Improved Evaluation of Large Network Matrices for Linear Power Flow Within Optimization Problems

Preprint

Alessandro Francesco Castelli and Jose Daniel Lara

National Renewable Energy Laboratory

Presented at the 2024 IEEE Power & Energy Society General Meeting
Seattle, Washington
July 21-25, 2024

NREL is a national laboratory of the U.S. Department of Energy
Office of Energy Efficiency & Renewable Energy
Operated by the Alliance for Sustainable Energy, LLC

This report is available at no cost from the National Renewable Energy Laboratory (NREL) at www.nrel.gov/publications.

Contract No. DE-AC36-08GO28308

Conference Paper
NREL/CP-6A40-88141
March 2024

Improved Evaluation of Large Network Matrices for Linear Power Flow Within Optimization Problems

Preprint

Alessandro Francesco Castelli and Jose Daniel Lara

National Renewable Energy Laboratory

Suggested Citation
Castelli, Alessandro Francesco and Jose Daniel Lara. 2024. *Improved Evaluation of Large Network Matrices for Linear Power Flow Within Optimization Problems: Preprint*. Golden, CO: National Renewable Energy Laboratory. NREL/CP-6A40-88141. https://www.nrel.gov/docs/fy24osti/88141.pdf.

NREL is a national laboratory of the U.S. Department of Energy
Office of Energy Efficiency & Renewable Energy
Operated by the Alliance for Sustainable Energy, LLC

This report is available at no cost from the National Renewable Energy Laboratory (NREL) at www.nrel.gov/publications.

Contract No. DE-AC36-08GO28308

Conference Paper
NREL/CP-6A40-88141
March 2024

National Renewable Energy Laboratory
15013 Denver West Parkway
Golden, CO 80401
303-275-3000 • www.nrel.gov

NOTICE

This work was authored by the National Renewable Energy Laboratory, operated by Alliance for Sustainable Energy, LLC, for the U.S. Department of Energy (DOE) under Contract No. DE-AC36-08GO28308. Funding provided by the U.S. Department of Energy Office of Electricity Delivery and Energy Reliability. The views expressed herein do not necessarily represent the views of the DOE or the U.S. Government. The U.S. Government retains and the publisher, by accepting the article for publication, acknowledges that the U.S. Government retains a nonexclusive, paid-up, irrevocable, worldwide license to publish or reproduce the published form of this work, or allow others to do so, for U.S. Government purposes.

Cover Photos by Dennis Schroeder: (clockwise, left to right) NREL 51934, NREL 45897, NREL 42160, NREL 45891, NREL 48097, NREL 46526.

Improved evaluation of large network matrices for linear power flow within optimization problems

Alessandro Francesco Castelli
National Renewable Energy Laboratory
Golden, Colorado, USA
alessandrofrancesco.castelli@nrel.gov

José Daniel Lara
National Renewable Energy Laboratory
Golden, Colorado, USA
josedaniel.lara@nrel.gov

***Abstract*—This work discusses methods for evaluating the Power Transfer Distribution Factor (PTDF) and Line Outage Distribution Factor (LODF) matrices by employing sparse linear algebra for large-scale computing applications. These matrices are critical in many power systems applications, such as the Unit Commitment Problem (UC), pre- and post-contingency power flow analysis, and transmission expansion. These matrices are typically dense, which means they require a significant amount of time and memory to be computed for large networks. However, by analyzing the structure of the matrices and their computation method, it is possible to use reduced memory methods based on sparse matrix operations. This paper shows that sparse linear algebra algorithms are faster and require less memory and time than traditional dense approaches. Additionally, we explore the effect of matrix sparsification by eliminating trailing digits on power flow calculations.**

