TensorConvolutionPlus

A python package for distribution system flexibility area estimation

Chrysostomou, Demetris; Torres, José Luis Rueda; Cremer, Jochen Lorenz

DOI
10.1016/j.softx.2025.102241

Publication date
2025

Document Version
Final published version

Published in
SoftwareX

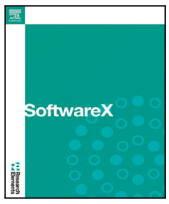
Citation (APA)
Chrysostomou, D., Torres, J. L. R., & Cremer, J. L. (2025). TensorConvolutionPlus: A python package for distribution system flexibility area estimation. *SoftwareX*, *31*, Article 102241. https://doi.org/10.1016/j.softx.2025.102241

SoftwareX 31 (2025) 102241

Contents lists available at ScienceDirect

SoftwareX

journal homepage: www.elsevier.com/locate/softx



Original software publication

TensorConvolutionPlus: A python package for distribution system flexibility area estimation

Demetris Chrysostomou *, José Luis Rueda Torres, Jochen Lorenz Cremer

TU Delft, Department of Electrical Sustainable Energy, Delft, The Netherlands

ARTICLE INFO

Keywords:
Python package
Flexibility area estimation
Distribution system flexibility
TSO-DSO coordination

ABSTRACT

Power system operators need new, efficient operational tools to use the flexibility of distributed resources and deal with the challenges of highly uncertain and variable power systems. Transmission system operators can consider the available flexibility in distribution systems (DSs) without breaching the DS constraints through flexibility areas. However, there is an absence of open-source packages for flexibility area estimation. This paper introduces TensorConvolutionPlus, a user-friendly Python-based package for flexibility area estimation. The main features of TensorConvolutionPlus include estimating flexibility areas using the TensorConvolution+ algorithm, the power flow-based algorithm, an exhaustive PF-based algorithm, and an optimal power flow-based algorithm. Additional features include adapting flexibility area estimations from different operating conditions and including flexibility service providers offering discrete setpoints of flexibility. The TensorConvolutionPlus package facilitates a broader adaptation of flexibility estimation algorithms by system operators and power system researchers.

Code metadata

Current code version	v0.1.1
Permanent link to code/repository used for this code version	https://github.com/ElsevierSoftwareX/SOFTX-D-25-00033
Permanent link to Reproducible Capsule	https://codeocean.com/capsule/4767295/tree/v1
Legal Code License	CC-BY
Code versioning system used	git
Software code languages, tools, and services used	Python
Compilation requirements, operating environments & dependencies	Python≥ 3.10, matplotlib≥ 3.8.2, networkx≥ 3.1, numpy≥ 1.24.3, pandapower≥ 2.13.1, pandas≥ 1.5.3, scikit-learn≥ 1.3.0, scipy≥ 1.11.2, seaborn≥ 0.13.2, tntorch≥ 1.1.1, torch≥ 2.0.1, tqdm≥ 4.66.1
Link to developer documentation/manual	https://demetris-ch.github.io/TensorConvolutionFlexibility/
Support email for questions	D.Chrysostomou@tudelft.nl

1. Motivation and significance

Power systems encounter an operational transition as renewable energy sources (RES) penetration rises, and the conventional generation output decreases. This operational transition includes coordinating transmission system operators (TSOs) and distribution system operators (DSOs). RES are mainly connected to distribution systems (DSs) and have high variability and uncertainty, challenging the TSOs and DSOs who need to maintain their system balance. However, RES and active users in DSs can also offer flexibility to contribute to the reduction of these challenges. This flexibility corresponds to the RES or active users changing their generation or consumption setpoints to support the system operators. The RES and active users that offer flexibility constitute the flexibility service providers (FSPs). Therefore, TSOs and DSOs need operational tools that can efficiently allow communicating and using FSP flexibility [1,2].

TSO-DSO coordination approaches can be categorized in TSO-managed, DSO-managed, or TSO-DSO hybrid models [3,4], with recent TSO-DSO coordination approaches also developing multi-interval

* Corresponding author.
E-mail address: D.Chrysostomou@tudelft.nl (D. Chrysostomou).

https://doi.org/10.1016/j.softx.2025.102241
Received 19 January 2025; Received in revised form 13 May 2025; Accepted 16 June 2025
Available online 26 June 2025

TSO-DSO coordination [5]. The proposed package is focused on DSO-managed coordination, where the DSO validates the feasibility and aggregates the FSP flexibility to inform the TSO of the available, feasible FSP services. The DSO can validate and aggregate the FSP flexibility using Flexibility areas (FAs) [3]. FAs are areas in the active (P) and reactive (Q) power plane, illustrating which setpoints TSOs can achieve at a TSO-DSO interconnection node when utilizing feasible flexibility from the DSs.

FA estimation approaches mainly apply power flows (PF) or optimal power flows (OPF) [6–16] to explore the limits of the offered flexibility in the PQ space. PF-based algorithms are simple and consistent but slow, whereas OPF-based algorithms are faster but can have convergence issues. A recently proposed FA estimation algorithm, TensorConvolution+ [17], explores the limits and the density of feasible flexibility shift combinations to reach each FA setpoint. TensorConvolution+ applies convolution and tensor operations to combine flexibility shifts and evaluate their feasibility for the system's technical constraints. Additional functionalities of the TensorConvolution+ approach include storing tensors from prior estimations and adapting FAs for altered operating conditions (OCs). Forecast errors in distribution systems can provide uncertainty in day ahead FA estimation, with recent approaches exploring chance constraint [14], robust optimization [8,18], or probabilistic [10,19–21] algorithms. The proposed package implements deterministic FA estimation. Fast FA estimation algorithms can potentially be used in close to real-time, reducing the impact of forecast errors [7]. Existing FA estimation approaches include case studies with real-world systems and data. For example [22] used data from the French DSO whereas [23] used the Swiss TS. The proposed software currently supports systems in the pandapower format. Thus, users should import their systems in the pandapower format, or use the existing pandapower systems.

