Fast Solutions in Power System Simulation through Coupling with Data-Driven Power Flow Models for Voltage Estimation

Siobhan Powell
Stanford University
siobhan.powell@stanford.edu

Alyona Ivanova
SLAC National Accelerator Lab
aivanova@slac.stanford.edu

David Chassin
SLAC National Accelerator Lab
dchassin@slac.stanford.edu

Abstract

Power systems solvers are vital tools in planning, operating, and optimizing electrical distribution networks. The current generation of solvers employ computationally expensive iterative methods to compute sequential solutions. To accelerate these simulations, this paper proposes a novel method that replaces the physics-based solvers with data-driven models for many steps of the simulation. In this method, computationally inexpensive data-driven models learn from training data generated by the power flow solver and are used to predict system solutions. Clustering is used to build a separate model for each operating mode of the system. Heuristic methods are developed to choose between the model and solver at each step, managing the trade-off between error and speed. For the IEEE 123 bus test system this methodology is shown to reduce simulation time for a typical quasi-steady state time-series simulation by avoiding the solver for 86.7% of test samples, achieving a median prediction error of 0.049%.

1. Introduction

With electricity demand on the rise and increased diversification of generation sources in the field, sophisticated power system modeling tools are required for electrical utility companies to be able to provide a clean, efficient, and reliable network to all customers. One of the challenges with modeling electrical distribution networks is the growing variety of system modes observed due to the emergence of new distributed resource technologies, making it computationally expensive for traditional power system solvers to produce results during long simulations. In many network design and analysis problems thousands of power flow solutions must be obtained in a deterministic time-sequence referred to as "quasi-steady state" to generate data for multiple days of operation or to conduct a sensitivity analysis on different model parameters. Improving the computational performance of quasi-steady state simulations can lead to faster progress in studying and deploying new system control strategies, power system optimization techniques and electricity markets analysis, particularly in the presence of highly variable distributed energy resources.

Among the open source solvers used to solve quasi-steady state power flow problems are GridLAB-D and Pandapower [1, 2, 3]. GridLAB-D in particular has a distinct use-case for power flow solvers, which can be used to compute large scale models that mimic real-world scenarios and aid in decision making processes that address operation, planning and resilience of distribution systems.

GridLAB-D uses a three-phase unbalanced power flow solver which makes a distinction between radial and non-radial distribution networks. GridLAB-D's default method is forward-back sweep [4] for strictly radial systems, while a Newton-Raphson (NR) method is used for non-radial three-phase unbalanced networks. The advantage of the NR method is that it can be applied independent of system-structure and it can provide faster, more reliable results. These advantages come at the cost of increased code complexity and memory usage compared to alternative methods, such as the Gauss-Seidel method [5, 6]. The NR methodology calculates voltage updates and current injections at each node by computing the inverse Jacobian of the entire system. Even though the NR methodology is very robust, for large system continuous time simulations reconstructing the Jacobian whenever the system changes significantly can be computationally expensive, and each additional iteration of the solver heavily prolongs the duration of the overall simulation.

GridLAB-D runs time-series simulations that require very large numbers of sequential power flow solutions for conditions that are often similar to those encountered earlier in the simulation, and very often almost identical to the last solution found. To exploit this GridLAB-D uses the most recent simulation output to seed the next solution and reduce the number

of iterations required. However, Jacobian updates for states that are close to previously solved states may be computationally redundant, which motivates the development of techniques complementary to the built-in methods that can address these situations.

Data-driven approximations of the system should be considered as a potentially faster alternative. The idea of using data-driven models to supplement or replace a slow, physics-based simulation has been studied before in other fields including robotics and fluids [7, 8]. In power system data-driven models have been studied for the estimation of voltage, network state, and network parameters [9, 10, 11].

In this paper we propose the integration of a data-driven power flow model into the NR power flow solution method used by GridLAB-D. We propose to train a data-driven model on the outputs from the power flow solver, a training data set, and then use the model's estimates to avoid running the full NR solver when it is not necessary, i.e., when the new system state input is sufficiently similar to the training set and confidence in the model's prediction is high.

To fully exploit the potential speed of this approach we implement a model based on Linear Regression, which is faster to train and implement than the Support Vector Regression model detailed in [10]. Augmenting the Linear Regression approach by learning a new model for each mode of the system (a cluster of similar states in time) we find the estimation is very good for a large majority of samples. We investigate how different fail-safes can be put in place to make the overall model robust and help the simulation choose reliably between the data-driven model solution and the full solver. This is shown to give a large reduction in the number of full NR solves required without compromising the accuracy of the simulation outputs.

This paper is organized as follows: in Section 2 the model is detailed; in Section 3 the data set and experiments are outlined, including the proposed system for coupling the power flow solver with the model; in Section 4 the results of these experiments are presented; they are discussed in Section 5; in Section 6 we detail ideas for future testing and improvement of the methodology; and in Section 7 we present our conclusions.