***Index Terms*—Power Flow, Sparse Linear Algebra, Large Scale**

I. INTRODUCTION

Evaluating power flows in bulk power systems is critical for multiple power systems computational problems like the Unit Commitment (UC) Problem and its Security Constrained version (SCUC). This operational model must capture generation and transmission operational constraints while guaranteeing an optimal solution. Although many contributions try to incorporate AC power flow into operational computational problems (e.g., [1], [2], [3]), the difficulties of its non-linear representation have lead to the use of the linearized (or "DC") version using the so-called PTDF and LODF matrices. This version does not guarantee a feasible solution of the non-linear counterpart [4] (i.e., it does not constitute a relaxation of the AC power flow problem), but it is commonly adopted by Independent Systems Operators (ISOs) to reduce the computational requirements of solving optimization models (see ref. [5] for more). However, solving the DC SCUC problem for large systems (e.g., more than 50,000 buses) is still challenging and requires great computing power. This is particularly demanding when forecast uncertainty related to non-dispatchable generation and load demands require the adoption of stochastic variants of the SCUC problem or updating the generator's schedule more often (e.g., every 10 minutes). As a result, there is significant value in decreasing the overall computational effort involved in evaluating and using the PTDF and LODF network matrices due to their impact on the creation of the optimization model and its solution (mainly when these tasks are performed on a single HPC node with limited memory). A frequently used approach to decrease the computational burden of incorporating network flows in optimization problems is rounding to remove entries with a relatively low impact on the overall flow calculations. However, while this is a common practice in the industry, there hasn't been a systematic approach to evaluating the effects of eliminating trailing zeros.

This paper focuses on improving the evaluation of large network matrices and offers three main contributions:

- We present detailed algorithm descriptions and techniques that employ computational sparse linear algebra for the calculation of PTDF and LODF matrices with reduced computational cost,
- An analysis of the performance that the choice of sparse linear solver choices has in compute time and memory usage and configuration alternatives that impact solve performance,
- A computational study on the approximation error introduced by removing trailing digits to "sparsify" the PTDF matrix to reduce the memory footprint of large power flow problems.

The techniques and analysis presented in this paper have been implemented in the open-source Julia package `PowerNetworkMatrices.jl`[1].

II. LINEARIZED POWER FLOW AND NETWORK MATRICES

The linearized representation assumes that given an electrical network with a set of nodes $\mathcal{N}$ of size n and a set of AC-Branches $\mathcal{L}$ of size l, the steady state active power flow $f_{i,j}$ through a line connecting nodes $\{i,j\} \in \mathcal{N}$ can be computed by the equation

$$f_{i,j} = \frac{\theta_i - \theta_j}{x_{i,j}} \tag{1}$$

where θ corresponds to the "voltage angle" at each bus and $x_{i,j}$ to the reactance of the line connecting the buses. Furthermore, by considering that active power must be balanced at each node:

$$p_i = \sum_{j \in \mathcal{N}} f_{i,j} \quad \forall i \in \mathcal{N} \tag{2}$$

[1] https://github.com/NREL-Sienna/PowerNetworkMatrices.jl

$$\begin{bmatrix} \boxed{b_1} & & & \\ & b_2 & & \\ & & b_3 & \\ & & & b_4 \end{bmatrix} \begin{bmatrix} \boxed{1} & \boxed{-1} & 0 & 0 \\ 0 & 1 & -1 & 0 \\ 0 & 1 & 0 & -1 \\ 0 & 0 & 1 & -1 \end{bmatrix} =$$

$$\begin{bmatrix} \boxed{b_1} & \boxed{-b_1} & 0 & 0 \\ 0 & b_2 & -b_2 & 0 \\ 0 & b_3 & 0 & -b_3 \\ 0 & 0 & b_4 & -b_4 \end{bmatrix}$$

Fig. 1. Evaluation and structure of the $\mathbf{BA}$ Matrix. Note that every row is obtained by simply multiplying each diagonal element of $\mathbf{B}$ for the elements of the respective row in $\mathbf{A}$.