The operational transition and data availability in power systems provided opportunities for digitalizing power systems. This digitalization corresponds to more intelligent, effective, green power grid operations [24]. Data-driven approaches for power system operations are emerging with works on dynamic [25] and static simulations [26,27], from protection gaps [25], to dynamic security assessment [28] and probabilistic approaches [26]. The main drivers for change in power systems are decarbonization, digitalization, and decentralization, with flexibility as a key for decarbonization [29]. The digitalization of power systems resulted in the emergence of open-source tools. Power systems open-source tools include the PandaPower [30] in Python, PSAT [31] in Matlab and GNU/Octave, MatPower [32] in Matlab. More recent tools with increased efficiency include [33] in C++. As highlighted by [30], software developed in languages with open-source licenses, such as Python, C++, and Julia, can be used as stand-alone or extended with other libraries. These advantages of open-source libraries drove researchers to design more specialized power system-related packages such as [34–37]. FA estimation is an emerging field in power engineering that can improve the power system stability and utilization of flexibility from decentralized resources. However, currently, there are no open-source FA estimation packages. An open-source package for FA estimation can accelerate the adoption of FAs by power system operators and attract more researchers to this emerging field.

The developed Python-based package for FA estimation focuses on the TensorConvolution+ algorithm [17] but also includes a traditional PF-based algorithm, an exhaustive PF-based algorithm, and an OPF-based algorithm.

2. Software description

The software framework is implemented in Python. The package can be installed from the Python package index (PyPi). The code implementation is available on GitHub. The documentation for the package's main functions, classes, supporting functions, and case studies is available online and was built using the Sphinx library [38].

The seven main software functionalities are two PF-based algorithms, one OPF-based algorithm, and four versions of the TensorConvolution+ algorithm. Fig. 1 illustrates the usage of the package functionalities. The user calls the *FA_Estimator* script of the TensorConvolutionPlus package and selects one of the main functionalities. The selected algorithm functionality estimates the FA and stores locally:

1. The FA image in a portable document format (PDF) file.
2. The FA results in a comma-separated values (CSV) file.
3. A text file with the simulation information on duration and algorithm-specific details.

The *tcp_plus_save_tensors* also includes additional files from the FA results. The user inputs depend on the functionality.

2.1. Software architecture

The proposed software architecture intends to allow efficient modification and expansion of specific sub-processes of the FA estimation problem. Table 1 highlights the roles of the Python scripts implementing the package functionalities. The *json_reader* script checks if each input is within the acceptable options to avoid erroneous results. If *json_reader* detects an unacceptable input, the FA estimation does not begin, and the user is informed about the input causing the issue and the acceptable values. Users can identify which scripts and functionalities to modify or expand to fulfill additional needs. For example, to add or modify the plotting functions of the package, one would modify the *plotting* script. For more complex modifications, such as adding different sampling techniques for the PF-based algorithms, the *data_sampler* and *json_reader* would be the only scripts requiring modification. The user could modify the *sample_from_rng* function in *data_sampler* to sample FSP shifts using a new distribution when the new distribution keyword is selected. The *json_reader* modification corresponds to adding new acceptable options, the new sampling distribution keyword in *self.distribution* in the function *tester* of the class *SettingReader*. An input outside the acceptable options in the *json_reader* stops the process before the simulation to avoid erroneous or untested results. This architecture also allows potential future expansions to new FA estimation algorithms, where an additional script can be created and integrated with the *json_reader* and *FA_Estimator* scripts without impacting other processes. For new FA estimation algorithms, the *json_reader* would need to create any new variables for the algorithms in the *_init_* of the class *SettingReader*, and add the acceptable options for each variable in the *tester* function. The new algorithms should be created as a new functionality in the *FA_Estimator* script. Depending on the needs of the new algorithms, the user can call sampling functions from the *data_sampler*, plotting functions from the *plotting* script, or other generic functions from the *utils* script.

The package's GitHub repository includes the Python scripts under the "src/TensorConvolutionPlus" directory. The package dependencies include *pandapower* to perform PF and OPF operations, *PyTorch* for tensor operations, *SciPy* for convolution operations, *tntorch* and *scikit-learn* for additional TensorConvolution+ subprocesses. *NumPy* and *pandas* are used for data storage, processing, and sampling, *tqdm* illustrates the FA estimation progress, and *matplotlib* and *seaborn* generate the figures.

2.2. Software functionalities

The *FA_Estimator* script includes the main functionalities as in Fig. 2. The *monte_carlo_pf* and *exhaustive_pf* functions apply PF-based FA estimation algorithms. The *opf* function applies the OPF-based FA estimation. The *tc_plus*, *tc_plus_merge*, *tc_plus_save_tensors*, *tc_plus_adapt* functions perform different versions of the TensorConvolution+ algorithm. The common inputs for all main functionalities are the network pandapower object (*net*), the network name (*net_name*), indices of load FSPs (*fsp_load_indices*), indices of distributed generation FSPs

