A Memory-Efficient Parallelizable Method for Computation of Thévenin Equivalents Used in Real-Time Stability Assessment

Jorgensen, Christina Hildebrandt Luthje; Moller, Jakob Glarbo; Sommer, Stefan; Johannsson, Hjortur

Published in:
IEEE Transactions on Power Systems

DOI:
10.1109/TPWRS.2019.2900560

Publication date:
2019

Document version
Peer reviewed version

Citation for published version (APA):
A Memory-Efficient Parallelizable Method for Computation of Thévenin Equivalents used in Real-Time Stability Assessment

Jørgensen, Christina Hildebrandt Lüthje; Møller, Jakob Glarbo; Sommer, Stefan; Jóhannsson, Hjörtur

Published in:
IEEE Transactions on Power Systems

Link to article, DOI:
10.1109/TPWRS.2019.2900560

Publication date:
2019

Document Version
Peer reviewed version

Citation (APA):
A Memory-Efficient Parallelizable Method for Computation of Thévenin Equivalents used in Real-Time Stability Assessment

Christina Hildebrandt Lüthje Jørgensen, Student Member, IEEE, Jakob Glarbo Møller, Member, IEEE, Stefan Sommer and Hjörtur Jóhannsson, Member, IEEE

Abstract—This paper introduces a factor-solve method, which efficiently computes Thévenin equivalents for all buses in the power system. A range of real-time stability assessment methods rely on Thévenin equivalents, and it is therefore essential that these can be determined fast and efficiently. The factor-solve method has runtime for computing Thévenin voltage that scales linearly with system size resulting in runtime of only a few milliseconds even for systems with several thousand buses. The computations only need the sparse admittance matrix for the power system and a sparse factorization resulting in low memory requirements, and furthermore Thévenin impedances can be determined in parallel. The factor-solve method is compared to a reference method, which uses coefficients for super-position to determine the Thévenin equivalents. The reference method is shown to have dissatisfying runtime and complexity. The factor-solve method is tested, parallelized and analysed, which shows a considerable speed-up in computations of Thévenin equivalents enabling them to be computed in real-time.

Index Terms—Algorithms, Power system analysis computing, Real-time assessment, Thévenin equivalent

I. INTRODUCTION

Thévenin equivalent computations are used in assessment of power system stability. Steady-state stability of a generator can be determined as a margin of the maximum power injection using Thévenin equivalents [1], [2]. In [3]–[5] methods for voltage stability assessment are introduced which takes advantage of Thévenin equivalents, while [6] derives sensitivities based on a Thévenin equivalent representation to detect transient voltage sags. In [7] the Thévenin equivalent static contingency assessment (TESCA) method is introduced, which uses Thévenin equivalents to solve the power-flow problem and to evaluate aperiodic small signal stability following a contingency.

The introduction of methods for stability and security assessment as well as methods for defining countermeasures will either result in a competition for the available computational resources or introduce a need for larger computational resources. It is therefore essential, that the methods are first of all fast, but they should also use few computational resources (CPU, memory, etc.). Therefore, Thévenin equivalents should be computed both fast and efficiently. Decomposing the system using a Schur complement has previously been proposed in order to reduce system size and optimize computations.

In [8] the super-position principle is used to determine the contribution of each voltage or current source to the Thévenin voltage. The method take advantage of a Schur complement, which [9] use to decompose the system for dynamic power system computations. In [5] a Schur complement is used to limit the computational burden when finding Thévenin equivalents for load buses, while [10] and the extended version of this [11] applies a Schur complement to efficiently compute Thévenin impedances for generators.

A range of methods for computing Thévenin equivalents for loads is analysed in [12] in order to assess a voltage stability margin. One method uses a Schur complement, and the complexity is estimated to have a cubic overhead, which is concluded to potentially become a burden for large systems. The Schur complement is generally considered to be dense [10], [13] and therefore it is computationally inefficient to determine. In reality all the coefficients for super-position and not just the Schur complement can be dense [14], which means that computing the coefficients is computational expensive.

In [14] different factorization methods are analysed to optimize the method for finding Thévenin equivalents. It is identified, that the Clark Kent LU factorization (KLU) factorization is the most efficient factorization method, however as in [8] it is determined that the greatest share of the execution time is spend on matrix multiplications due to the density of the matrices involved. The coefficients for super-position and Thévenin impedances are computed, whenever the topology of the power system change, whereas the Thévenin voltage can be determined for every new system state to monitor the system or for every iteration of a steady-state analysis. In [8] the method for determining coefficients for super-position was optimized, however the computation of Thévenin voltages was not addressed.

KLU is a factorization method optimized for sparse systems [15]. The method transform the system to block triangular...
form and use approximate minimum degree ordering of the blocks to minimize fill-in before factorizing each block separately. KLU uses block back substitution to solve a system of linear equations. The factorization and the fill-in generated for sparse systems is close to linear with systems size in the context of this paper [10]. KLU is part of the library SuiteSparse [16].

This paper introduces a new method for computing Thévenin equivalents, opposed to the reference method [8], [14]. The method builds on the ideas given in [10] of developing a factor-solve method, where the method here will compute the Thévenin equivalents for the entire system instead of only the Thévenin impedences for generators. The method will take advantage of block back substitution in KLU to avoid computing coefficients for super-position. Thereby, the computationally expensive matrix multiplication of the reference method will be omitted, and additionally the method will be considerably more memory-efficient.

