Power System Transient Stability Simulation Results Compression Method

Eugene Mashalov
STF9
Scientific and Technical Center of Unified Power System
Ekaterinburg, Russia
mashalov@gmail.com

***Abstract*—Transient stability analysis simulation software can produce large amount of the resulting data, representing the time series of parameters. Modern power system models are highly detailed, and the duration of simulation is usually tens of minutes. In addition, power system simulations are generally performed on multiple variants with varying model parameters. The resulting time series for tens of thousands of variables may require significant storage and its cost may be tangible even for modern computer systems. This article discusses the compression algorithm and its implementation, which can losslessly reduce the amount of time series data by about 40% with small computational overhead.**

***Index Terms*—Power System Transient Stability Simulation, Time Series Lossless Compression.**

I. Introduction

MODERN power system transient stability analysis practice is mainly involved in the development of new facilities and also plays a significant role in post-contingency control. The last task requires just a single answer to the question ”will this contingency be stable or not ?” Simple criteria, such as angle or energy-based, can be used to evaluate transient stability, and in most cases, they do not require time series analysis. But for system setup, tuning and incident inquiries, detailed time series is still required. With regard to development tasks, time series is the most important supplement material for design documentation and their study requires at least all machine angles, voltage magnitudes and angles of buses, some control signals. In most cases, a lot more data should be presented. Thus, the ability to store large amounts of transient stability time series should be provided, at least optionally.

The resulting time series are snapshots of model state variables at arbitrary moments of transient. The sampling rate of snapshots is uneven and is conditioned by the integration step. Dense output is not used often, as it produces excess data for long-term transient decay phases. Some model variables have a discrete nature, such as breaker positions and network parameters change. Their dense representation is also inefficient.

To get an idea of the amount of data to be stored, consider the number of state variables for basic transient stability equipment models in the table I. The equipment models used are not highly detailed, but overall practical model dimensions give 6624 state variables. For 60 s simulation with an output step 10^{-2} s and double precision representation of the 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 requires extra care for various cases of contingencies.

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

The storage required for the resulting time series depends on model dimensions, 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 at a 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 increasing 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 simulations with limited error tolerances require 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.

II. 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 is 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 events. It is assumed that discrete events frequency is limited in most cases.

When data compression is needed, one of the many available commercial or free software libraries can be used. Most data compression algorithms are general purpose, and do not consider data features. Data compression is based on redundancy detection and removal [1], [2]. Some methods use preconditioning to improve further compression ratios [3] or dictionaries with the most frequently occurring data sequences. All of these algorithms assume that the data to be compressed is fully available or available in relatively large successive blocks. Decompression assumes lossless restoration of the original data from the compressed data and also operates on the full data set. This leads to memory requirements to fit the 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. Table II shows relationships between simulation results features, general-purpose compression algorithms peculiarities and possible workarounds to use algorithms to work with results.

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 an increased amount of memory required. For time series consisting of thousands of separate channels, the memory amount may be inappropriate.

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 decomressed than needed

There are compression algorithms designed for specific data types. Lossless audio compression algorithms, for example, work actually on time series [4]. 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. High compression ratios are achieved by trying several compression methods and further best ratio selection. This may also lead to performance drops.

Data compression performance is one of the most important features to be considered. When data compression is used for transient stability simulation results, it is an auxiliary function, as system resources must be used for computation. A reasonable approach is to run result compression in a 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.

III. Transient stability simulation results compression algorithm

Besides the algorithms reviewed above, which are suitable for compressing general or media data, there are algorithms designed for scientific data exchange. They are widely used for MPI systems [6]. When bandwidth is limited, data compression can improve performance, even at an additional cost in processing. These algorithms are designed for 32- or 64-bit floating point values and exploit an encoding scheme for IEEE-754 numbers. Consider the 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 exponents and most 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. The higher non-zero bit count in c will decrease as a approaches 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	0100000000001001001000011111101101010100010000001000101110100010 0

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 the floating point number sequence a_i, if one can generate a sequence b_i with values close enough to the a_i values. Assume a function $f(i)$, whose values are sufficiently close to a_i. In [5] this function is called a predictor and it is based on the DFCM algorithm, used for the prediction of cache operations for microprocessors. The function algorithm successively builds a hash table of previous sequence value differences. With a sufficient hash table size, the algorithm can predict repeating patterns in the data sequence. The compression algorithm with predictor is shown in Fig 1.

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. The predictor generates b_i and restores a using an 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 most of the features of the data being processed. A hash table-based predictor suits 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

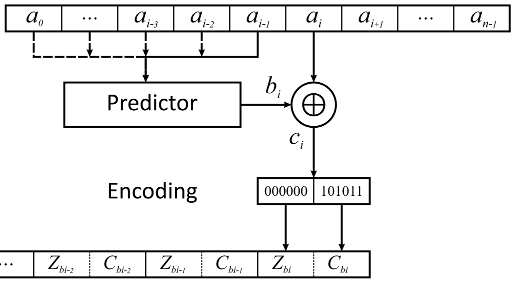

Fig. 1. Compression algorithm.