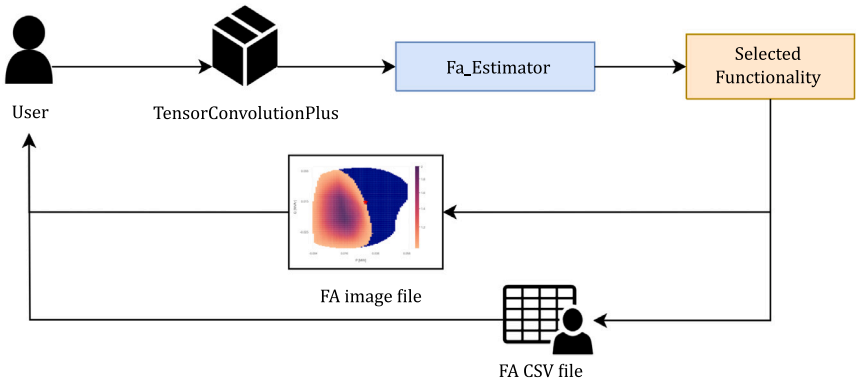


Fig. 1. TensorConvolutionPlus package usage through the script ( ) FA_Estimator and its main functionalities ( ).

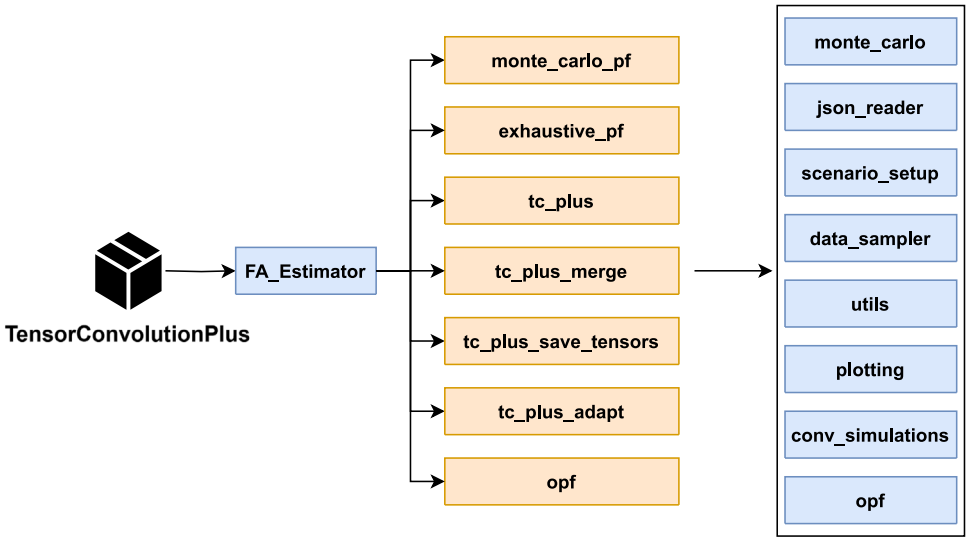


Fig. 2. Package main functions ( ) relationship (⟶) with python scripts ( ).

Table 1
Package script roles.

Script	Role
FA_Estimator	Package main script which includes the main functionalities.
json_reader	(i) Read input settings and create a SettingReader object with the algorithm parameters. (ii) Validate that the inputs are within the acceptable options.
data_sampler	Sample flexibility shifts from flexibility providers.
scenario_setup	Update network and SettingReader object based on the algorithm input parameters.
opf	Perform the OPF-based FA estimation algorithm.
monte_carlo	Perform the PF-based FA estimation algorithms.
conv_simulations	Perform the TensorConvolution+ algorithm functionalities.
utils	Provide generic functions to the other scripts.
plotting	Generate figures of resulting FAs.

(*fso_dg_indices*), scenario type for initial topology and OCs (*scenario_type*), and system constraints for maximum component loading [%] (*max_curr_per*), maximum voltage [p.u.] (*max_volt_pu*), and minimum voltage [p.u.] (*min_volt_pu*). All functionality inputs are optional. However, to estimate FAs, at least one distributed generation or load FSP is required. The remaining scripts, at the right of Fig. 2, provide functions and sub-processes to implement the main functionalities.

3. Implementation and empirical results

The main building blocks for the implemented FA estimation algorithms are (i) initializing network and FA estimation settings, (ii) performing simulations for FSP flexibility shifts on the network, (iii) processing the simulation results, and (iv) plotting and storing the simulation results. All functions have similar block (i), the Algorithm 1. The plotting functions differ between the functionalities.

3.1. PF-based functions

The PF-based functions differ from the FSP flexibility shift sampling functions. Therefore, the main difference between the PF-based functions is:

- *monte_carlo_pf* uses a probability *distribution* (input) to obtain *no_samples* (input) of flexibility shift combinations.

Algorithm 1 Initialize network and FA estimation settings.

Require: *fsp_load_indices* and/or *fsp_dg_indices*,
initialize *SettingReader* object with estimation settings,
check if SettingReader has acceptable values,
if *net* is None **then**,
net ← pandapower network with name=*net_name*,
change *net* topology and OC for *scenario_type*,
end if
return *SettingReader* with *net*.

- *exhaustive_pf* uses the increments *dp, dq* (inputs) for P and Q to sample all possible discretized flexibility shift combinations.

Algorithm 2 illustrates the algorithm for both the PF-based functions after the samples are obtained. The *samples* array includes the FSP shift combination samples. PCC is the point of common coupling between the TSO and the DSO.

Algorithm 2 PF-based FA estimation.

Require: *samples*, *SettingReader*, *net*,
init_net ← *net*,
for *sample* ∈ *samples* **do**,
apply *sample* on *net*,
run PF on *net*,
if *net* OC are within system constraints **then**,
store *sample* index and PCC P, Q as feasible,
else
store *sample* index and PCC P, Q as not-feasible,
end if
net ← *init_net*,
end for
store FA PDF, CSV, and text file.

