table II shows relationships between simulation results features, general purpose compression algorithms peculiarities and possible workarounds to use algorithms to work with results.

TABLE II
GENERAL PURPOSE COMPRESSION FOR SIMULATION RESULTS

Results feature	General Purpose compression peculiarity	Workaround
Available sequentially as integration progresses	Requires access to full data set or it's large parts	Intermediate buffer to accumulate results to be compressed
Access to selected results	Restores full compressed dataset	Much more data should be decomemressed than needed

One can consider separate compression of each time series or their groups by general purpose algorithms. This approach will require separate instances of the algorithm as well as increase amount of memory required. For time series consisting of thousands of separate channels memory amount may be inappropriate.

There are compression algorithms designed for specific data. Lossless audio compression algorithms, for example, work actually on time series [**?**]. But those algorithms are designed for fixed sample rates and integer sample values, which is not the case for simulation results. Audio compression works well with multiple channels, but not with thousands of simultaneous separate channels, due to performance penalties. A high compression ratios achieved by trying several compression methods and further best ratio selection. This also may lead to performance drop.

Data compression performance is one of most important features to be considered. When data compression is used for transient stability simulation results it is auxiliary function, as system resources must be used for computation. A reasonable approach is to run result compression in separate process/thread and give it enough CPU time to process the computed results while computing the next results. Thus system resources will be optimally distributed and synchronization bottlenecks will be avoided.

VI. TRANSIENT STABILITY SIMULATION RESULTS COMPRESSION ALGORITHM

Besides algorithms reviewed above which are suitable for compressing general or media data, there are algorithms designed for scientific data exchange. The are widely used for MPI systems [**?**]. When the bandwidth is limited data compression can improve performance even at additional cost on processing. These algorithms are designed for 32- or 64-bits floating point values and exploit encoding scheme for IEEE-754 numbers. Consider 64-bit value pattern:

Bits	63	62-52	51-0
Purpose	Sign	Exponent	Mantissa

Let a and b be floating point numbers, and $a \approx b$. This means most exponent and most of mantissa bits will match between a and b. If we compute exclusive or $c = a \oplus b$ the high bits of c will be zero. Higher non-zero bits count in c will decrease as a approaches to b. If one can count higher non zero bits, c can be represented with the following encoding scheme:

Z_B	C_B

where Z_B it the number of zero bits divided by 4, C_B is non-zero bits sequence. To store Z_B 4 bits are needed. To store C_B — $64 - 4 * Z_B$ bits. Since $\oplus$ reversible, the original a can be restored by $a = c \oplus b$ with c and b casted to the IEEE-754 bit pattern. It can be seen that encoding and decoding are lossless.

Consider the example of encoding $a \approx \pi$ up to 16 decimal digits. Let $b \approx \pi$ up to 10 decimal digits. The IEEE-754 representation of a and b is as follows:

a	0100000000001001001000011111101101010100010001000010110100011000
b	0100000000001001001000011111101101010100010000010001011101000100

The resulting $c = a \oplus b$ is:

c	00000000000000000000000000000000000000000000	01010011101001011100
	$Z_B * 4$	C_B

Since c has 45 higher zero bits, Z_B=11. The encoded form of c would then be:

Z_B	C_B
1011	01010011101001011100

To store a 24 bits required instead of 64. The compression ratio $K_c = 24/64 = 0.375$. Decoding of a can be done in reverse order.

An encoding scheme can be applied to floating point numbers sequence a_i, if one can generate sequence b_i with values close enough to a_i values. Assume function $f(i)$, whose values sufficiently close to a_i. In [**?**] this function is called predictor and it is based on DFCM algorithm, used for prediction of cache operations for microprocessors. The function algorithm successively builds a hash table of previous sequence values differences. With a sufficient hash table size algorithm can predict repeating patterns in the data sequence. Compression algorithm with predictor is shown in Fig 5.

To compress a_i in the sequence of $a_i = a(t_i)$ the predictor generates b_i to compute $c = a \oplus b$. Then Z_{bi} is determined by counting higher zero bits of c_i and encoded pair Z_{bi}, C_{bi} then ready to store. Predictor is updated with a_i.

To decompress a_i from the stored sequence c_i, the encoded pair Z_{bi}, C_{bi} is mapped to IEEE-754 bit pattern. Predictor

Fig. 5. Compression algorithm.