It is possible to relate the angle in each node with the power injection as follows:

$$\boldsymbol{p} = (\mathbf{A}^T\mathbf{BA})\boldsymbol{\theta} \tag{3}$$

where $\mathbf{A}$ is the $l \times n$ Incidence Matrix, defining the start and end points of each line using values 1 and -1, respectively. The $\mathbf{B}$ Matrix is a $l \times l$ diagonal matrix containing the impedance values of the system AC Branches. Then, the vector of the power flows on each line $\boldsymbol{f}$ from the angle values vector can be evaluated as follows:

$$\boldsymbol{f} = (\mathbf{BA})\boldsymbol{\theta} \tag{4}$$

By combining Eqs. (3) and (4), the $\mathbf{PTDF}$ matrix can be evaluated as follows [6], [7]:

$$\boldsymbol{f} = (\mathbf{BA})(\mathbf{A}^T\mathbf{BA})^{-1}\boldsymbol{p} = \mathbf{PTDF}\ \boldsymbol{p} \tag{5}$$

This matrix relates the steady-state line flows $\boldsymbol{f}$ with the power injections $\boldsymbol{p}$ such that each entry $\mathbf{PTDF}[k, i]$ represents the increase in power flow on line f_k as a consequence of a change in power at node i under the assumption (2). Once the $\mathbf{PTDF}$ is defined, the $\mathbf{LODF}$ matrix can be calculated. This matrix is used to estimate the sensitivity factors related to the network flows between branches in steady state. The common use case is the calculation of the flows diverted on all the other branches in $\mathcal{L}$ resulting from the trip of a specific like k. The $l \times l$ $\mathbf{LODF}$ Matrix is calculated as follows [8]:

$$\mathbf{LODF} = \mathbf{PTDF_A} \cdot \mathbf{M} \tag{6}$$

with $\mathbf{PTDF_A} = \mathbf{PTDF} \cdot \mathbf{A}^T$ and $\mathbf{M} = (\mathbf{I} - \mathbf{PTDF}^{\mathbf{d}}_{\mathbf{A}})^{-1}$. In particular, the $\mathbf{PTDF}^{\mathbf{d}}_{\mathbf{A}}$ features only the diagonal elements of $\mathbf{PTDF_A}$. Please note that as last step all diagonal elements of $\mathbf{LODF}$ are set to -1.

III. Improved evaluation of the network matrices

A. Evaluation of the PTDF matrix

To improve the computational efficiency of the $\mathbf{PTDF}$ Matrix evaluation, a thorough analysis on each computational steps must be performed. In particular, the use of sparse matrix operations is maximized while trying to minimize memory allocation. Starting from the the $\mathbf{A}$ Matrix, it can be noted how it can be defined and stored in a sparse fashion, while the $\mathbf{B}$ Matrix does not need to be explicitly evaluated. Thus, given $\mathbf{A}$ and the vector $\boldsymbol{b}$ (containing the diagonal elements of $\mathbf{B}$), the product $\mathbf{BA}$ can be evaluated by a simple `for` loop, as shown in Algorithm 1. In this way the evaluation can be performed in way less operations than full matrix multiplication. At first, the $\mathbf{A}$ Matrix and the structure $non\_zero\_positions$ must be evaluated and saved. This structure stores the column positions of the two non-zero elements of $\mathbf{A}$ per each row.

Algorithm 1 *Evaluation of the* $\mathbf{BA^T}$ *Matrix.*

Require: `BA_I, BA_J` *and* `BA_V` $\leftarrow$ *empty list*
  for $i := 1 : l$ **do** ▷ iterate over each line
    $(i, j) \leftarrow non\_zero\_positions(\mathbf{A}[i, :])$
    $b \leftarrow$ $line\_impedence(l)$
    Append i *to* `BA_I`, l *to* `BA_J`, b *to* `BA_V`
    Append j *to* `BA_I`, l *to* `BA_J`, $-b$ *to* `BA_V`
  end for
  $\mathbf{BA^T} \leftarrow$ $sparse$(`BA_I, BA_J, BA_V`) ▷ Save sparse matrix