3.1.1. Empirical results

The PF-based functions can illustrate consistent performance under various network structures and FSP combinations. However, as FSPs increase, the performance deteriorates. The Monte-Carlo-based algorithm could require large *no_samples* to capture the margins of the flexibility area. The exhaustive PF-based algorithm can become intractable for more than 3 FSPs and small *dp, dq*.

3.2. OPF-based function

The OPF-based algorithm applies four multi-objective optimizations (MOO). These optimizations aim to identify the maximum feasible active (P_{PCC}) and reactive power (Q_{PCC}) at the PCC achieved using the available flexibility as:

1. $\max(\alpha P_{PCC} + (1-\alpha)Q_{PCC})$,
2. $\max(-\alpha P_{PCC} + (1-\alpha)Q_{PCC})$,
3. $\max(\alpha P_{PCC} + (\alpha-1)Q_{PCC})$,
4. $\max(-\alpha P_{PCC} + (\alpha-1)Q_{PCC})$.

The variable $\alpha \in [0,1]$ provides a plane in which the active and reactive power shifts are combined. Therefore, the algorithm iteratively changes α in steps provided through the additional input the *opf_step*. For example, an *opf_step*= 0.1 results in 11 iterations per MOO, thus 44 OPFs to estimate the FA.

3.2.1. Empirical results

The OPF-based function has convergence issues for different network structures. The OPF-based function can converge for the radial CIGRE MV network when ignoring transformer loading limitations but might not converge in other networks, e.g., the Oberrhein network. These issues are due to the OPFs performed for each MOO failing to converge to identify an optimal solution within the constraints and network settings.

3.3. TensorConvolution+ functions

The TensorConvolution+ functions correspond to the algorithm proposed in [17]. The function *tc_plus* corresponds to the generic approach of the algorithm, whereas the rest accommodate specific use cases.

TensorConvolution+ initially creates samples of all flexibility shifts for each FSP with increments *dp, dq* (inputs) for active and reactive power, respectively. The samples do not include combinations of FSPs. Thus, the number of samples increases linearly with the addition of FSPs. The *flex_shape* input characterizes the boundaries of each FSP flexibility. Currently, the FSP shapes can be:

1. *Smax*: The FSP output apparent power cannot exceed its maximum apparent power, resulting in a semi-oval flexibility shape.
2. *PQmax*: The FSP active and reactive power outputs cannot exceed the maximum apparent power, resulting in a rectangular flexibility shape.

Algorithm 3 *tc_plus* FA estimation.

Require: *samples*, *SettingReader*, *net*,
init_net ← *net*,
Ω^{γ} ← set of *net* components,
for *sample* ∈ *samples* **do**,
run PF on *sample* and record impact on network components,
end for
Ω_{sm}^{FSPs} ← set of FSPs with capacity smaller than dp, dq,
impacts ← the FSP impacts on each $\gamma \in \Omega^{\gamma}$,
uFA ← the unconstrained FA using convolutions on all $FSP \in \Omega^{FSP}$,
Ω_{γ}^{FSP} ← set of FSPs that impact γ more than the sensitivity thresholds,
Ω^{γ} ← remove all γ from Ω^{γ} that cannot reach the system constraints from the maximum FSP impacts,
for $\gamma \in \Omega^{\gamma}$ **do**,
Ξ_{γ} ← apply tensor-convolution for all feasible Ω_{γ}^{FSP} combinations,
A_{γ} ← sum Ξ_{γ} in all dimensions except the first 2,
Y_{γ} ← apply convolution between A_{γ} and the $\Omega^{FSP} \setminus \Omega_{\gamma}^{FSP}$ shifts,
end for
if $\Omega^{\gamma} = \emptyset$ **then**,
$FA \leftarrow uFA$,
else
$FA \leftarrow$ the element-wise minimum between $Y_{\gamma} \forall \gamma \in \Omega^{\gamma}$,
end if
normalize FA,
get axes of result and create a result data frame,
if $\Omega_{sm}^{FSPs} \neq \emptyset$ **then**,
$FA_i \leftarrow$ bilinear interpolation on FA to increase its size,
$FA_{sm} \leftarrow$ convolute FA_i with the FSPs in Ω_{sm}^{FSPs},
normalize FA_{sm},
end if
store FA PDF, CSV, and text file.

Using these samples and the outputs of Algorithm 1, the function *tc_plus* performs Algorithm 3 to estimate and plot the FA. The TensorConvolution+ algorithm applies convolutions to combine the flexibility shifts between FSPs. However, as FAs consider the network constraint limitations, the FA algorithm needs to first filter the feasible and non-feasible combinations. Thus, for sensitive network components (close to network constraints), TensorConvolution+ modifies the convolution operation to avoid the summation step and stores the resulting impacts from combined FSP shifts to tensors. The first two dimensions of the

tensors correspond to the PCC active and reactive powers, whereas the additional dimensions include the impact of a specific flexibility combination on a sensitive network component. Following these tensor operations, TensorConvolution+ filters out the FSP combinations resulting in non-feasible conditions and sums all but the first two tensor dimensions, as the convolution summation step.[17] provides further illustrative and mathematical analysis of TensorConvolution+. The main differences between *tc_plus*, *tc_plus_merge*, *tc_plus_save_tensors*, and *tc_plus_adapt*:

- *tc_plus_save_tensors* stores extracted information and sensitivity tensors locally. This functionality reduces the tensors' memory requirements using tensor train decomposition (TTD). This reduction causes delays and is therefore excluded from the *tc_plus* function.
- *tc_plus_adapt* does not sample flexibility shifts nor estimates network component sensitives, as it adapts from the FA estimated in previous simulations for the same FSP offers.
- *tc_plus_merge* is useful when memory limitations do not allow estimating FAs with the *tc_plus* function. The *tc_plus_merge* function estimates the electrical distance between all FSPs. When a network component is sensitive to more than *max_FSPs* (input), this function merges the flexibility between the two electrically closest components iteratively until the network component is sensitive to *max_FSPs*.