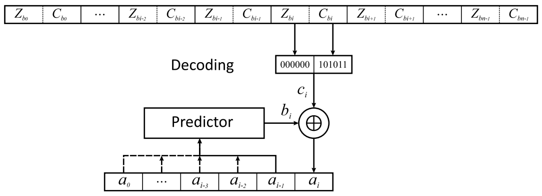

Fig. 2. Decompression algorithm.

a major disadvantage of such a predictor since its storage requires memory (about 16MB for optimal prediction quality at transient simulation durations over 60 s). 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 of piecewise smooth functions. This property gives the option to use Lagrange polynomial extrapolation for prediction:

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

with basis polynomials:

$$l_m(t_i) = \prod_{k=1, k \neq m}^{np+1} \frac{t_i - t_{i-k}}{t_{i-m} - t_{i-k}} \tag{2}$$

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

The prediction method used for encoding is similar to the prediction method used 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 transients. Considering also that all values of the results are related to common time values, the basis polynomials 2 can be calculated once for all state variables that are 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, the 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 us 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 values of the simulation result, the encoding scheme allows the representation to be reduced 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 the outputs of discrete elements or parameters that are at their 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 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 simulations as they carried out with the finite accuracy of the integration method. The local error is controlled by the predictor-corrector integration scheme:

$$d_i = C\frac{e_i}{|y_i|Rtol + Atol} \leq 1 \tag{3}$$

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 one to get the result in the form of a piecewise constant function and the use of run-length encoding for intervals in which the change in the result value does not exceed $Atol$, as shown in Fig. 3. 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 the compression ratio.

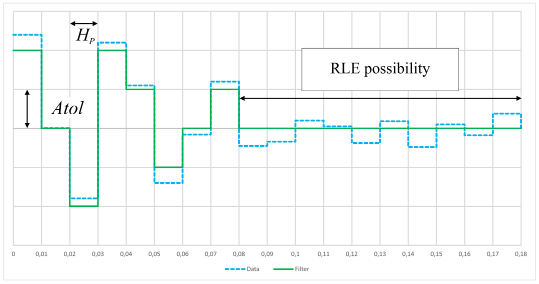

Fig. 3. Pre-compression results filtering for RLE in range of $Atol$.

In addition, filtering allows us 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 the 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 for maintaining 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 tasks, a single precision real number format requiring only 32 bits can be used to store the results. At the cost of accuracy, this will halve the size of results but at the same time retain the ability to use the proposed compression method with minimal changes in the encoding scheme.

IV. Implementation and Comparison with Mainstream Compression Algorithms

To study the proposed compression method implementation, a 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 I/O times negligible.

The total number of state variables in the model was 6892. The transient with a duration of 100 s 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. The absolute integration error tolerance was $Atol = 10^{-7}$. A comparison of the simulation results of the slip of one of the generators of the model is shown in Fig. 4:

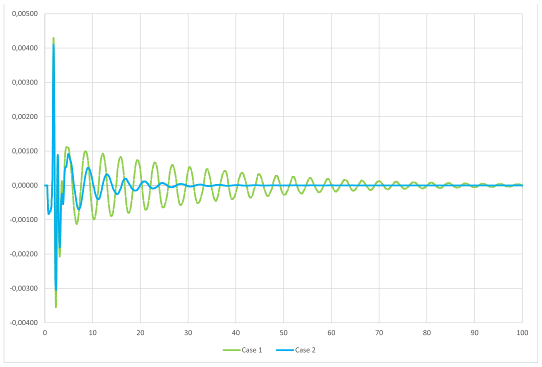

Fig. 4. Transient cases with different decays.

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 Fig. 5. Therefore, a significantly smaller number of the results were saved for that case than for Case 1. The resulting sizes were compressed by the proposed method during 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 table IV. The best results are in bold.

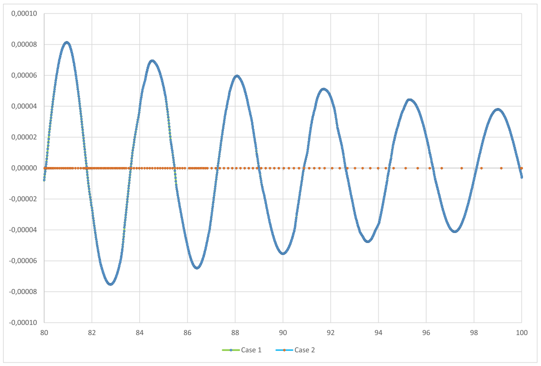

Fig. 5. Integration step impact on result output step.

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 compressed data sizes and compression ratios were measured, as well as the time spent on compression. The proposed method gives the best compression ratio for a larger number of results. For smaller 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 the 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 the result data. At the end of the comparison, the results compressed by the proposed method were additionally compressed by WinRAR. The compression ratio for Case 1 turned out to be 96%, for Case 2 — 94%, which can be explained by the better compression of auxiliary data. However, compression of the auxiliary data for the result will require its full decompression when read 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.

V. 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 effort. 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 one 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 and 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] J. Ziv and A. Lempel, “A Universal Algorithm for Data Compression”, IEEE Transactions on Information Theory, Vol. 23, 1977, pp. 337-343.

[2] T. A. Welch, ”A technique for high-performance data compression”, Computer, vol. 17, no. 6, pp. 8- 19, Jun. 1984.

[3] M. Burrows and D. J. Wheeler, “A Block-Sorting Lossless Data Compression Algorithm”, Digital SRC Research Report 124, 1994.

[4] Khalid Sayood, “Lossless Compression Handbook”. Academic Press, 2003, ISBN 0-12-620861-1, chapter 12.

[5] Ratanaworabhan, J. Ke, and M. Burtscher, ”Fast lossless compression of scientific floating-point data,” in Data Compression Conference, 2006, pp. 133-142.

[6] Camata Jose, Burtscher Martin, Barth, William, Coutinho, Alvaro. Accelerating MPI Broadcasts using Floating-Point Compression. 2010