Although [12] estimates a cubic runtime, when using a Schur complement, it will be shown that this is not the case for the methods in this paper. The factor-solve method will facilitate a speed-up of the runtime of both the computation of the Thévenin impedances as well as the Thévenin voltages. The method is split in to a sequential and parallel part compared to the sequential reference method, making it suitable for parallelization, which often enables the performance to become considerably better.

Following the introduction Section I.A gives some examples of real-time stability assessment methods using Thévenin equivalents. Section II describes the reference method for computing Thévenin equivalents using a Schur complement and investigate the runtime of this method. A factor-solve method for computing Thévenin equivalents is proposed in Section III. The method is implemented and tested in Section IV and lastly parallelized for optimal performance. The scalability of the method is evaluated as well as the resulting runtime and memory requirements of the computations. Section V discusses the results and gives perspectives on the method, while Section VI concludes the paper.

A. Thévenin equivalents in stability and security assessment

A range of methods for stability and security assessment use Thévenin equivalents in their computations. To ensure that these method can run in real-time, the Thévenin equivalent computations should be fast and efficient. Below follows some examples of Thévenin equivalents used in assessment methods.

1) Aperiodic small-signal rotor angle stability: In [1] the maximal injectable power by a generator is determined. This is used to determine a margin to the boundary for aperiodic small-signal rotor angle stability as a percentage margin to the maximal injectable power. This is based on algebraically derived equations [17] and given as

\[
\%\Delta P_{\text{inj}} = \frac{P_{\text{inj,max}} - P_{\text{inj}}}{P_{\text{inj,max}}} \cdot 100\%
\]

\[
= \cos(\delta + \phi_{th}) + \frac{1}{1 + \left|\frac{V_i}{V_{th}}\right| \cos \phi_{th}} \cdot 100\%,
\]

where the generator is represented as a voltage source \( V \) with angle \( \delta \) and the remaining grid by its Thévenin equivalent with a voltage source \( V_{th} \) and an impedance \( Z_{th} \) with angle \( \phi_{th} \). \( V_{th} \) is used as the phase angle reference.

A real-time remedial action can be computed as a countermeasure to keep the system from becoming unstable [2]. This is done by computing the reduction in power needed for the critical generator to become N-1 secure. The power is then redispatched to the remaining generators in the system ensuring these also remain secure.

2) Voltage stability: In [5] a voltage stability margin for the loads is determined. This uses the impedance match criterion to determine the maximum deliverable power to the load, which is given by

\[
S_{\text{max}} = \frac{|V_{th}^2| |Z_{th}| - (\text{imag}(Z_{th}) \sin \theta + \text{real}(Z_{th}) \cos \theta)}{2 \text{imag}(Z_{th}) \cos \theta - \text{real}(Z_{th}) \sin \theta}^2
\]

with \( \theta \) being the power factor angle of the load. The margin is then determined using the apparent power of the load \( S_i \)

\[
\%\Delta S_i = \frac{S_{\text{max,i}} - S_i}{S_{\text{max,i}}} \cdot 100\%,
\]

which determines the distance to the boundary of voltage stability.

3) Post-contingency aperiodic small-signal stability: In [7] a method for security assessment is described that determines the aperiodic small-signal stability of the power system following a contingency. The post-contingency steady-state nodal voltages is determined by first computing the Thévenin impedances post-contingency and then computing the Thévenin voltage and nodal voltage angle iteratively until the steady state voltage is found. The voltage angle \( \delta_i \) is determined at each iteration as

\[
\delta_i = \arccos \left( \frac{P_i |Z_{th,i}|}{|V_i||V_{th,i}|} - \frac{R_{th,i} |V_i|}{|Z_{th,i}||V_{th,i}|} \right)
\]

The resulting steady-state nodal voltage can then be used to determine the N-1 stability using the earlier mentioned margin for maximum injectable power by generators.

II. REFERENCE METHOD

A. Schur complement and Thévenin equivalents

In [8], [14] a method for computing Thévenin equivalents is described, which uses coefficients for super-position. Thévenin equivalents consist of a Thévenin impedance \( Z_{th} \) and Thévenin voltage \( V_{th} \). The Thévenin equivalent seen from node \( i \) satisfies

\[
V_{th,i} = V_i - Z_{th,i} I_i
\]

where \( V_i \) is the node voltage and \( I_i \) is the current injected in to the network at node \( i \).

Sources in a circuit can as in power-flow calculations be partitioned in to two sets - current sources (cs) and voltage sources (vs). Floating nodes may be represented as a current source injecting 0 current; loads may be represented as impedances, dependent- or independent current sources; generators with automatic voltage regulator (AVR) may be
represented as voltage sources; internal voltages of manually excited machines may be represented as voltage sources.

The admittance matrix for the system can then be block-wise partitioned as follows

\[
\begin{bmatrix}
I_{cs} \\
I_{vs}
\end{bmatrix} =
\begin{bmatrix}
Y_{cs} & Y_{v\rightarrow c} \\
Y_{c\rightarrow v} & Y_{us}
\end{bmatrix}
\begin{bmatrix}
V_{cs} \\
V_{vs}
\end{bmatrix}
\]

(7)

Eliminating \( V_{cs} \) from (7) yields

\[
I_{vs} = Y_{eq} V_{vs} - Q_{ac} I_{cs}
\]

(8)

with

\[
Y_{eq} = Y_{vs} - Y_{c\rightarrow v} Y_{cs}^{-1} Y_{v\rightarrow c}
\]

(9)

\[
Q_{ac} = -Y_{c\rightarrow v} Y_{cs}^{-1}
\]

(10)

where \( Y_{eq} \) is the Schur complement and \( Q_{ac} \) is the accompanying matrix. This reduction of the network is also known as Kron reduction [18].