3.3.1. Empirical results

The TensorConvolution+ functions perform computationally better in GPUs, where tensor operations can be faster. Simulations in different network topologies showcased consistent performance with TensorConvolution+.

TensorConvolution+ can have memory issues and terminate the simulation for networks with multiple components close to the system constraints, small *dp, dq*, and increased FSPs. GPUs with higher VRAM reduce these limitations. When memory issues persist, *tc_plus_merge* can mitigate these issues but could reduce the estimation accuracy.

4. Illustrative examples

All examples were performed using the A100 GPU in Google Colab [39]. To use the package, the user can perform two steps. The first step is installing the package through pip as:

```
pip install TensorConvolutionPlus
```

The second step is importing the package's *FA_Estimator* in a Python script as:

```
from TensorConvolutionPlus import FA_Estimator as TCP
```

The user can use any main function from Fig. 2 using the imported *TCP*. The following subsections showcase the main functions of the package after the above steps.

4.1. PF and OPF functionalities

This section includes examples using the Monte Carlo PF, exhaustive PF, and OPF functionalities. These examples used the Python script code:

```
TCP.monte_carlo_pf(net_name='MV Oberrhein0', no_samples=1000, fsp_load_indices=[1, 2, 3], fsp_dg_indices=[1, 2, 3], distribution='Uniform')

TCP.monte_carlo_pf(net_name='MV Oberrhein0', no_samples=6000, fsp_load_indices=[1, 2, 3], fsp_dg_indices=[1, 2, 3], distribution='Uniform')

TCP.monte_carlo_pf(net_name='MV Oberrhein0', no_samples=20000, fsp_load_indices=[1, 2, 3], fsp_dg_indices=[1, 2, 3], distribution='Uniform')

TCP.monte_carlo_pf(net_name='MV Oberrhein0', no_samples=40000, fsp_load_indices=[1, 2, 3], fsp_dg_indices=[1, 2, 3], distribution='Uniform')

TCP.exhaustive_pf(net_name='MV Oberrhein0', dp=0.01, dq=0.02, fsp_load_indices=[5], fsp_dg_indices=[5])

TCP.opf(net_name='CIGRE MV', opf_step=0.1, fsp_load_indices=[1, 4, 9], fsp_dg_indices=[8])
```

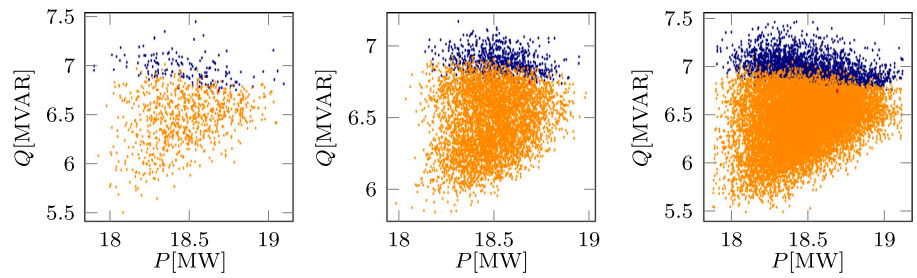
Fig. 3 illustrates the expected output FA for each line respectively. In terms of computational burden, the simulations required 55 s for Fig. 3(a), 5 min and 5 s for Fig. 3(b), 18 min for Fig. 3(c), 36 min for Fig. 3(d), 36 min and 18 s for Fig. 3(e), and 33.7 s for Fig. 3(f). Fig. 3(e) performed 43121 power flows and Fig. 3(f) executed 44 OPFs. The Monte Carlo-based results showcase clearer FA margins for $20K$ and $40k$ samples than lower values. However, the number of samples for clearer margins can also depend on the number of FSPs. Monte Carlo-based functions can be better than the exhaustive PF-based function in exploring FA margins for scenarios with more FSPs. Lowering the resolution for the exhaustive approach for producing clear FA margins can be intractable as FSPs increase.

4.2. TensorConvolution+

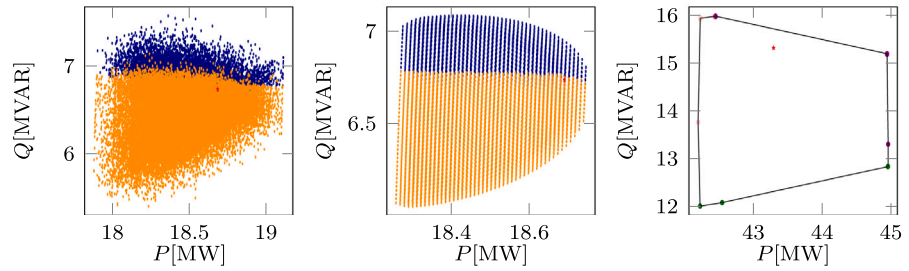
This section illustrates examples using the TensorConvolution+ FA estimation functionality, using the Python lines:

```
TCP.tc_plus(net_name='MV Oberrhein0', fsp_load_indices=[1, 2, 3], dp=0.05, dq=0.1, fsp_dg_indices=[1, 2, 3])

TCP.tc_plus(net_name='MV Oberrhein0', fsp_load_indices=[1, 2, 3], dp=0.075, dq=0.15, fsp_dg_indices=[1, 2, 3])

TCP.tc_plus_merge(net_name='MV Oberrhein0', fsp_load_indices=[1, 2, 3], dp=0.025, dq=0.05, fsp_dg_indices=[1, 2, 3], max_fsps=5)

TCP.tc_plus(net_name='MV Oberrhein0', fsp_load_indices=[1, 2], dp=0.05, dq=0.1, fsp_dg_indices=[1, 2, 3])

TCP.tc_plus(net_name='MV Oberrhein0', fsp_load_indices=[1, 2], dp=0.025, dq=0.05, fsp_dg_indices=[1, 2, 3])

TCP.tc_plus(net_name='CIGRE MV', fsp_load_indices=[3, 4, 5], dp=0.05, dq=0.1, fsp_dg_indices=[8], non_linear_fsps=[8])
```