2. Methods

A load-driven power flow simulation environment calculates the voltage at each bus or measurement point in the system. Let $\{\mathbf{p}, \mathbf{q}, \mathbf{v}, \mathbf{a}\}$ denote the vectors of real power injection, reactive power injection, voltage magnitude, and voltage phase angle at each point in the system where they are measured. There may be a different number of measurement points for the loads and voltages, so let n_p and n_v denote the lengths of the $\{\mathbf{p}, \mathbf{q}\}$ and $\{\mathbf{v}, \mathbf{a}\}$ vectors respectively.

The mapping being resolved by the solver then is

$$\mathbf{v} = f_1(\mathbf{p}, \mathbf{q}; Y), \qquad (1)$$
$$\mathbf{a} = f_2(\mathbf{p}, \mathbf{q}; Y), \qquad (2)$$

where Y is the admittance matrix for the network which includes connectivity and line parameter information.

A data-driven version of this mapping takes the form

$$\mathbf{v} = \tilde{f}_1(\mathbf{p}, \mathbf{q}), \qquad (3)$$
$$\mathbf{a} = \tilde{f}_2(\mathbf{p}, \mathbf{q}), \qquad (4)$$

where the models $\tilde{f}_1$ and $\tilde{f}_2$ are learned only from historical measurements of $\{\mathbf{p}, \mathbf{q}, \mathbf{v}, \mathbf{a}\}$. We call each measurement a time stamp or sample and denote individual sample measurements with $\mathbf{p}_t$, where t represents time.

2.1. Base Model

In [10] the authors constructed their data-driven model to match the form of the forward power flow equations, so that learning the model parameters represented learning the values in Y. In our experiments we find that constructing the model as a simple Linear Regression gives as good results for the majority of inputs, is simpler to implement in a complex software set-up like GridLAB-D, and is faster to train and apply. For our application the speed is most important and less accurate predictions can be covered by using the real solver. Therefore we formulate $\tilde{f}_1$ and $\tilde{f}_2$ as Linear Regressions

$$\mathbf{v} = A_1 \begin{bmatrix} \mathbf{p} \\ \mathbf{q} \end{bmatrix}, \qquad (5)$$

$$\mathbf{a} = A_2 \begin{bmatrix} \mathbf{p} \\ \mathbf{q} \end{bmatrix}, \qquad (6)$$

where A_1 and A_2 are matrices with dimension $[n_v, n_p]$. Given a time-series set of training data, this mapping is learned using the Linear Regression model in the Python sklearn library which implements ordinary least squares [12].

2.2. Clustering

Testing on a multi-day data set revealed that the system operates in different modes, as the

training and testing prediction errors clearly showed time-dependence throughout each day. Clustering in time is implemented to reflect this data pattern in the model.

The training set is used to sort the time stamps into n_c clusters by their input vectors $[\mathbf{p}, \mathbf{q}]_t^T$. Then one model is trained for each cluster, with each of the n_c clusters becoming a new, smaller training set. Each time stamp in the testing set is sorted into its closest cluster, and the estimate for that time stamp is made using the cluster's model.

Three clustering approaches were considered: first, we considered K-Means clustering for its versatility and scalability [12, 13]; second, we considered Gaussian Mixture Models as an extension of K-Means [12, 14]; and third, we adjusted the clusters manually to match the day of the week (7 clusters total).

3. Experiment

An IEEE test network was used to demonstrate the approach, with data generated using GridLAB-D. This section details the experimental set-up, including how the model interacts with the data and power flow solver. The error metric used throughout the results is introduced, and three heuristics for managing the model error are given.

3.1. Data

The data set was generated using GridLAB-D, an open-source power flow simulation software [3]. The tool is capable of simulating a distribution system on a detailed level in high temporal resolution. Two primary objects, nodes and links, form the majority of the distribution network model. The power flow module within GridLAB-D solves the steady-state node voltages and line currents at each point of the model. The solution of the power flow solver is determined by a three-phase unbalanced flow solver using the NR method [15].

To implement the methodology described in Section 2, an IEEE 123 bus standard feeder test network was formulated in the GridLAB-D modeling environment. The feeder model, as described in detail in [16], operates at 4.16 kV nominal voltage and connects to 344 typical residential houses which were customized to replace the original constant current, power and impedance spot loads. In addition, the circuit accommodates overhead and underground lines, four voltage regulators, shunt capacitor banks and a number of switching elements for controlling two-phase and three-phase lateral feeders.

Coupling the GridLAB-D power flow solver with the IEEE 123 model, we were able to generate a data set to serve as an input to the data-driven power flow solvers described in Section 2. The data set was generated using one minute time-resolution, and contains real and reactive power components at the load level and voltage magnitudes and phases at each node for each time step.