The Thévenin impedances seen from node \( i \) is determined by short circuiting all voltage sources and open-circuiting all current sources, which will be

\[
Z_{th,i} = \begin{cases} 
Z_{cs}(i,i) & \text{if } i \in cs \\
Y_{eq}(i,i)^{-1} & \text{if } i \in vs
\end{cases}
\]

(11)

where \( Z_{cs} = Y_{cs}^{-1} \) [8].

Using the definition for Thévenin voltage given in (6) and the above network equations (8)-(10) the Thévenin voltages for the \( cs \) and \( vs \) nodes respectively are defined as

\[
\begin{bmatrix}
V_{th,cs} \\
V_{th,vs}
\end{bmatrix} =
\begin{bmatrix}
Z_{c} & K_{v\rightarrow c} \\
Z_{c\rightarrow v} & K_{v}
\end{bmatrix}
\begin{bmatrix}
I_{cs} \\
V_{vs}
\end{bmatrix} = K \begin{bmatrix}
I_{cs} \\
V_{vs}
\end{bmatrix}
\]

(12)

with

\[
Z_{c} = Z_{cs} - D(Z_{th,cs})
\]

(13)

\[
K_{v\rightarrow c} = -Z_{cs} Y_{v\rightarrow c}
\]

(14)

\[
Z_{c\rightarrow v} = D(Z_{th,cs}) Q_{ac}
\]

(15)

\[
K_{v} = I - D(Z_{th,vs}) Y_{eq}
\]

(16)

Algorithm 1 determines the Thévenin impedances and the coefficient matrix \( K \) needed for computing the Thévenin voltages.

Algorithm 1: Thévenin equivalents

\[
\begin{align*}
L_{cs}, U_{cs} \leftarrow & \text{factorization of } Y_{cs} \\
U_{Z_{cs}} \leftarrow & \text{solve}(L_{cs}, I) \\
L_{Z_{cs}}^{-1} \leftarrow & \text{solve}(U_{cs}^{T}, I) \\
Z_{cs} & \leftarrow L_{Z_{cs}} U_{Z_{cs}} \\
Z_{th,cs} & \leftarrow D(Z_{cs}) \\
Q_{ac} & \leftarrow -Y_{c\rightarrow v} Z_{cs} \\
Y_{eq} & \leftarrow Y_{cs} + Q_{ac} Y_{v\rightarrow c} \\
Z_{th,vs} & \leftarrow D(Y_{eq}) Y_{eq}^{-1} \\
Z_{c} & \leftarrow Z_{cs} - D(Z_{th,cs}) \\
K_{v\rightarrow c} & \leftarrow -Z_{cs} Y_{v\rightarrow c} \\
Z_{c\rightarrow v} & \leftarrow D(Z_{th,vs}) Q_{ac} \\
K_{v} & \leftarrow I - D(Z_{th,vs}) Y_{eq} \\
Z_{th} & \leftarrow Z_{th,vs} \\
K & \leftarrow \begin{bmatrix} Z_{c} & K_{v\rightarrow c} \\
Z_{c\rightarrow v} & K_{v}
\end{bmatrix}
\end{align*}
\]

return \( Z_{th} \) and \( K \)

B. Complexity of reference method

All computations of the reference method are done with sparse matrices and the complexity of the computations will therefore depend on the density of these matrices.

Computing \( Z_{cs} \) will have the complexity \( O(|LZ_{cs}| |cs|) \), which is the most expensive computation in Algorithm 1. By comparison inverting sparse matrices will at maximum have one computation per non-zero element in the matrix for each column in the identity matrix, which gives a complexity of \( O(|LZ_{cs}| |cs|) = O(|U_{cs}| |cs|) \) for solve(\( L_{cs}, I \)) and solve(\( U_{cs}^{T}, I \)). Empirically \( |U_{cs}| \leq |LZ_{cs}| \), and therefore inversion will be computationally cheaper than computing \( Z_{cs} \). The remaining computations of Algorithm 1 will also be of lower complexity, since they involve at least one sparse matrix, and as stated earlier the factorization is close to linear in complexity. Hence the complexity of the algorithm will be \( O(|LZ_{cs}| |cs|) \).

Assuming that \( |LZ_{cs}| \sim |U_{cs}| \) scale close to linear with system size like it’s the case in practise with the number of non-zeros and fill-in generated in the factorization in this context [10] the complexity will be \( O(|cs|^2) \).

Determining Thévenin voltages (12) is \( O(|K|) \), and therefore the computation of Thévenin voltages will depend on the density of the coefficient matrix.

C. Test of reference method

The test method for determining Thévenin equivalents using Algorithm 1 and equation (12) is analysed in MATLAB on a CPU of Intel(R) Xeon(R) CPU E5-2650 v4 @ 2.20GHz. In the implementation KLU is used as the factorization method, due to its efficiency for sparse systems.