Then, the lists `BA_I`, `BA_J` and `BA_V` are initialized as empty. They collect the row, column indices, and values of the non-zero matrix elements respectively. Therefore, the product $\mathbf{BA}$ can be defined as a sparse Matrix, significantly reducing the memory requirements. Algorithm 1 represents in a programmatic way the example shown in Figure 1, with the only difference that the $(\mathbf{BA})^T$ is saved rather than the original one. The reason of this choice derives by analyzing (5). By transposing the entire equation, the evaluation of $\mathbf{PTDF^T}$ does not require the explicit computation of $(\mathbf{A}^T\mathbf{BA})^{-1}$ but simply to solve the resulting linear system ($\mathbf{A}^T\mathbf{BA}$ is symmetric). To do so matrix factorization can be considered, and among the different methods available `LU` factorization was chosen since it consistently showed to be the most effective. The overall algorithmic procedure is described in Algorithm 2, which allows an efficient computation of $\mathbf{PTDF^T}$.

Algorithm 2 *Evaluation of the* $\mathbf{PTDF^T}$ *Matrix.*

Require: $(\mathbf{BA})^T$, $\mathbf{A}$
Require: i^{ref_bus} ▷ column index of reference bus
  $\mathbf{A^TBA} \leftarrow (\mathbf{BA})^T\mathbf{A}$
  $(\mathbf{L}, \mathbf{U}) \leftarrow lu(\mathbf{A^TBA})$ ▷ get L, U factorization matrices
  $\mathbf{PTDF^T} \leftarrow solve(\mathbf{L}, \mathbf{U}, (\mathbf{BA})^T)$
  $\mathbf{PTDF^T}[i^{ref\_bus}, :] := 0$ ▷ setting to zero the column related to the reference bus

B. Evaluation of the LODF matrix

By analyzing (6) it can be seen how the evaluation of the $\mathbf{LODF}$ Matrix requires to compute the $\mathbf{PTDF}$ one and $(\mathbf{I} - \mathbf{PTDF}^{\mathbf{d}}_{\mathbf{A}})^{-1}$. A more efficient approach is to compute the $\mathbf{LODF^T}$ matrix, by transposing Eq. (6). In this way, direct inversion on $\mathbf{I} - \mathbf{PTDF}^{\mathbf{d}}_{\mathbf{A}})$ can be avoided and the

following linear system can be solved with matrix factorization techniques.

$$\mathbf{LODF^T} = (\mathbf{I} - \mathbf{PTDF_A^d})^{-1}\mathbf{PTDF_A^T} \tag{7}$$

The whole procedure is described in Algorithm 3.

Algorithm 3 *Evaluation of the* $\mathbf{LODF^T}$ *Matrix.*

Require: $\mathbf{PTDF^T}$, $\mathbf{A}$ ▷ use Algorithm 2 for $\mathbf{PTDF^T}$
Require: $\mathbf{m\_I}$, $\mathbf{m\_V} \leftarrow empty_list$
  $\mathbf{PTDF_A^T} \leftarrow \mathbf{APTDF^T}$
  for $i := 1 : l$ **do** ▷ iterate over each line
    if 1-$\mathbf{PTDF^T}[l,\ l] \leq 0$ **then**
      Append l *to* `m_I`, *Append* 1 *to* `m_V`
    else
      Append l *to* `m_I`
      Append $(1 - \mathbf{PTDF_A^T}[l,\ l])$ *to* `m_V`
    end if
  end for
  $(\mathbf{I} - \mathbf{PTDF_A^d}) \leftarrow sparse(\mathbf{m\_I}, \mathbf{m\_I}, \mathbf{m\_V})$
  $(\mathbf{L},\ \mathbf{U}) \leftarrow lu(\mathbf{I} - \mathbf{PTDF_A^d})$
  $\mathbf{LODF^T} \leftarrow solve(\mathbf{L},\ \mathbf{U},\ \mathbf{PTDF_A^T})$
  for $i := 1 : l$ **do**
    $\mathbf{LODF^T}[l,\ l] := -1$
  end for

C. Matrix sparsification

With the aim of decreasing the memory required to store the network matrices, they can be made more sparse by dropping elements whose absolute values is below a certain threshold. Despite good results are easily achievable by setting to zero elements of the matrix, a careful assessment on the error induced by the loss of information on the power flows must be performed. This error can be evaluated as the difference between $\boldsymbol{f}$ and $\boldsymbol{f^{sp}}$, both evaluated through Eq. (5) by considering $\mathbf{PTDF}$ and $\mathbf{PTDF^{sp}}$ (sparsified version) respectively.