Of the month-long data set created, the first three days were discarded due to the initialization transient behavior of the quasi-steady state simulation, and one week was used for training. A typical week of the data should include all frequently occurring network states and is representative of the future inputs the model will encounter. Since the purpose of this project is also to reduce simulation time, using more than the minimum representative set for training would reduce the utility of this approach. Figure 1 shows how this split was made. The testing set was used as a "ground truth" to evaluate the results of the model-integrated approach.
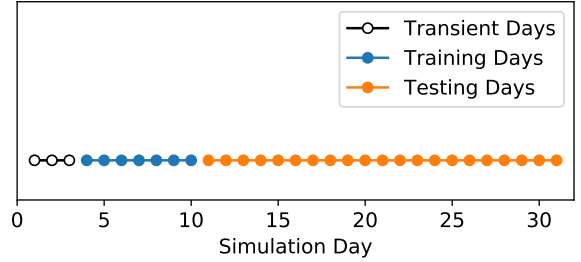

Figure 1: Split of training and testing sets in the month-long data set, including the three dropped days of simulation with initialization transients.

3.2. Set-Up

The main methodology of the test and the interactions of the solver are shown in Figure 2.

In the experiments this process is followed for one day, one week, or all three weeks of the test set. The end output is a mix of estimates from the model and results from the full solver. Comparing this output to the true solution gives an estimate of the error for this approach.

3.3. Error Metric

In this approach we build a model for $\mathbf{v}$ and $\mathbf{a}$ separately, but to represent the error well we use a metric of the total vector error. Where v and a are the values at an individual bus, $\vec{v} = v \exp(ia) = [v \cos a,\ v \sin a]$. We predict this vector for each of the n_v voltage measurement points, $\tilde{\vec{v}}$, and construct the
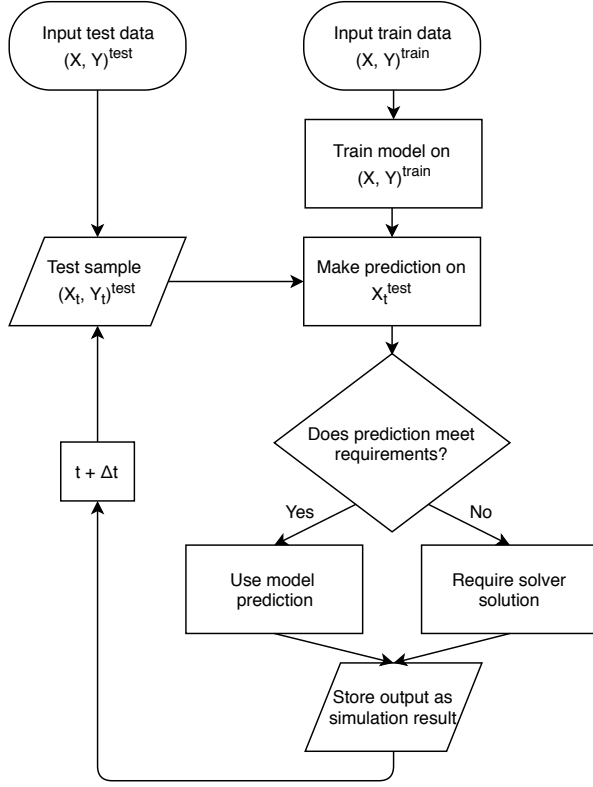
Figure 2: Flow diagram showing the process of building the model, making estimations on testing set samples, and interacting with the full solver to produce the overall simulation result.

normalized vector error:

$$\epsilon_{\vec{v}} = \frac{||\tilde{\vec{v}} - \vec{v}||_2}{||\vec{v}||_2}. \tag{7}$$

No normalization is done where $||\vec{v}||_2$ is zero. Doing this calculation for each measurement point gives a n_v vector of errors, $\epsilon_{\vec{v}}$, and we calculate a measure of the overall prediction error as

$$\epsilon_{\inf\vec{v}} = ||\epsilon_{\vec{v}}||_\infty. \tag{8}$$

We use this error metric throughout the results of this paper, and every time errors are given they are values of $\epsilon_{\inf\vec{v}}$.

3.4. Error Parameters

We consider three tools for making this method more robust and protecting against high-error predictions:

1. **Error Check.** When the solver is used, apply the model as well and store the error of the estimate.

Require that this estimate be below a threshold to invoke the model again.

2. **Distance Check.** Some clusters do not perform as well as others when evaluated on the training set. For under-performing clusters where difficult prediction points have been grouped together, the higher errors typically occur when the sample is near the edge of the cluster. Calculate in what percentile of distance away from the cluster center the new data point falls, and require that this distance be below a certain percentile to use the model.

3. **Step Change Check.** When there is a significant jump between the model's current estimate and the previous simulation solution, that can be an indicator that the model prediction is about to deviate from the true solution. Require that this change in time be below a certain threshold to use the model.

Each of these methods involves parameter tuning and poses a trade-off: making more conservative choices will avoid occasional high error estimates but also slow down the simulation by using the solver more than necessary.

4. Results

Following the methods and experimental set-up described above, we carry out tests to select the best model and error parameters for this data set. Using those modeling choices we then outline results showing the performance relative to the original solver solution.

4.1. Clustering and Model Choice

The initial method learns only one model for the network and uses no clustering, setting a baseline level of performance. The results are shown in Figures 3 and 4. The error distributions are similar between the training and testing sets, though we observe a very small number of extreme values in the test set which stretch the tail of the distribution. Centered around 0.2% overall prediction error these results do not match the accuracy level of the solver, but for many applications and studies using tools like GridLAB-D this is a sufficient estimate.