The test systems are given in Table I. The Pegase and Polish-Winter systems can be found in MATPOWER [19], the PTI systems are included in the PSS@E 33.0 examples and Nordic32 can be found in [20]. EECC-PSSE-33-0 is a representation of the American Eastern interconnection.

<table>
<thead>
<tr>
<th>Case</th>
<th>no. of buses</th>
<th>no. of vs nodes</th>
<th>non-zeros in ( Y )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nordic32</td>
<td>46</td>
<td>20</td>
<td>160</td>
</tr>
<tr>
<td>Pegase89</td>
<td>89</td>
<td>12</td>
<td>501</td>
</tr>
<tr>
<td>Pegase1354</td>
<td>1354</td>
<td>260</td>
<td>4774</td>
</tr>
<tr>
<td>PTI-WECC-1648</td>
<td>1648</td>
<td>313</td>
<td>6680</td>
</tr>
<tr>
<td>Polish-Winter99</td>
<td>2383</td>
<td>327</td>
<td>8155</td>
</tr>
<tr>
<td>Polish-Winter03</td>
<td>2746</td>
<td>374</td>
<td>9344</td>
</tr>
<tr>
<td>Pegase2869</td>
<td>2869</td>
<td>510</td>
<td>10805</td>
</tr>
<tr>
<td>Polish-Winter07</td>
<td>3012</td>
<td>347</td>
<td>10144</td>
</tr>
<tr>
<td>PTI-EECC-7991</td>
<td>7917</td>
<td>1325</td>
<td>32211</td>
</tr>
<tr>
<td>Pegase9241</td>
<td>9241</td>
<td>1445</td>
<td>37655</td>
</tr>
<tr>
<td>Pegase13659</td>
<td>13659</td>
<td>4092</td>
<td>50909</td>
</tr>
<tr>
<td>EECC-PSSE-33-0</td>
<td>29827</td>
<td>3780</td>
<td>107527</td>
</tr>
</tbody>
</table>

Table II shows the density of the coefficient matrix for each test system. Here the density is given as the number of non-zeros and as the percentage of non-zeros to the maximum size of the matrix. Table III shows the resulting runtime.
TABLE II
DENSITY OF K FOR TEST SYSTEMS

<table>
<thead>
<tr>
<th>Case</th>
<th>non-zeros in K</th>
<th>density of K (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nordic32</td>
<td>547</td>
<td>25.9</td>
</tr>
<tr>
<td>Pegase89</td>
<td>7663</td>
<td>96.7</td>
</tr>
<tr>
<td>Pegase1354</td>
<td>1120078</td>
<td>61.1</td>
</tr>
<tr>
<td>PTI-WECC-1648</td>
<td>1706580</td>
<td>62.8</td>
</tr>
<tr>
<td>Polish-Winter99</td>
<td>2825717</td>
<td>49.8</td>
</tr>
<tr>
<td>Polish-Winter03</td>
<td>3027555</td>
<td>40.2</td>
</tr>
<tr>
<td>Pegase2869</td>
<td>2961793</td>
<td>36.0</td>
</tr>
<tr>
<td>Polish-Winter07</td>
<td>4265858</td>
<td>46.4</td>
</tr>
<tr>
<td>PTI-EECC-7991</td>
<td>4852962</td>
<td>71.6</td>
</tr>
<tr>
<td>Pegase9241</td>
<td>42515659</td>
<td>49.8</td>
</tr>
<tr>
<td>Pegase13659</td>
<td>184786127</td>
<td>99.0</td>
</tr>
<tr>
<td>EECC-PSSE-33-0</td>
<td>854782395</td>
<td>96.1</td>
</tr>
</tbody>
</table>

TABLE III
RUNTIME OF ALGORITHM 1 AND OF COMPUTING THÉVENIN VOLTAGES

<table>
<thead>
<tr>
<th>Case</th>
<th>Runtime (s) Algorithm 1</th>
<th>Runtime (ms) Vfh</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nordic32</td>
<td>2.98 × 10⁻⁴</td>
<td>1.27 × 10⁻²</td>
</tr>
<tr>
<td>Pegase89</td>
<td>1.34 × 10⁻³</td>
<td>3.13 × 10⁻²</td>
</tr>
<tr>
<td>Pegase1354</td>
<td>0.11</td>
<td>2.74</td>
</tr>
<tr>
<td>PTI-WECC-1648</td>
<td>0.20</td>
<td>4.62</td>
</tr>
<tr>
<td>Polish-Winter99</td>
<td>0.30</td>
<td>7.08</td>
</tr>
<tr>
<td>Polish-Winter03</td>
<td>0.31</td>
<td>8.50</td>
</tr>
<tr>
<td>Pegase2869</td>
<td>0.41</td>
<td>7.65</td>
</tr>
<tr>
<td>Polish-Winter07</td>
<td>0.42</td>
<td>10.51</td>
</tr>
<tr>
<td>PTI-EECC-7991</td>
<td>6.31</td>
<td>118.13</td>
</tr>
<tr>
<td>Pegase9241</td>
<td>11.58</td>
<td>118.95</td>
</tr>
<tr>
<td>Pegase13659</td>
<td>36.70</td>
<td>530.13</td>
</tr>
<tr>
<td>EECC-PSSE-33-0</td>
<td>253.08</td>
<td>2519.75</td>
</tr>
</tbody>
</table>

For small systems the method is viable and the algorithm has a runtime of a few milliseconds, however as the system size and complexity grows, so does the runtime. Figure 1 shows the runtime of Algorithm 1 and the runtime for computing Thévenin voltages $V_{th}$ (12) plotted against system size.