IV. Test cases and linear solve libraries

The proposed methodology requires to solve linear systems for the evaluation of the $\mathbf{PTDF^T}$ and $\mathbf{LODF^T}$, and therefore different libraries can be used. In this work four different open-source linear system solvers were considered and compared: `KLU` [9], `OpenBLAS`, `MKLBLAS` and `MKLPardiso` [10]. In this work both `MKLPardiso`, `MKLBLAS` and `OpenBLAS` are set to use the maximum number of threads available. The proposed methodology was tested over 48 systems from the PGLib library [11] and from Texas A&M synthetic library systems. These are all those systems with a number of buses/branches between 500/686 and 82000/104121. Six tolerances between $10e^{-6}$ and $10e^{-1}$ ($10e^{1}$ increments) were considered for the evaluation of the error induced by matrix sparsification on power flows ($10e^{-15}$ considered as reference). Computation of these was done by using the open-source Julia package `PowerFlows.jl`[2]. Finally, all the numerical tests were executed on a workstation featuring 18 cores (36 threads) 3.0 GHz CPU and 720 GB of RAM.

[2]https://github.com/NREL-Sienna/PowerFlows.jl

PTDF computational time

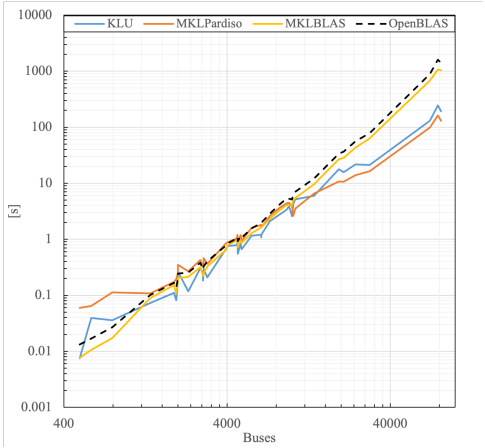

Fig. 2. Computational time required to evaluate the PTDF matrix for the different test cases and solvers, given the number of system buses.

V. Results

A. Computational performance for the evaluation of the PTDF and LODF matrices

The PTDF and LODF matrices were evaluated for each system considered in the library and for each method. Starting from the PTDF matrix, the computational time required for its evaluation can be found in Figure 2. By comparing the different solvers it can be noted how MKLBLAS and KLU are the fastest for those systems with a number of buses ranging between 500 and 4619. In particular, MKLBLAS was the fastest for systems with less than 1000 buses. When looking at medium size systems (from 4661 to 10000 buses), KLU showed to require the least amount of time for evaluating the PTDF matrix. For these system sizes, MKLPariso was up to +214% slower than KLU , while the dense solvers showed to be slower up to a +117%. When considering large systems with more than 10189 buses, MKLPardiso showed to be the fastest solver. Here KLU is between +23% and +40% slower, while MKLBLAS and OpensBLAS resulted significantly slower (between +55% and +635%). The reasons of the difference between KLU and MKLPardiso can be found in how the solvers are designed. MKLPardiso requires the Right Hand Side (RHS) of the linear system to be dense, therefore increasing the computational burden, while KLU accepts sparse matrices on both sides of the equation. This explains the better performance on smaller instances. However, the ability of MKLPardiso to run the calculations in parallel allows it to be faster when larger systems are considered (KLU has no parallel computation option as of today).