Fig. 4 shows the expected output FAs from the above lines respectively. Fig. 4(a) required 13.1 s whereas the second line reduces the resolution, with Fig. 4(b) requiring 7.6 s. Increasing the resolution from the first line to $dp = 0.025, dq = 0.5$ exceeded the A100 GPU memory, stoping the simulation. Thus, running the *tc_plus_merge* function merged FSPs when more than 5 FSPs impacted a network component and estimated the FA of Fig. 4(c) for this higher resolution in 37.9 s. The fourth line reduced the number of FSPs to 5 from Fig. 4(a) with the



(a) Monte Carlo PF with $1k$ samples. (b) Monte Carlo PF with $6k$ samples. (c) Monte Carlo PF with $20k$ samples.



(d) Monte Carlo PF with $40k$ samples. (e) Exhaustive PF (f) OPF

Fig. 3. PF-based and OPF-based FA estimations.

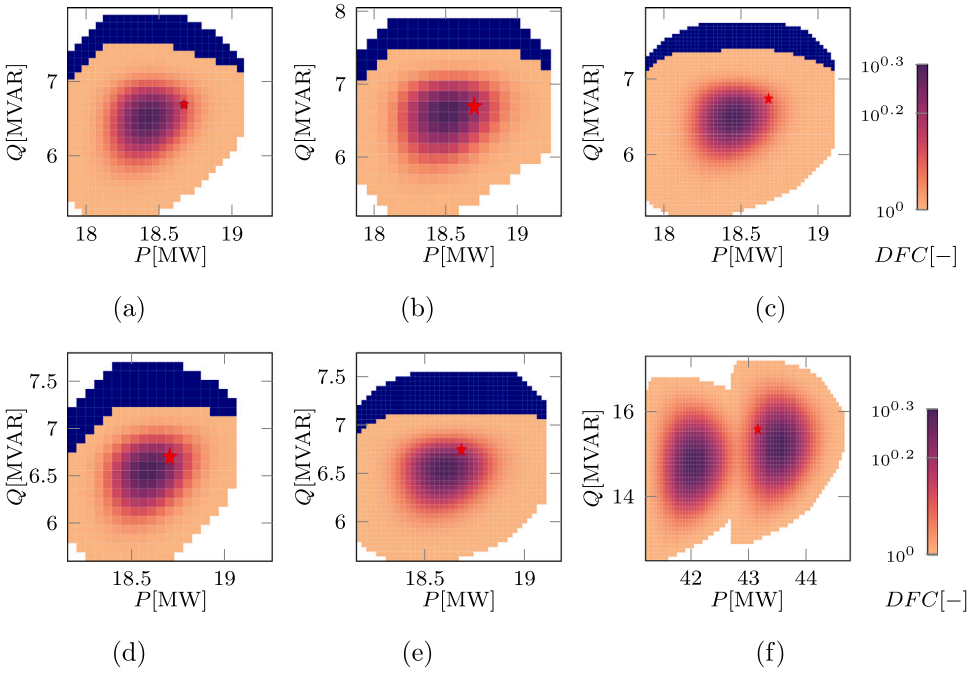


Fig. 4. TensorConvolution+ algorithm examples with all linear FSPs in (a)–(e), merging of FSPs in (c) and one non-linear FSP in (f).

same resolution, resulting in Fig. 5 in 9 s. Increasing the resolution of Fig. 5 to $dp = 0.025, dq = 0.5$ did not cause memory issues for the GPU, resulting in Fig. 4(e) in 26.5 s. The last line includes an FSP offering discrete setpoints of flexibility. The input *non_linear_fsps* specifies which of the FSPs referenced in the *fsp_dg_indices* can only offer 2 setpoints; current output or full output reduction. The duration for Fig. 4(f) was 17.8 s.

4.3. TensorConvolution+ adapt

To adapt FA estimations using estimations from expected or prior operating conditions, TensorConvolution+ requires storing the relevant information from these prior FA estimations locally. Therefore, the package's user should first call the *tc_plus_save_tensors* function to store