The time-series of the error shows clear patterns of time dependence in the models performance throughout the day. We capture this by clustering in time using the three previously discussed methods, and the prediction errors for the testing set are presented here: Figure 5 shows the error distribution when clustered based on the day of the week, Figure 6 shows that result for K-Means
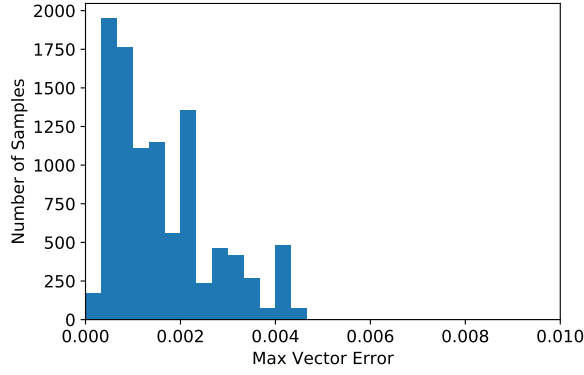
Figure 3: Month-long data set with no clustering: overall prediction error, $\epsilon_{\inf \vec{v}}$, on the training set data.
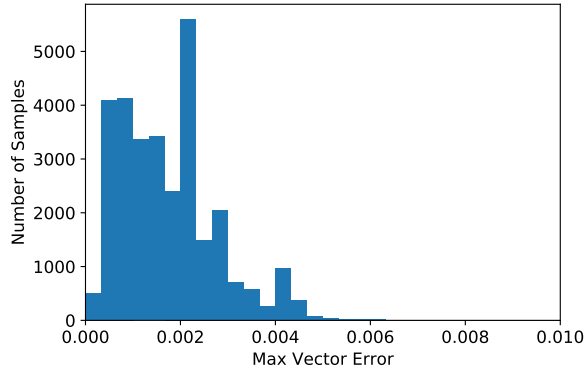

Figure 4: Month-long data set with no clustering: overall prediction error, $\epsilon_{\inf \vec{v}}$, on the testing set data.

clustering, and Figure 7 shows that result for Gaussian Mixture Model clustering. Each of these distributions includes a small number of extreme values, and the figures shown here are clipped. For the testing set the three models respectively have 45.9, 9.7, and 0.3% of values above the cut-off, and maximum errors of 1.4, 4.3e7, and 125.4.

From these figures it is clear that clustering by day of the week gives the least favourable results, significantly worse than the initial cluster-free model from Figure 4. Both K-Means and Gaussian Mixture are improvements over the original and greatly increase the number of samples in the smallest error bin. Both have error tails, but the K-Means method has a smoother distribution of good predictions. Using 7 clusters is enough to represent the system modes well, and increasing the number of clusters seems to flatten the distribution. Therefore we move forward with the K-Means 7 cluster models for the following experiments.

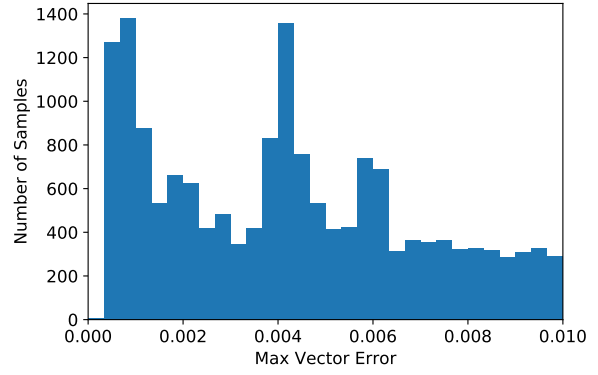Pictured using the time-series of training set error

Figure 5: Trimmed histogram of the errors on the test set using 7 cluster models, one for each day of the week. 45.9% of the samples were excluded for being above this cut-off, and the maximum value was 1.4.
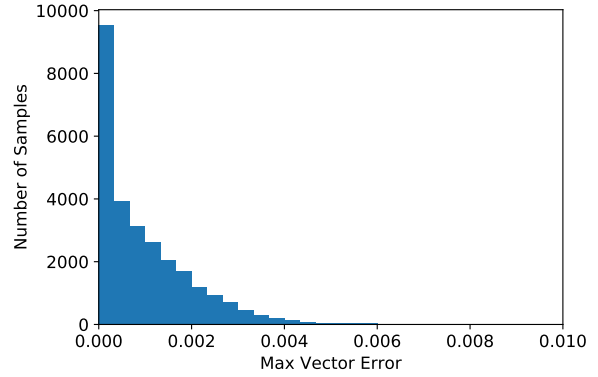

Figure 6: Trimmed histogram of the errors on the test set using 7 K-Means cluster models. 9.7% of the samples were excluded for being above this cut-off, and the maximum value was 4.3e7.

from Figure 3, Figure 8 shows the distribution of the 7 clusters automatically assigned by K-Means clustering.