When looking at the memory used for the evaluation of the PTDF matrix (Figure 3), it is evident how KLU is significantly less demanding than all other solvers. In particular, MKLPardiso needs between +37% and +50% more memory,

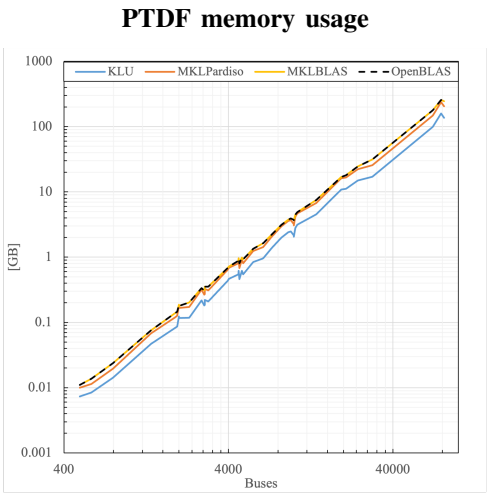

Fig. 3. Memory required during the evaluation of the PTDF matrix for the different test cases and solvers, given the number of system buses.

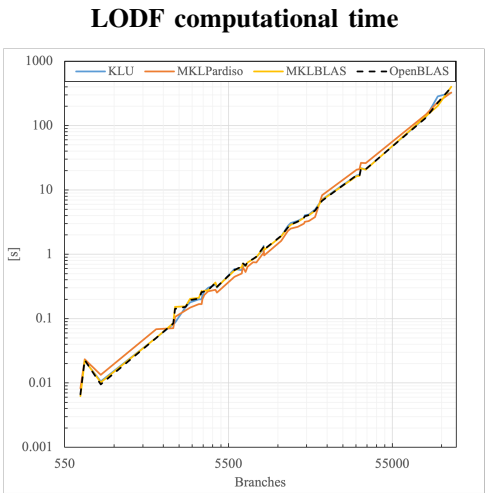

Fig. 4. Computational time required to evaluate the LODF matrix for the different test cases and solvers, given the number of system branches.

while the BLAS-based solvers show an higher memory usage in the range of +51% and +84%. The improved memory performance is in part explained by the fact that KLU can solve linear systems with sparse RHS preventing the allocation of an additional large dense matrix.

When looking at the computation cost of building the LODF, Figure 4 and 5 show the time and memory needed. It is important to note that the figures shown are related to the evaluation of the LODF matrix only (therefore related to the solution of (7)). By looking at Figure 4, similar calculation times are seen across the different solvers. This can be understood by knowing that each method evaluates $\mathbf{PTDF}_{\mathbf{A}}^{\mathbf{T}}$ in the same way, and they differ just for how the linear system in (7) is solved. However, solving this system in (7) is a relatively easy task since in requires the factorization of the diagonal matrix $(\mathbf{I}-\mathbf{PTDF}_{\mathbf{A}}^{\mathbf{d}})$. Finally, KLU, MKLBLAS and OpenBLAS are the equivalently better options for small and large case systems, while for mid-size ones MKLPardiso results to be fasts (although with a limited margin).

Regarding the memory needed for the evaluation of the LODF matrix, Figure 5 shows how the BLAS-based solvers required more memory than MKLPardiso and KLU, which show very similar figures. However, KLU needed less memory for those test cases featuring more than 13659 buses, with MKLPardiso requiring between +57% and +100% more memory. Regarding both MKLBLAS and OpenBLAS, they required between +42% and +50% more memory accross all test cases. When compared with the PTDF matrix, it can be noted how the evaluation of the LODF required less time but more memory (e.g., the largest case - 82000 buses - required 283 seconds sand 136 GB for evaluating the PTDF with KLU, 192 seconds and 173 GB for the LODF). This is related to the less operations needed for solving (4) and (5) (less computational time), that however need to handle the $\mathbf{PTDF}$ matrix for the evaluation of the $\mathbf{PTDF}_{\mathbf{A}}^{\mathbf{T}}$ and $(\mathbf{I}-\mathbf{PTDF}_{\mathbf{A}}^{\mathbf{d}})$ terms (more memory).