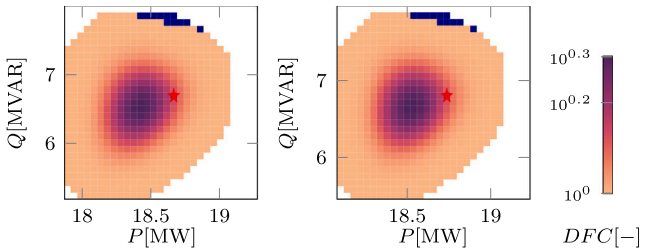
the information. This function performs TTD to reduce the space required to store tensors but requires more extensive computational time than *tc_plus* to execute the additional TTD and storing operations. After *tc_plus_save_tensors* is executed, then *tc_plus_adapt* can use the stored information to estimate FAs for altered related OCs if the network topology and FSPs are consistent. Bellow, an example script storing the information, altering the operating conditions, and adapting the FA for the new operating conditions:

```
# Step1: Define the consistent FSPs for the storing and adapting functions
fsp_load_indices = [1, 2, 3]
fsp_dg_indices = [1, 2, 3]
# Step 2: Estimate the FA and store the relevant information for adaptation
TCP.tc_plus_save_tensors(net_name='MV Oberrhein0', fsp_load_indices=fsp_load_indices, dp=0.05, dq=0.1, fsp_dg_indices=fsp_dg_indices)
# Step 3: Modify the network operating conditions
net, net_tmp = pn.mv_oberrhein(separation_by_sub=True)
net.load['sn_mva'] = list(net.load['p_mw'].pow(2).add(net.load['q_mvar'].pow(2)).pow(0.5))
net.load['scaling'] = [1 for i in range(len(net.load))]
net.sgen['scaling'] = [1 for i in range(len(net.sgen))]
net.switch['closed'] = [True for i in range(len(net.switch))]
# Step 4: Fix the network structure
net = fix_net(net) # This function is included in the documentation (C8 of Tab.1)
# Step 5: Sample a new operating condition with randomness
rng = np.random.RandomState(212)
net, rng = rand_resample(net, fsp_load_indices, fsp_dg_indices, rng, 0.05, 0.01, 0.05, 0.01) # This function is included in the documentation (C8 of Tab.1)
# Step 6: Adapt the FA using the locally stored information
TCP.tc_plus_adapt(net=net, fsp_load_indices=fsp_load_indices, fsp_dg_indices=fsp_dg_indices)
# Step 7: Estimate the FA without adapting to compare with the above-adapted result
TCP.tc_plus(net=net, fsp_load_indices=fsp_load_indices, fsp_dg_indices=fsp_dg_indices, dp=0.05, dq=0.1)
```

The expected output FA from the storing function is the same as in Fig. 4(a). However, this function also stores:

1. TTD results for 20 network components with total size 241 MB.
2. FA axes values with total size 2 KB.
3. Matrix of the unconstrained convolution results with total size 4 KB.
4. Dictionary with FSP impacts 382 KB.
5. Dictionary of impactful FSPs per network component 4 KB.

The storing function required 61 s. Fig. 5 illustrates the expected output for the adapted FA and the FA without adaptation, i.e., not using the stored information. The FAs of Fig. 5 have a high resemblance. The GPU needed 1.4 s for the adapted FA of Fig. 5(a) and 10.4 s for the FA of Fig. 5(b).



(a) Adapted FA. (b) FA witouth adaptation.

Fig. 5. TensorConvolution+ example in adapted FA estimation.

5. Impact

TensorConvolutionPlus is the first open-source package for FA estimation. Users can execute the package functionalities with ease. The package can estimate FAs only with two lines of Python code, importing the package and calling the selected functionality. Existing power systems-related libraries do not offer FA estimation functionalities, but rather power flows and optimal power flows. Other FA frameworks have been proposed to make use of such power flows. However, to the best of the authors knowledge, this proposed package is the first that focuses directly on the FA estimation algorithms and is ready to use without requiring the implementation of FA algorithms. Existing works have released code implementations for FA estimation [16,40,41], but no dedicated FA package has been available. This package release enhances FA estimation accessibility, reusability, and ease of integration, providing a standardized and user-friendly solution. Nevertheless, the structure also allows using networks developed by the user. The software design strengthens the potential for further expansion, improvement, and adoption of FA estimation methods. The package structure enables users to expand FA estimation subprocesses without modifying other subprocesses. For example, possible expansions for the PF-based functions include new shapes from the flexibility resources or new sampling distributions for the Monte Carlo-based function. These expansions should not impact Algorithm 2 but only the scripts *data_sampler* for the new sampling shapes or distributions, *json_reader* to accept the new input keywords, and *FA_Estimator* to obtain the new inputs and call the new *data_sampler* functions.

The developed package includes different FA estimation approaches, allowing users to select and identify the best-performing approach for their tasks. Nevertheless, this package focuses on the TensorConvolution+ [17] algorithm, which can require more complex implementation compared to OPF-based and PF-based algorithms. Through this package, researchers will be able to familiarize themselves with the FA estimation topic and the TensorConvolution+ algorithm and further advance the field of FA estimation. Similarly, power system operators can use this package directly for their networks and case studies, improving the potential of adopting FAs in their operations. Recent publications explore the application of FA for improved TSO-DSO coordination frameworks and more efficient real-time grid operation [3,23]. The proposed package can further facilitate these applications by alleviating the burden of reimplementing the FA estimation algorithms. Existing power system workflows in Python can integrate with the proposed package if the systems are in a pandapower format. Power system workflows in other languages or software such as DigSilent or Matlab would first require integration with Python before using TensorConvolutionPlus.

5.1. Limitations

The main limitation of the released package is the absence of unit testing for future package versions. Individual limitations from the main functionalities include the GPU memory overflow for TensorConvolution+ and convergence failures for the OPF-based functionality. For TensorConvolution+, the memory overflow can be reduced by using the *tc_plus_merge* functionality. Another mitigation can extend the increments dp, dq, resulting in lower resolution FAs. For the OPF-based convergence failures, simplified OPF implementations might mitigate the issues, but were not tested in the present package release.

6. Conclusions

Power system digitalization and developing open-source power system specialized tools are significant for intelligent and effective power grid operations. In the absence of open-source tools for FA estimation, the developed package can improve the reachability and adoption of TensorConvolution+ and FA estimation algorithms in academia and industry. With a user-friendly structure, the package allows straightforward installation and execution of FA estimation.