As expected, the figure shows complexity that is close to quadratic to system size for Algorithm 1. The complexity is resulting in a runtime for $V_{th}$, which is also 5 times larger. This is consistent with the analysed complexity.

### III. INTRODUCTION OF FACTOR-SOLVE METHOD

Performance of Algorithm 1 and the computation of Thévenin voltages is dissatisfying and doesn’t scale well with system size, but it turns out that there is a potential for improvements. KLU solves a system by using block back substitution. By use of this solver it is possible to find the Thévenin voltages without computing the coefficient matrix.

The equations from (12) can be written as

$$V_{th,cs} = (Z_{cs} - D(Z_{th,cs}))I_{cs} - Z_{cs} Y_{v→c} V_{vs}$$

(17)

$$V_{th,vs} = D(Z_{th,vs}) \mathbf{Q}_{ac} I_{cs} + (I - D(Z_{th,vs}) \mathbf{Y}_{eq}) V_{vs}$$

(18)

Defining $\tilde{V}$ as

$$\tilde{V} = Z_{cs} (I_{cs} - Y_{v→c} V_{vs})$$

(19)

and inserting this in to (17) and (18) gives

$$V_{th,cs} = \tilde{V} - Z_{th,cs} I_{cs}$$

(20)

$$V_{th,vs} = V_{vs} - Z_{th,vs} (Y_{cs} V_{cs} + Y_{c→v} \tilde{V})$$

(21)

Solving $Y_{eq} x = b$ for $x$ i.e. finding $x = Z_{cs} b$ can be determined by block back substitution using the factors $LU$ from the KLU factorization. This will be defined as $\text{klu}(LU, b)$ and $\tilde{V}$ can then be calculated by $\text{klu}(LU, I_{cs} - Y_{v→c} V_{vs})$. It is hereby possible to determine the Thévenin voltages using only the factorization of $Y_{cs}$ and the Thévenin impedances. This means, that the entire coefficient matrix $K$ is no longer needed, which will simplify the algorithm.

The Thévenin impedances for $cs$ and $vs$ nodes are defined in (11). The diagonal of $Z_{cs}$ and the diagonal of the Schur complement $Y_{eq}$ is needed in these computations.

The diagonal of $Z_{ws}$ determines the Thévenin impedances for $cs$ nodes and is computed by taking $U_{Z_{ws}} L_{1}^{-1}$ and $L_{Z_{ws}} = U_{Z_{ws}}^{-1}$ and multiplying the rows and columns, that result in the diagonal i.e.

$$Z_{th,cs,k} = L_{Z_{ws}} (k,:) U_{Z_{ws}} (:,k) \quad \forall k \in cs$$

(22)
where \( L_{Z_{cs}}(k,:) \) is the \( k \)’th row of \( L_{Z_{cs}} \) and \( U_{Z_{cs}}(\cdot,:) \) is the \( k \)’th column of \( U_{Z_{cs}} \).

The Thévenin impedances for the \( vs \) nodes are given as the inverse of the diagonal elements of the Schur complement \( Y_{eq} \).

\[
Z_{th,vs,k} = Y_{eq}(k,k)^{-1} \quad \forall k \in vs,
\]

which is scalar inversion.

Using (9) this can be determined by

\[
Y_{eq}(k,k) = Y_{vs}(k,k) - Y_{c \rightarrow v}(k,:)Z_{cs}Y_{v \rightarrow c}(\cdot,:)k
\]

As with the Thévenin voltages block back substitution is used to determine part of the equation. The Thévenin impedances for the \( vs \) nodes are found as

\[
\hat{U}(\cdot,k) = Z_{cs}Y_{v \rightarrow c}(\cdot,:)k \leftarrow \text{klu}(LU,Y_{v \rightarrow c}(\cdot,:)k)
\]

\[
Y_{eq}(k,k) = Y_{vs}(k,k) - Y_{c \rightarrow v}(k,:)\hat{U}(\cdot,k)
\]

\[
Z_{th,vs,k} = Y_{eq}(k,k)^{-1}
\]

The computations for the \( vs \) nodes is similar to the computations given in [10], and the method will therefore be called a factor-solve method. The method factorize part of the system and then solves for part of the equations to efficiently compute the solution. This approach is used to find the Thévenin impedances for the \( vs \) nodes and to find the Thévenin voltages for the entire system.

Algorithm 2 factorize \( Y_{cs} \) and compute the Thévenin impedances possibly in parallel.