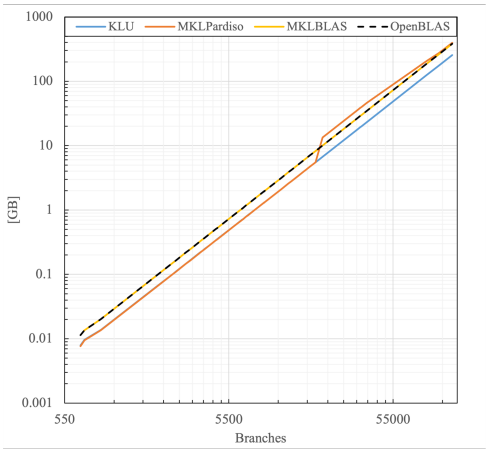

Fig. 5. Memory required to evaluate the LODF matrix for the different test cases and solvers, given the number of system branches.

B. Sparsification of the PTDF matrix

The effects of sparsifying the PTDF matrix are here reported. Three different metrics are shown: the Root Mean Square Error (RMSE) related to the power flows evaluation, the matrix density and memory reduction. The matrix density is defined as the ratio between the number of non-zero values and the total number of elements of the matrix. The memory reduction refers to the percentage drop in memory requirement after sparsification. The relationship between RMSE, sparsity and tolerance can be found in Figure 6. At first, a reference tolerance of 1e-15 was considered as reference since it does not produce any reduction in density and RMSE is practically zero. As the tolerance increases to 1e-5, the density does not change significantly (values keep on being in the 51% and 95% range), while the RMSE increases of some orders of magnitude. For a tolerance larger than 1e-5, the RMSE increases significantly, up to values too high for practical use.

Sparsification tolerance, sparsity and RMSE

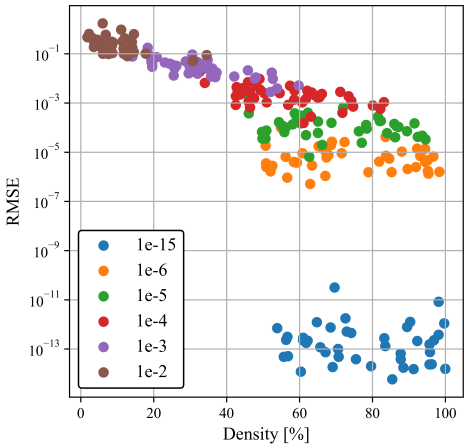

Fig. 6. RMSE as a function of sparsity. The different colors refer to the different sparsification tolerances considered.

Sparsification tolerance, memory reduction and RMSE

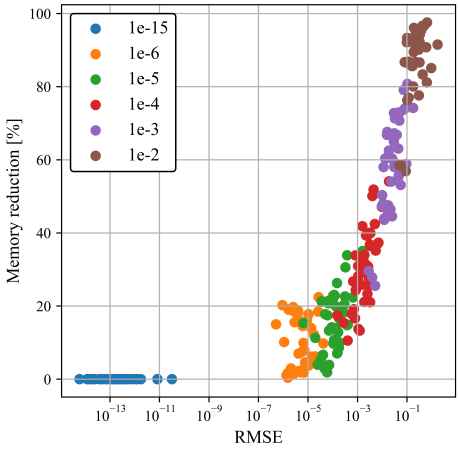

Fig. 7. Memory reduction as a function of RMSE. The different colors refer to the different sparsification tolerances considered.

Finally, the relationship between RMSE and memory reduction can be found by looking at Figure 7. In order to keep an acceptable RMSE, tolerances exceeding 1e-5 should be avoided since the marginal memory reduction is very low compared to the increase in error. Further, the results show the achievable density of the resulting matrices is very variable and system dependent at tolerances below 1e-5. Hence, it is recommended to perform power flow studies to determine the induced RMSE error prior to the implementation of this technique.