Fig. 6. Decompression algorithm.

generates b_i and a restores by XOR operation. Predictor is updated with a_i. The b_i sequences generated by the predictor for encoding and decoding are the same, as they use the same a_i.

The closer b_i given by the predictor function, to the original values of a_i, the better compression ratio can be obtained. Therefore, the predictor function should consider the most of the features of the data being processed. A hash table-based predictor suits for general floating point periodic data. It automatically adjusts to the processed sequence and, if a period can be distinguished in the data, it provides high accuracy of the forecast. However, the use of a hash table comes with a major disadvantage of such a predictor, since its storage requires memory (about 16MB for optimal prediction quality at transient simulation duration over 60s). In the considered problem of compression of the results of transients simulation, an individual hash table must be created for each state variable.

It was noted above that simulation results in most cases are approximations to piecewise smooth functions. This property gives option to use Lagrange polynomial extrapolation for prediction:

$$b_i = b_i(t_i) = \sum_{m=1}^{np+1} a(t_{i-m}) l_m(t_i), \tag{3}$$

with basis polynomials:

$$l_m(t_i) = \prod_{k=1, k \neq m}^{np+1} \frac{t_i - t_{i-k}}{t_{i-m} - t_{i-k}} \tag{4}$$

where np is the order of the polynomial, which also the order of predictor. The Lagrange polynomial does not require t_i to be equally spaced. This property allows to output the results from integration method directly without dense output processing.

The prediction method used for encoding is similar to the prediction method for DAE integration. For such a predictor to work, it is sufficient to have a vector of previous values with a depth equal to the order of the predictor. The order can be changed, if necessary, at virtually no additional cost. Increasing the order of the predictor improves the quality of prediction for high-order components of transient. Considering also that all values of the results are related to common time values, the basis polynomials 4 can be calculated once for all state variables that going to output at the current step t_i. Thus, for the results, it is optimal to choose a predictor with Lagrange extrapolation, which requires a negligible amount of memory and makes it possible to eliminate redundant computations.

The order of the predictor polynomial can vary from zero to some n_{Pmax}. For the first step of transient stability simulation, the value of the DAE initial conditions is used as the predictor value. The next step gives a linear extrapolation, and so on. It has been found empirically that the best compression ratio gives the value $n_{Pmax} = 4$. A the order increases prediction worsens. Presumably, the reason for this is the Runge phenomenon, since most of the results are stored with a step h_P, with slight deviations.

As noted above, for discrete changes, the time instants t_d^- and t_d^+ are preserved. Since $t_d^- \neq t_d^+$, the proposed compression method remains operational, but at the instants of discrete changes it can degrade the compression ratio, as the function undergoes a discontinuity and polynomial extrapolation cannot give a qualitative prediction. In the instants of discrete changes, the integration stops and new initial conditions are calculated for t_d^+, after which the integration resumes. The order of the predictor after calculating the new initial conditions is reset to zero and increases to n_{Pmax} as new integration results are accumulated. This approach to processing discrete changes allows to maintain an acceptable compression ratio, since it takes into account the discontinuity of the function of the simulation result.

With an exact match between the predicted and stored value of the simulation result, the encoding scheme allows to reduce the representation to 1 byte - F0 (in hexadecimal), while $K_c = 0.125$. Such prediction accuracy is unlikely for time-varying parameters, but for piecewise constant parameters such as outputs of discrete elements, or parameters that are at the limits, this is easily achievable. Taking into account the fact that the time intervals during which the values of such parameters do not change can be long, it is possible to further improve the compression ratio by using Run Length Encoding (RLE). Before saving, the results compressed by the proposed method are accumulated in a fixed size buffer with a repetition counter. If the buffer is not empty and the value of the current calculation result is the same as the previous one, the current value is not written to the buffer, but instead the repeat counter is incremented. Thus, a series of repeating consecutive values is encoded by the given value and the number of its repetitions. If the repeat counter has a non-zero value, but the previous and current values of the calculation result do not match, the contents of the buffer are stored, after which the buffer size and the repeat counter are reset to zero values.

The simulation results when using additional run-length encoding are stored in the form of blocks. The block is marked with a repeat counter. If the repetition counter is zero, then the block contains non-repeating data compressed by the proposed method. Otherwise, the block contains a series of equal values that can be recovered by decoding the single value and copying it in sequence. The buffer size within 50-100 samples does not lead to a significant memory consumption, but allows I/O operations to process more data, which increases write speed when saving results to file and the speed of access when reading and restoring results for viewing and analysis. File I/O operations, both read and write, are sequential.