**Algorithm 2** Thévenin equivalents

\[
L_{cs}, U_{cs} \leftarrow \text{factorization of } Y_{cs} \{\text{Output: } LU\}
\]

\[
L_{Z_{cs}} \leftarrow \text{solve}(L_{cs}, I)
\]

\[
L_{Z_{cs}} \leftarrow \text{solve}(U_{cs}^T, I)^T
\]

for \( k = 1,|cs| \) \{In parallel\} do

\[
Z_{th,cs,k} \leftarrow L_{Z_{cs}}(k,:)U_{Z_{cs}}(\cdot,:)k
\]

end for

for \( k = 1,|vs| \) \{In parallel\} do

\[
\hat{U}(\cdot,k) \leftarrow \text{klu}(LU,Y_{v \rightarrow c}(\cdot,:)k)
\]

\[
Y_{eq}(k,k) \leftarrow Y_{vs}(k,k) - Y_{c \rightarrow v}(k,:)\hat{U}(\cdot,k)
\]

\[
Z_{th,vs,k} \leftarrow Y_{eq}(k,k)^{-1}
\]

end for

\[
Z_{th} = \begin{bmatrix} Z_{th,cs} \\ Z_{th,vs} \end{bmatrix}
\]

return \( Z_{th} \) and \( LU \)

This way no computation time is spent on the coefficients and furthermore the computation of the Thévenin voltages is altered from matrix vector multiplication with the dense coefficient matrix to a block back substitution and matrix vector multiplications with sparse matrices.

### A. Complexity of factor-solve method

The complexity of Algorithm 2 is split in to a sequential and a parallel part. The first part is sequential and will be \( O(|cs|^2) \), since the inversions are \( O(|L_{cs}| |cs|) \) and \( O(|U_{cs}| |cs|) \) respectively and the fill-in scales linearly with system size. The factorization has negligible runtime compared to this due to the close to linear complexity.

The two loops run sequential will be \( O(|cs|^2) \) and \( O(|cs| |vs|) \), due to the linear complexity of the computations in the loops. When run in parallel the runtime is theoretically determined by the number of cores. By Amdahl’s law the total runtime will only be limited by the sequential part of the algorithm, since an unlimited number of cores can be added to completely parallelize the loops. In practise however there will be an overhead due to communication between the cores.

For the computation of the Thévenin voltages the complexity will be \( O(|cs|) \). The matrix-vector multiplication is linear due to the sparsity of the matrices scaling with system size, and the block back substitution of KLU is close to linear. This complexity is lower than the reference method’s complexity of \( O(|K|) \), which is almost quadratic to system size for the largest systems.

### IV. Implementation and Test of Factor-Solve Method

The factor-solve method is implemented in MATLAB to evaluate the method with respect to runtime of both Algorithm 2 and the computation of Thévenin voltages as well as the accuracy of the results and the memory requirements. The method is tested on Intel(R) Xeon(R) CPU E5-2650 v4 @ 2.20GHz. The test systems were given earlier in Table I.

The resulting runtime for Algorithm 2 and the runtime for computing the Thévenin voltages (19)-(21) is shown in Table IV, while Fig. 2 shows a plot of the runtime.

**TABLE IV**

<table>
<thead>
<tr>
<th>Case</th>
<th>Runtime (s)</th>
<th>Speed-up Algorithm 2</th>
<th>Speed-up (ms) V(_\text{th})</th>
<th>Speed-up V(_\text{th})</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nordic32</td>
<td>0.045</td>
<td>0.01</td>
<td>0.084</td>
<td>0.15</td>
</tr>
<tr>
<td>Pegase89</td>
<td>0.050</td>
<td>0.03</td>
<td>0.101</td>
<td>0.31</td>
</tr>
<tr>
<td>Pegase1354</td>
<td>0.114</td>
<td>0.97</td>
<td>0.295</td>
<td>9.28</td>
</tr>
<tr>
<td>PTI-WECC-1648</td>
<td>0.544</td>
<td>0.37</td>
<td>0.731</td>
<td>6.32</td>
</tr>
<tr>
<td>Polish-Winter99</td>
<td>0.241</td>
<td>1.23</td>
<td>0.457</td>
<td>15.50</td>
</tr>
<tr>
<td>Polish-Winter03</td>
<td>0.319</td>
<td>0.98</td>
<td>0.540</td>
<td>15.72</td>
</tr>
<tr>
<td>Pegase2869</td>
<td>0.355</td>
<td>1.15</td>
<td>0.604</td>
<td>12.66</td>
</tr>
<tr>
<td>Polish-Winter07</td>
<td>0.300</td>
<td>1.39</td>
<td>0.565</td>
<td>18.60</td>
</tr>
<tr>
<td>PTI-EECC-7991</td>
<td>1.745</td>
<td>3.62</td>
<td>1.473</td>
<td>80.18</td>
</tr>
<tr>
<td>Pegase9241</td>
<td>2.983</td>
<td>3.88</td>
<td>1.708</td>
<td>69.63</td>
</tr>
<tr>
<td>Pegase13659</td>
<td>6.276</td>
<td>5.85</td>
<td>2.328</td>
<td>227.75</td>
</tr>
<tr>
<td>EECC-PSS-33-0</td>
<td>20.592</td>
<td>12.29</td>
<td>5.522</td>
<td>456.28</td>
</tr>
</tbody>
</table>

Algorithm 2 is seen to have close to quadratic complexity as analysed earlier, while the Thévenin voltages is seen to have an almost linear complexity as analysed earlier. The change in complexity for the Thévenin voltages result in a significant decrease in runtime compared to the reference method. It is notable that, all test systems have runtimes for computing Thévenin voltages below 6 ms. The system PTI-WECC-1648 has a runtime that is considerably different compared to the other systems. The complexity is only close to linear and actually depends on the fill-in in the factorization, which can differ depending on the structure of the system.