VI. Conclusions

- Linear approximations of power flows require dense matrices and for large real-scale system can generate significant increases in the required compute power when used within optimization problems. The time and memory cost of computing these matrices can be reduced by employing linear solvers to perform the inversion operations of the calculation procedures. Further improvements can be achieved in the computation by employing appropriate solvers that exploit sparsity and parallelism.
- The results show that linear solver choice greatly depends on the system size and available memory for the computation. However, for very large scale systems multi-theated sparse libraries such as MKLPardiso provide the best alternative in terms of speed for the computation of the matrices but with a larger memory footprint. On the other hand, although KLU showed slower compute times, it provided significant memory savings which could be relevant if matrix computation is done at the same compute node as the optimization problem solution.
- Sparsification techniques to reduce the memory footprint of network matrices has an exponential effect on the increase of the RMSE. The results show diminishing return in the memory savings with respect to the increased RMSE resulting from larger tolerances, which reflects that aggressive sparsification can hurt result quality regardless of the improved memory footprint. In general, for tolerances that make practical sense, power flow studies should be conducted to determine appropriate sparsification tolerances, this technique's performance is highly dependent on the system size.

References

[1] Y. Fu, M. Shahidehpour, and Z. Li, "Security-constrained unit commitment with ac constraints," *IEEE Transactions on Power Systems*, vol. 20, pp. 1001–1013, 5 2005.

[2] F. Q. Gonzalo E. Constante-Flores, Antonio J. Conejo, "Ac network-constrained unit commitment via relaxation and decomposition," *IEEE Transactions on Power Systems*, vol. 37, no. 3, pp. 2187–2196, 2022.

[3] D. K. Molzahn, I. A. Hiskens *et al.*, "A survey of relaxations and approximations of the power flow equations," *Foundations and Trends® in Electric Energy Systems*, vol. 4, no. 1-2, pp. 1–221, 2019.

[4] K. Baker, "Solutions of dc opf are never ac feasible," in *Proceedings of the Twelfth ACM International Conference on Future Energy Systems*, 2021, pp. 264–268.

[5] A. Papavasiliou, *Optimization Models in Electricity Markets*. Cambridge University Press, 2024.

[6] C. S. Song, C. H. Park, M. Yoon, and G. Jang, "Implementation of ptdfs and lodfs for power system security," *Journal of International Council on Electrical Engineering*, vol. 1, pp. 49–53, 2011.

[7] C. Barbulescu, S. Kilyeni, G. Vuc, B. Lustrea, R.-E. Precup, and S. Preid, "Software tool for power transfer distribution factors (ptdf) computing within the power systems," in *IEEE EUROCON 2009*. IEEE, 2009, pp. 517–524.

[8] J. Guo, Y. Fu, Z. Li, and M. Shahidehpour, "Direct calculation of line outage distribution factors," *IEEE TRANSACTIONS ON POWER SYSTEMS*, vol. 24, p. 1633, 2009. [Online]. Available: http://motor.ece.iit.edu/

[9] T. A. Davis and E. Palamadai Natarajan, "Algorithm 907: Klu, a direct sparse solver for circuit simulation problems," *ACM Trans. Math. Softw.*, vol. 37, no. 3, sep 2010. [Online]. Available: https://doi.org/10.1145/1824801.1824814

[10] O. Schenk and K. Gärtner, "Solving unsymmetric sparse systems of linear equations with pardiso," *Future Generation Computer Systems*, vol. 20, no. 3, pp. 475–487, 2004.

[11] S. Babaeinejadsarookolaee, A. Birchfield, R. D. Christie, C. Coffrin, C. DeMarco, R. Diao, M. Ferris, S. Fliscounakis, S. Greene, R. Huang, C. Josz, R. Korab, B. Lesieutre, J. Maeght, T. W. K. Mak, D. K. Molzahn, T. J. Overbye, P. Panciatici, B. Park, J. Snodgrass, A. Tbaileh, P. V. Hentenryck, and R. Zimmerman, "The power grid library for benchmarking ac optimal power flow algorithms," 2021.