Investigating the tails of the distribution in Figure 6 shows that the extreme error results always occur for samples far from their cluster center. If the cluster sample-to-center distance is found for each vector in the training set, we can calculate in what percentile of that distribution the extreme error sample falls. We find this is consistently very high, motivating the Distance Check error parameter described in Section 4.2.

4.2. Error Parameter Tuning

Each of the three error parameters was calibrated using the first day of the testing set. The parameters were implemented with a range of values and the prediction errors are shown here in Figures 10a, 10c, and
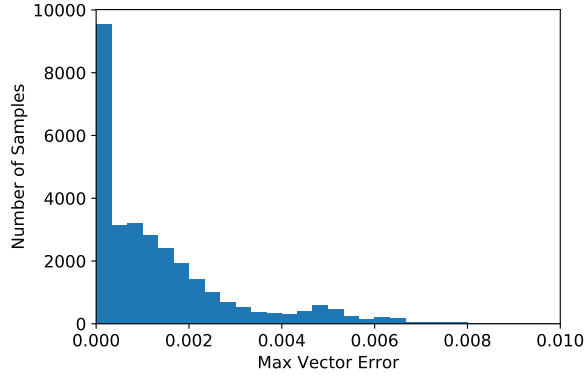
Figure 7: Trimmed histogram of the errors on the test set using 7 Gaussian Mixture cluster models. 0.3% of the samples were excluded for being above this cut-off, and the maximum value was 125.4.
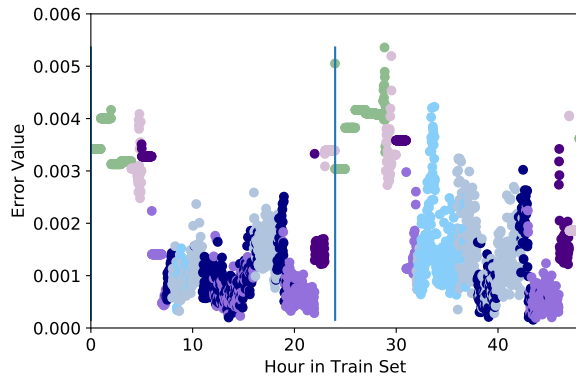

Figure 8: Training set error from the cluster-free model, coloured to show the assignment of points into 7 K-Means clusters. The first two days of the training set data are illustrated here.

10e. For each parameter value the error is calculated for every testing sample in the calibration day, and the 25th, 50th, and 75th percentiles of those results are shown by the shaded regions and solid lines. The vertical dashed lines shows the cut-offs after which extreme error values from the tail of the testing set error distribution (orders of magnitude off) start to enter the solution. We can also compare the results by how often the solver was called. Figures 10b, 10d, and 10f show these results.

Comparing between the two sets of figures we see that using the Step Change Check gives the best guarantee against large errors, as we can reach the higher value of approximately 91% model uses before observing any extreme results. In that case the solver was avoided more than 90% of the time while achieving a median error of less than 0.1%. This reduces the
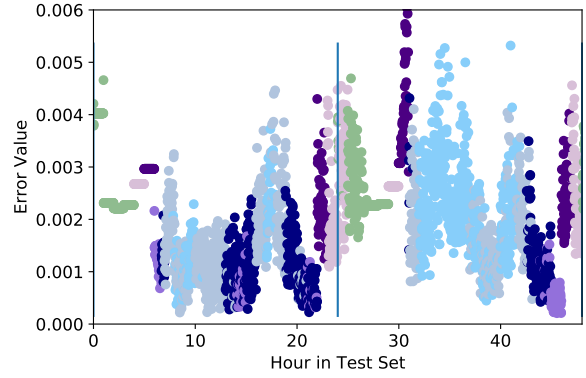

Figure 9: Testing set error from the cluster-free model, as in Figure 8, coloured to show the assignment of points into 7 K-Means clusters. Only the first two days of the testing set data are illustrated. These two days of the week align with the two days shown in Figure 8, so the differences highlight the complexity of the clustering and system modes.

simulation time significantly by minimizing the number of solutions requiring the full solver. Compared with the Error Check, it reaches the same model count and median error at approximately the same time, but is more robust to outliers. It is also noticeable that the Distance Check gives a very low median error, though some of this may be attributable to its lower number of model uses.

Some combination of these checks is optimal. Taking from each of their performance highlights for this network we set the Step Change Check threshold to 20% (-1.6 on the log scale plots) and use an Error Check to try to take advantage of its steeper rise in model uses.

The value of the Error Check threshold should depend on how frequently a check is forced. Without requiring occasional solves to update the error metric, the metric value can become out of date; this may have contributed to the extreme errors included in this test, as the Error Check value was not updated frequently enough to catch those events.

To find the optimal parameters, a sensitivity test was conducted over the values of the Error Check. No Distance Check was used, and the Step Change Check threshold was implemented at 20%. The sensitivity results are shown in Figures 10g and 10h, comparing the parameters in terms of their time savings and accuracy. Analyzing the maximum test error rather than the mean showed a very similar distribution of results to Figure 10h. Together these results point to the best solution of parameters to achieve high accuracy and avoid using the solver: set the maximum interval between checks at 60

minutes and use 0.01 or 1% error as the threshold for when to resume using the solver.