The speed-up is calculated as a quantity for the performance of the factor-solve method compared to the reference method. This is defined as \( \frac{t_2}{t_1} \), where \( t_1 \) is the runtime of the reference
method and \( t_2 \) is the runtime of the factor-solve method. The speed-up is shown alongside the runtimes in Table IV.

For the smaller systems neither the algorithm nor the calculations of Thévenin voltage receives a speed-up. Systems larger than 1000 buses is sped up in Thévenin voltage computations but only some benefit from Algorithm 2. Algorithm 2 requires large power systems to be faster than Algorithm 1.

Some systems benefit more from computing Thévenin voltages with the factor-solve method instead of the reference method. The systems Pegase2869 and Pegase9241 both have a lower speed-up than test systems of similar size. The reason for this is found in Table II. The coefficient matrix for both Pegase2869 and Pegase9241 is less dense than for the systems with similar size, and therefore weren’t as slow with the reference method. The complexity has changed from being dependent on the number of non-zeros in the coefficients to be close to linearly dependent on system size, which now gives runtimes, that scale better with system size.

Errors in Thévenin voltages obtained with the two methods can be stated in terms of a total vector error (TVE) \([21]\) using runtimes, that scale better with system size.

\[
\text{TVE} (\%) = \sqrt{\frac{(\hat{X}_t - X_t)^2 + (\hat{X}_i - X_i)^2}{X_t^2 + X_i^2}}, \quad 100\% \quad (28)
\]

where \( \hat{X} \) is the estimate (reference or factor-solve method) and \( X \) is the true value (reference method with UMFPACK).

The maximum TVE can be seen in Table V. The two methods only differs by a small margin in accuracy of the resulting Thévenin voltages.

A benefit from the factor-solve method is the amount of memory needed. The reference method need to store the Thévenin impedances along with the coefficient matrix \( K \), while the factor-solve method need to store the Thévenin impedances and the sparse factorization of \( Y_{cs} \). Especially for the larger systems there is a significant reduction in memory using the factor-solve method, since the coefficient matrix is dense. Sparse matrices store the location and value of a non-zero entry, while full matrices store all entries of a matrix. The test systems with a coefficient matrix with a high density, see Table II, will therefore be more efficiently stored as a full matrix than a sparse.

The KLU factorization of a matrix \( A \) is defined as

\[
PRAQ = LU + F \quad (29)
\]

\( P, Q \) are permutations stored as vectors, \( R \) is a scaling matrix optimally stored as a vector, \( L, U \) are complex sparse matrices and \( F = 0 \) in this context \([14]\).

The memory requirements for storing the the coefficient matrix (either in full or sparse format) compared to storing the sparse factorization can be seen in Table VI. Integers like doubles are stored using 64 bits.
This difference is due to the scaling of the memory. The memory for storing \( K \) as a sparse matrix is scaling close to quadratic with system size, while the memory for the sparse factorization of \( Y_{cs} \) is scaling linearly.

### A. Parallelization of Algorithm 2

The runtime for computing Thévenin voltages has been decreased significantly, however Algorithm 2 is only considerably faster for the larger systems. A benefit from the factor-solve method is that Algorithm 2 can easily be parallelized. The runtime is therefore tested on a machine with 2 CPUs of Intel(R) Xeon(R) CPU E5-2650 v4 @ 2.20GHz with 12 cores each. The algorithm will be tested on the following number of cores \( 1, 2, 4, 6, \ldots, 22, 24 \).

Fig. 3a shows the speed-up of Algorithm 2 compared to Algorithm 1 for different number of cores for each test system and Fig. 3b shows the same with the 4 largest systems excluded.

Parallelization decrease the runtime of Algorithm 2 considerably. As system size grows the speed-up increases significantly up to a factor of 90 for the largest system compared to a factor 12 without parallelization. The smallest systems still has no speed-up, since these systems already have a short runtime due to their limited size, and the introduction of loops and overhead time only slows them down. There is a range of test systems that when run sequential did not benefit from Algorithm 2, however when parallelized there is now a benefit. Fig. 3a and 3b furthermore show, that the optimal number of cores increase with system size. After the point of maximal speed-up it decreases due to the increased overhead time, which will be larger than the gain of adding additional cores.

The for loops in Algorithm 2 are implemented with the function \texttt{parfor} in Matlab for all cores. This will for 1 core i.e. the sequential version be a little slower than using \texttt{for}, since there is a small penalty when using \texttt{parfor}. The sequential version could therefore be a little faster.

Table VII shows the number of cores that maximize the speed-up for Algorithm 2 and thereby also minimize the runtime. It shows explicitly that the optimal number of cores increase with systems size and so does the speed-up. Table VII show a runtime of 2.8s for EECC-PSSE-33-0 with the parallelized factor-solve method compared to 253s for the reference method. This means that the factor-solve method will be able to respond faster to a sudden change in the system topology than the reference method.