The proposed compression method provides exact data recovery, that is, it is a lossless compression method. This property of the algorithm is valuable, but it is redundant for storing the results of transient stability simulation, as it is carried out with the finite accuracy of the integration method. The local error is controlled by predictor-corrector integration scheme:

$$d_i = C \frac{e_i}{|y_i| Rtol + Atol} \leq 1 \qquad (5)$$

where

e_i is the corrector equation error with respect to the state variable y_i;
C is a constant of the integration method;
$Rtol$ is the relative error tolerance;
$Atol$ is the absolute error tolerance.

Obviously, limiting the accuracy while storing the simulation results to the value of $Atol$ will not lead to a deterioration in the quality of the transient stability analysis. Therefore, a preliminary filter can be applied, which rounds values to $Atol$. Filtering by itself does not provide compression, but allows to get the result in the form of a piecewise constant function and use run-length encoding for intervals in which the change in the result value does not exceed $Atol$, as shown on Fig. 7. Filtering does not introduce an additional error into the simulation result and does not affect the quality of the analysis of the simulation result, but improves compression ratio.

Fig. 7. Pre-compression results filtering for RLE in range of $Atol$.

In addition, filtering allows to eliminate the effect that may occur when the result values oscillate around zero. The IEEE-754 representation of a double-precision number contains sign information in the high bit. Slight oscillations in the result values around zero can lead to a sign change. The relatively small absolute prediction error, but affecting the sign, will lead to the fact that when encoding Z_b, counted from the most significant bits, will be equal to zero, and instead of reducing the size of data, the algorithm will increase it, since 4 extra bits will be required for Z_b encoding. Filtering eliminates the sign change caused by oscillations in the results with amplitudes not exceeding the $Atol$, and allows to maintain an acceptable compression ratio. When $Atol$ values of 10^{-4} or more are acceptable, which is usually adequate for transient stability simulation for post-contingency control task, a single precision real number format requiring only 32 bits can be used to store the results. This will halve the size of results at the cost of accuracy, but at the same time retain the ability to use the proposed compression method with minimal changes in the encoding scheme.

VII. Implementation and Comparison with Mainstream Compression Algorithms

To study proposed compression method implementation, series of tests were carried out to store transient stability simulation results for the model with the following parameters:

TABLE III
Test model features

Equipment model	Count
Bus	842
Branch	1189
Generator(simplified Park's)	149
Excitation system (Exciter+AVR+DEC)	145

The tests were performed on a PC with an Intel Core i7-4770 CPU 3.4GHz and 32GB of RAM. A 4GB disk drive was emulated in RAM, to make file data files I/O times negligible.

The total number of state variables in the model was 6892. The Transient with a duration of 100s and a step for the results output $H_p = 10^{-2}s$ in this model was simulated for two cases. In the first case (Case 1), the frequency stabilization channels of the AVR were switched off, due to which the transient process decayed much more slowly compared to the second case (Case 2), in which the AVR stabilization channels were in operation. Absolute integration error tolerance were $Atol = 10^{-7}$. Comparison of the simulation results of the slip of one of the generators of the model is shown in the figure 8:

Fig. 8. Transient cases with different decays.

For Case 2, due to faster decay, the integration method increased the step at the final stage of the simulation over H_p, which can be seen from the location of the markers in the figure 9. For that case, therefore, a significantly smaller amount of the results were saved than for Case 1. The resulting sizes were compressed by the proposed method during the simulation. After simulation, the results were restored and compressed by two mainstream archiving programs: WinRAR 5.5 and 7zip 16.02 with maximum compression settings. The results of the experiment are shown in the table IV. The best results are in bold.

Fig. 9. Integration step impact on result output step.