4.3. Final Model

The final parameters of the model for this network are outlined in Table 1. This model was applied to the full test set. The test error results shown in Figure 11 reveal a smooth distribution which grows steeply and has a high peak of near-zero error estimates.

Table 1: Final model design.

Model Choice	Value
Train set length	1 week
Test set length	3 weeks
Clustering method	K-Means
Number of clusters	7
Error Parameter	**Value**
Distance Check	None
Error Check threshold	1 %
Max interval between Error Checks	60 min
Step Change Check threshold	20 %

5. Discussion

There are two main comparison points for these results: one, the cluster-free initial approach; and two, the model-free solver-only approach.

Table 2: Comparison of model results.

Metric	Solver	Initial	Final
Avoided solves	0	100%	86.7%
Median test $\epsilon_{\inf \vec{v}}$	0	0.0014	0.00049
Max test $\epsilon_{\inf \vec{v}}$	0	0.0094	0.0168
Clipped above 0.01	0	0	0.05%

The three simulations are compared in Table 2. Comparing the test error distributions of the initial and final models in Figure 12, it is easy to see the impacts of the model adjustments.

Compared with the first clustering results in Figure 6, the distribution tail of high errors has been virtually eliminated: only 0.05% of the test samples see errors higher than 1% and none of those are higher than 1.68%.

The final model calls on the solver for 13.3% of the samples so it is slower than the initial model, but that is balanced by the improvement in the median test error. With a median overall prediction error, $\epsilon_{\inf \vec{v}}$, of just under 0.05%, the final model is much more accurate.

This error compared to the solver is small enough for the method to be used in many power system modeling applications, especially when it allows the simulation to avoid expensive full solves 86.7% of the time.

Making more conservative choices for each of the error parameters would decrease this error result even further and target the tail of the distribution, but that would also decrease the utility of this approach by requiring more frequent use of computationally expensive solutions. The choice for this trade-off is certainly problem dependent.

A limitation of this method is that it gives no hard guarantees on the model error. Without using the solver it is impossible to exactly estimate the error for each sample. Even with conservative safe-guards or applying the model only to samples in areas where confidence in the model is very high, this may pose a problem for applications that are highly sensitive to solver error.

Tuning these error parameters will likely be different for each network, and the same results cannot be guaranteed if the values selected here in Table 1 are naively applied to new systems. This is a limitation as modeling software tools that integrate this approach will either need to choose conservative defaults or include a mechanism early in the simulation to perform training and calibration.

6. Future Work

We plan to pursue many different avenues of future work expanding on this idea.

We will deploy this methodology in GridLAB-D using an existing program to boost that tools performance for use-cases that are important to California's renewable energy policies. In doing so we will apply the method to larger systems including the IEEE 8500 node test model, the North American Taxonomy Feeder models [6], and real system models provided by utilities to evaluate the method's performance impact on analyses they perform on their distribution systems.

Future work will include automating the error tuning mechanism and finding more data-driven ways to predict each samples test error, for example by learning to predict the NR solver steps required for each input and using that metric as a decision making tool.