**Table VII**

<table>
<thead>
<tr>
<th>Case</th>
<th>Optimal no. of cores</th>
<th>Runtime (s) Algorithm 2</th>
<th>Speed-up Algorithm 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nordic32</td>
<td>1</td>
<td>0.045</td>
<td>0.007</td>
</tr>
<tr>
<td>Pegase89</td>
<td>1</td>
<td>0.050</td>
<td>0.027</td>
</tr>
<tr>
<td>Pegase1354</td>
<td>2</td>
<td>0.101</td>
<td>1.105</td>
</tr>
<tr>
<td>PTI-WECC-1648</td>
<td>4</td>
<td>0.281</td>
<td>0.727</td>
</tr>
<tr>
<td>Polish-Winter99</td>
<td>6</td>
<td>0.121</td>
<td>2.439</td>
</tr>
<tr>
<td>Polish-Winter03</td>
<td>10</td>
<td>0.155</td>
<td>2.008</td>
</tr>
<tr>
<td>Pegase269</td>
<td>6</td>
<td>0.164</td>
<td>2.477</td>
</tr>
<tr>
<td>Polish-Winter07</td>
<td>8</td>
<td>0.136</td>
<td>3.070</td>
</tr>
<tr>
<td>PTI-EECC-7991</td>
<td>16</td>
<td>0.344</td>
<td>18.354</td>
</tr>
<tr>
<td>Pegase9241</td>
<td>14</td>
<td>0.497</td>
<td>23.289</td>
</tr>
<tr>
<td>Pegase13659</td>
<td>16</td>
<td>0.901</td>
<td>40.750</td>
</tr>
<tr>
<td>EECC-PSSE-33-0</td>
<td>16</td>
<td>2.759</td>
<td>91.715</td>
</tr>
</tbody>
</table>

Fig. 4 show the distribution of runtime on each part of Algorithm 2 for the optimal number of cores given in Table VII. Factorization time is negligible as expected by its almost linear complexity compared to the quadratic complexity of the entire algorithm as for the reference method [14].

The majority of time is spent on computing the Thévenin impedances for the \( cs \) nodes. However, there is also a significantly larger number of \( cs \) nodes than \( vs \) nodes as seen in Table I. Dividing the runtimes by the number of \( cs \) and \( vs \) nodes respectively, gives an average runtime for \( cs \) nodes that is lower than for the \( vs \) nodes in every case. The reason might be found in the difference in the computations. However, it might also be that the optimal number of cores for the
The smallest systems does not benefit from the factor-solve method, since these are so small that the time spend on matrix multiplications is negligible and changing these computations in to loops only worsen the runtime due to the overhead.

A few systems benefit from computing Thévenin voltages by use of the factor-solve method without benefiting from Algorithm 2 even with parallelization. An alternative method for these system would be to combine the two methods. The Thévenin impedances along with the factorization could be computed by line 1-8 in Algorithm 1 and then the Thévenin voltages would be computed by the factor-solve method. These systems will then get the lowest possible runtime, and furthermore only running line 1-8 of Algorithm 1 will also result in a further decrease in the runtime of computing the Thévenin impedances.

This sort of hybrid method would be useable for systems consisting of between 1.000 and 2.000 busses. It should however be noted that the runtime for both methods for these systems is low enough for real-time computations and either one would be suitable. The method can be used for security assessment in for example the Thévenin equivalent static contingency assessment method [7]. Here Thévenin impedances is determined for all N-1 contingencies and Thévenin voltages are then computed several times when determining the steady-state nodal voltage. For contingency analysis it is important to use the fastest combination to ensure that assessment can be done in reasonable time.

An idea for future work would be to investigate the potential use of GPUs instead of doing all the computations on the CPU. Moving data to and from the GPU is expensive, but when the data is there the computations can be executed and parallelized more efficiently. It could potentially be used on the loops in Algorithm 2. However, since there is no reuse of data in the loops it might not give a better performance, since a GPU excel when doing the same computations multiple times. Furthermore, it would also be satisfactory if the sequential part of the algorithm could have it’s complexity reduced in order to scale better or be changed to be able to run in parallel.

In the complexity analysis the runtime was determined to be dependent on the sparsity of the factors in the factorization, which is almost linear to the system size. From the results it can be seen that this is the case for most of the test systems, however the structure of some systems give rise to a larger fill-in and will affect the resulting runtime. It would be interesting to find specific reasons for the behaviour of these system.

A clear benefit from the factor-solve method is the decrease in memory usage, where a reduction in required memory was seen for all test systems. This together with the new linear complexity of the Thévenin voltages computations will make the method use fewer computational resources and thereby leave room for other assessment methods.

Future work would also be to include the factor-solve method in stability assessment methods to test their performance with the new calculations. It would then be possible to analyse the these methods and determine new areas suitable for optimization. It would furthermore be interesting to test the method in a real-time setting on a SW-platform like [24].
VI. Conclusion

The paper describes a reference method for determining the Thévenin equivalents for all nodes in a power system. The reference method was analysed to be insufficient especially for large power systems. The given complexity of the method is dependent on the density of the coefficients and will therefore be less viable for some systems compared to others due to the structure of the power system.

A factor-solve method was introduced, which takes advantage of the block back substitution in KLU. The method spends no computation time on generating coefficients and the calculations of Thévenin voltages computation can be done in linear time. Furthermore, the memory usage is significantly lower than for the reference method changing from gigabytes to a couple of megabytes for larger systems.

Computations of Thévenin impedances can take advantage of parallelization and runtime will therefore be dependent on the number of cores used. The optimal number of cores is shown to increase with system size. The runtime for determining Thévenin impedances is still not satisfying for the largest system, however these will only be determined, when system topology change. Thévenin voltages are determined without relying on parallization and the linear scaling with system size, enables the factor-solve method to have considerably lower computation time than the reference method.

REFERENCES