TABLE IV
Compression performance comparison

	Case 1	Case 2
Proposed method		
Samples count	8052	5298
State vars count	6892	6892
Uncompressed, MB	423.39	278.58
Time, s	**16**	**10**
RAM, GB	**0.003**	**0.003**
Compressed, MB	**169.47**	102.24
Ratio, %	**40.03**	36.70
WinRAR		
Time, s	19	10
Time 1 core, s	69	38
RAM, GB	0.82	0.82
Compressed, MB	175.20	**98.16**
Ratio, %	41.38	**35.24**
7zip		
Time, s	38	25
Time 1 core, s	120	62
RAM, GB	2.94	1.86
Compressed, MB	197.57	106.52
Ratio, %	46.66	38.24

During the comparison, the sizes of the compressed data were measured and the compression ratios were determined, , as well as the time spent on the compression. The compression ratio of the proposed method turns out to be comparable with the compression ratios of mainstream archivers. For a larger amount of results, the proposed method gives the best compression ratio. For a smaller size of results, the proposed method is slightly inferior to WinRAR.

It should be noted that in the comparison, the proposed compression method was run in parallel with the process of the transient stability simulation on one core of an eight-core processor. The load on the processor from the compression method, thus, could not exceed 12.5%. The WinRAR and 7zip archivers were run with no threading restrictions. The average load on the processor when running WinRAR was about 65%, and when running 7zip — 53%. In order to compare the time required by that archivers to process data using only one processor core, they were artificially restricted by the operating system. The execution time of archivers on one processor core is also shown in the table. The amount of memory required by the proposed method is negligible compared to the amount required by archivers that use general purpose compression methods.

The "Compressed" row shows the full size of the results compressed by the proposed method, which, in addition to these results, also includes auxiliary data, for example, a list of state variables with their names and units, a list of model elements, etc., as the compressed size for archivers, is only the size of result data. At the end of the comparison, the results compressed by the proposed method was additionally compressed by WinRAR. The compression ratio for Case 1 turned out to be 96%, for Case 2 — 94%, which can be explained by a better compression of auxiliary data. However, compression of the auxiliary data for the result will require its full decompression when reading for analysis. Considering that the additional possible compression is negligible, the auxiliary data is stored without using compression to speed up the reading of results.

VIII. Conclusion

The proposed method for compressing the results of transient stability simulation makes it possible to obtain a compression ratio close to the compression ratio of mainstream archiving programs that implement general purpose algorithms, with significantly less computational efforts. The amount of memory required for the method is several orders of magnitude smaller than the amount required for general purpose compression algorithms. The method allows to compress the result of the simulation for each state variable separately, therefore, to restore a subset of the data required for analysis, it is not necessary to restore the entire amount of data. The method supports a lossy compression option with a normalized error not exceeding the absolute integration error tolerance. The results obtained are achieved through the use of specific properties of the data of the results of the transient stability simulation. The compression method is implemented as a component of the developed software for transient stability simulation.

References

[1] Simulink - Simulation and Model-Based Design // www.mathworks.com. [2020]. URL: https://www.mathworks.com/products/simulink.html

[2] Modelica Language Documents - Version 3.4 - April 2017. // modelica.org. [2020]. URL: https://modelica.org/documents.

[3] Flexible and secure modeling // www.eurostag.be. URL: http://www.eurostag.be/en/products/eurostag/features/flexible-and-secure/flexible-secure/.

[4] PSS SINCAL Platform // www.simtec-gmbh.at. URL: http://www.simtec-gmbh.at/sites_en/platform.asp.

[5] Francisco Gonzalez-Longatt JLRT, editor. Advanced Smart Grid Functionalities Based on PowerFactory. Springer International Publishing AG, 2018.

[6] Gear C.W. The Simultaneous Numerical Solution Of Differential - Algebraic Equations SLAC-PUB-0723. // IEEE Trans.Circuits Theory, No. 18, 1971. pp. 85-95.

[7] Joachim von zur Gathen J.G. Modern Computer Algebra. 3rd ed. Cambridge University Press, 2013.

[8] Pearl J. Heuristics: Intelligent Search Strategies for Computer Problem Solving. Addison-Wesley Pub, 1984.

[9] Timothy A. Davis E.P.N. Sparse Matrix Methods for Circuit Simulation Problems // In: Scientific Computing in Electrical Engineering. Springer, Berlin, Heidelberg, 2012. pp. 3-14.

[10] Davide Fabozzi S.W.B.W.F.V. Semi-implicit Formulation of Proportional-integral Controller Block with Non-windup Limiter According to IEEE Standard 421.5-2016 // Proceedings Of IREP'2017 Symposium, 2017.

[11] Robert Sedgewick K.W. Algorithms. 4th ed. Pearson Education, 2011.

IX. 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