We also plan to study how this method could be adapted using periodic re-clustering and re-training of the models for longer time-series simulations where the system evolves slowly over time, such as growing solar photovoltaic penetration or battery storage. The training set can change dynamically as new operating modes emerge, and the process for choosing which samples to keep will become part of the research question.
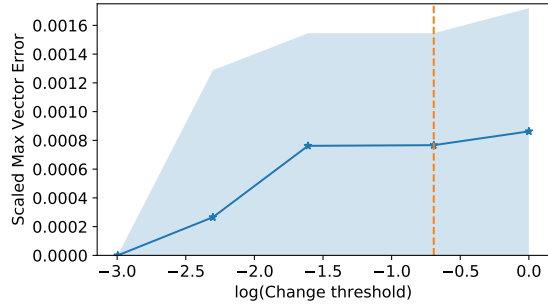
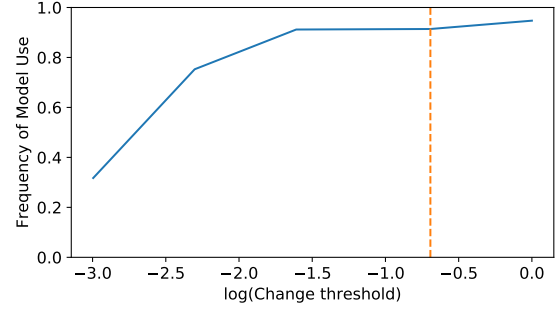Another extension will be to consider more
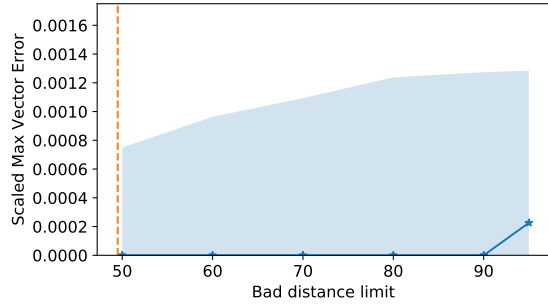
(a) Error based on tuning the Distance Check.
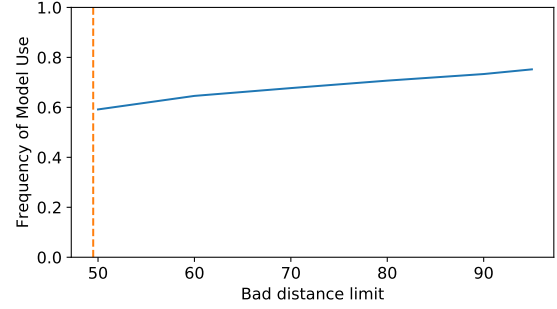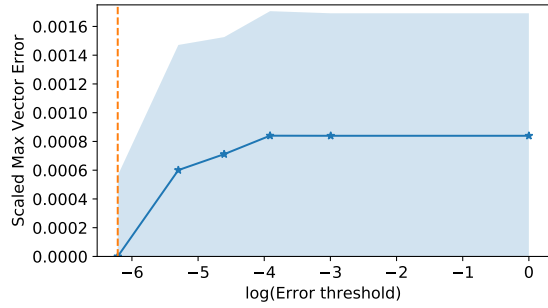
(b) Fraction of times the model was used (rather than the solver) with tuning of the Distance Check.
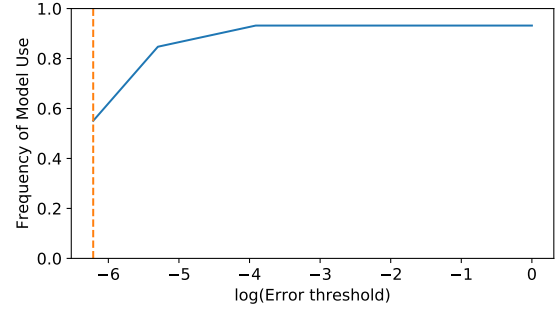
**(c)** Error based on tuning of the Step Change Check.

(d) Fraction of times the model was used (rather than the solver) with tuning of the Step Change Check.
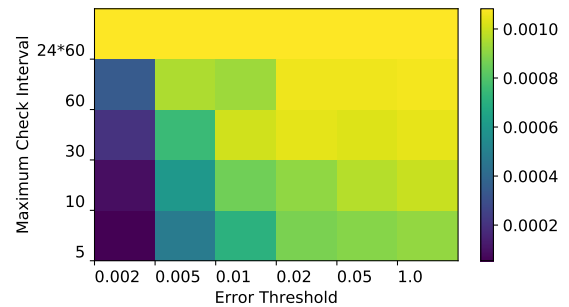
(e) Error based on tuning of the Error Check threshold, with a maximum interval of one hour between checks.

(f) Fraction of times the model was used (rather than the solver) with tuning of the Error Check threshold, with a maximum interval of one hour between checks.
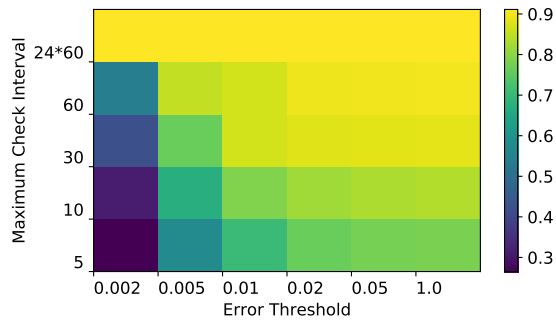
(g) Fraction of test samples for which the model was used in the Error Check sensitivity test.

(h) Mean $\epsilon_{\inf \vec{v}}$ error across the Error Check sensitivity test samples.

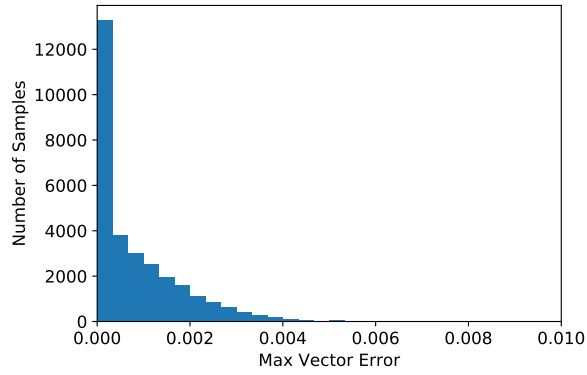**Figure 10:** Results of parameter calibration.

Figure 11: Trimmed histogram of the overall prediction errors on the testing set using the final model. 0.05% of the samples were clipped and the maximum value was 0.0168.
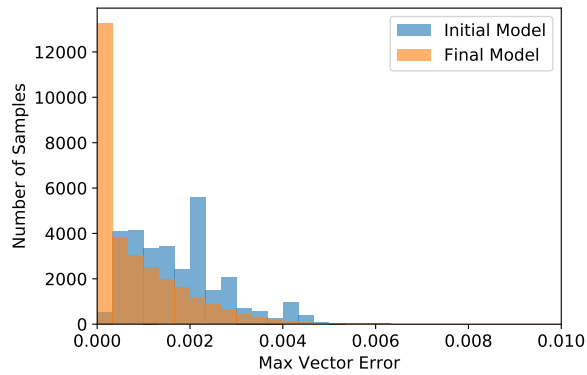

Figure 12: Histograms of the overall prediction error for the initial and final models on the full testing set, highlighting the improvement in the error distribution. This is a combination of Figures 4 and 11.

clustering methods and study how the clustering results represent phenomena in the data set or the solver.

7. Conclusion

A data-driven power flow solution method was developed for a typical distribution system model. Its implementation was shown to reduce the computation time required for a large-scale quasi-steady state power flow simulation. By coupling the solver with a data-driven approach, the usage of the physics-based solver was reduced by 86.7% and the median overall prediction error achieved was 0.049%.

Further investigation of the methodology will include a variety of larger distribution networks as well as longer time-series demonstrations and periodic re-training of the model. To improve the speed and

efficiency of utility planning studies and the evaluation of operation strategies developed by researchers and electrical utility companies, this method will be implemented in GridLAB-D.

Acknowledgements

References

[1] L. Thurner, A. Scheidler, F. Schafer, J. H. Menke, J. Dollichon, F. Meier, S. Meinecke, and M. Braun, "Pandapower - An Open-Source Python Tool for Convenient Modeling, Analysis, and Optimization of Electric Power Systems," *IEEE Transactions on Power Systems*, vol. 33, pp. 6510–6521, 11 2018.

[2] D. P. Chassin, K. P. Schneider, and C. Gerkensmeyer, "GridLAB-D: An open-source power systems modeling and simulation environment," in *IEEE/PES Transmission and Distribution Conference and Exposition.*, (Chicago), pp. 1–5, IEEE, 2008.

[3] D. P. Chassin and R. G. Pratt, "Nested, hierarchical resource allocation schema for management and control of an electric power grid," 2016.

[4] W. Kersting, *Distribution System Modeling and Analysis*. 3 ed., 2012.

[5] W. F. Tinney, S. Member, and C. E. Hart, "Power Flow Solution by Newton's Method," *IEEE Transactions on Power Apparatus and Systems*, vol. PAS-86, no. 11, pp. 1449–1967, 1967.

[6] K. P. Schneider, J. C. Fuller, F. K. Tuffner, and Y. Chen, "Modern Grid Strategy: Enhanced GridLAB-D Capabilities Final Report," tech. rep., Pacific Northwest National Lab, 2009.

[7] R. F. Reinhart, Z. Shareef, and J. J. Steil, "Hybrid analytical and data-driven modeling for feed-forward robot control," *Sensors (Switzerland)*, vol. 17, 2 2017.

[8] L. Ladický, S. Jeong, B. Solenthaler, M. Pollefeys, and M. Gross, "Data-driven fluid simulations using regression forests," *ACM Transactions on Graphics*, vol. 34, pp. 1–9, 10 2015.

[9] Y. Weng, R. Negi, and M. D. Ilić, "Historical data-driven state estimation for electric power systems," in *2013 IEEE International Conference on Smart Grid Communications, SmartGridComm 2013*, pp. 97–102, 2013.

[10] J. Yu, Y. Weng, and R. Rajagopal, "Robust mapping rule estimation for power flow analysis in distribution grids," in *2017 North American Power Symposium, NAPS 2017*, Institute of Electrical and Electronics Engineers Inc., 11 2017.

[11] J. Yu, Y. Weng, and R. Rajagopal, "PaToPa: A Data-Driven Parameter and Topology Joint Estimation Framework in Distribution Grids," *IEEE Transactions on Power Systems*, vol. 33, pp. 4335–4347, 7 2018.

[12] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, E. Duchesnay, and G. Louppe, "Scikit-learn: Machine Learning in Python," *Journal of Machine Learning Research*, vol. 12, no. Oct, pp. 2825–2830, 2011.

[13] D. Arthur and S. Vassilvitskii, "k-means++: The Advantages of Careful Seeding," in *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pp. 1027–1035, Society for Industrial and Applied Mathematics, 2007.

[14] P. D. McNicholas, *Mixture Model Based Classification*. Boca Raton: Taylor & Francis, 2017.

[15] C. Fuerte-Esquivel, T. Venegas, C. Angeles-Camacho, A. Garcia, J. Pereira, S. Carneiro, V. da Costa, and N. Martins, "Discussion of "Three-phase power flow calculations using the current injection method" [and reply]," *IEEE Transactions on Power Systems*, vol. 15, pp. 1451–1453, 8 2002.

[16] Distribution System Analysis Subcommittee, "IEEE 123 Node Test Feeder," tech. rep., IEEE Power Engineering Society, 1992.