The package documentation showcases example usages and details on the scripts and their functions. The package structure diversifies between FA sub-processes. This diversification allows users to better understand and expand the FA estimation algorithms in specific sub-processes, such as the FSP shapes, without impacting the remaining sub-processes. This research is part of the MegaMind project which involves Dutch system operators and grid companies. The package will be pursued for integration with relevant industrial partners in future developments. Unit testing implementation will also be pursued in future work to facilitate consistency in later package versions. Future package expansions include implementing FA functionalities using alternative power system tools and libraries, and comparing the performance to the present functionalities. This package will be maintained with updates reviewed annually. Pull requests are welcome and will be tested and reviewed by the authors before merging. Improvement on the GPU requirements of TensorConvolution+ and the OPF implementations will also be pursued in future developments.

CRediT authorship contribution statement

Demetris Chrysostomou: Writing – review & editing, Writing – original draft, Visualization, Validation, Software, Resources, Methodology, Investigation, Formal analysis, Data curation, Conceptualization. **José Luis Rueda Torres:** Writing – review & editing, Writing – original draft, Validation, Supervision, Project administration, Methodology, Conceptualization. **Jochen Lorenz Cremer:** Writing – review & editing, Writing – original draft, Validation, Supervision, Project administration, Methodology, Conceptualization.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This research is part of the research program ‘MegaMind - Measuring, Gathering, Mining and Integrating Data for Self-management in the Edge of the Electricity System’, (partly) financed by the Dutch Research Council (NWO) through the Perspectief program under number P19-25.

References

[1] Rodríguez Pérez N, Domingo JM, López GL, et al. ICT architectures for TSO-DSO coordination and data exchange: A European perspective. IEEE Trans Smart Grid 2023;14(2). http://dx.doi.org/10.1109/TSG.2022.3206092.
[2] Troncia M, Ruggeri S, Soma GG, Pilo F, Ávila JPC, Muntoni D, et al. Strategic decision-making support for distribution system planning with flexibility alternatives. Sustain Energy, Grids Networks 2023;35:101138. http://dx.doi.org/10.1016/j.segan.2023.101138.
[3] Givisiez AG, Petrou K, Ochoa LF. A review on TSO-DSO coordination models and solution techniques. Electr Power Syst Res 2020;189.
[4] Weng Y, Xie J, Wang P, Nguyen HD. Asymmetrically reciprocal effects and congestion management in TSO-DSO coordination through feasibility regularizer. IEEE Trans Power Syst 2023;38(2):1948–62. http://dx.doi.org/10.1109/TPWRS.2022.3193052.
[5] Bakhtiari H, Hesamzadeh MR, Bunn DW. TSO-DSO operational coordination using a look-ahead multi-interval framework. IEEE Trans Power Syst 2023;38(5):4221–39. http://dx.doi.org/10.1109/TPWRS.2022.3219581.
[6] Silva J, Sumaili J, Bessa RJ, Seca L, Matos M, Miranda V. The challenges of estimating the impact of distributed energy resources flexibility on the TSO/DSO boundary node operating points. Comput Oper Res 2018;96.
[7] Capitanescu F. TSO-DSO interaction: Active distribution network power chart for TSO ancillary services provision. Electr Power Syst Res 2018;163.
[8] Kalantar-Neyestanaki M, Sossan F, Bozorg M, Cherkaoui R. Characterizing the reserve provision capability area of active distribution networks: A linear robust optimization method. IEEE Trans Smart Grid 2019;11(3).
[9] Savvopoulos N, Evrenosoglu CY, Konstantinou T, Demiray T, Hatziargyriou N. Contribution of residential PV and BESS to the operational flexibility at the TSO-DSO interface. In: International conference on smart energy systems and technologies. SEST, IEEE; 2021.
[10] Gonzalez DM, Hachenberger J, Hinker J, Rewald F, et al. Determination of the time-dependent flexibility of active distribution networks to control their TSO-DSO interconnection power flow. In: Power systems computation conference. PSCC, IEEE; 2018.
[11] Savvopoulos N, Hatziargyriou N. An effective method to estimate the aggregated flexibility at distribution level. IEEE Access 2023;11. http://dx.doi.org/10.1109/ACCESS.2023.3262730.
[12] Prionistis G, Vournas C, Vrakopoulou M. A fast method to approximate the flexibility region of an active distribution network in PQ space. In: IEEE belgrade powerTech. 2023, http://dx.doi.org/10.1109/PowerTech55446.2023.10202983.
[13] Churkin A, Sanchez-Lopez M, Alizadeh MI, Capitanescu F, et al. Impacts of distribution network reconfiguration on aggregated DER flexibility. In: IEEE belgrade powerTech. 2023, http://dx.doi.org/10.1109/PowerTech55446.2023.10202791.
[14] Chen T, Song Y, Hill DJ, Lam AY. Enhancing flexibility at the transmission-distribution interface with power flow routers. IEEE Trans Power Syst 2021;37(4).
[15] Bolfek M, Capuder T. An analysis of optimal power flow based formulations regarding DSO-TSO flexibility provision. Int J Electr Power Energy Syst 2021;131.
[16] Chrysostomou D, Torres JLR, Cremer JL. Exploring operational flexibility of active distribution networks with low observability. In: 2023 IEEE belgrade powertech. IEEE; 2023.
[17] Chrysostomou D, Torres JLR, Cremer JL. Tensor convolution-based aggregated flexibility estimation in active distribution systems. IEEE Trans Smart Grid 2024. http://dx.doi.org/10.1109/TSG.2024.3453667.
[18] Savvopoulos N, Hatziargyriou N, Laaksonen H. A holistic approach to the efficient estimation of operational flexibility from distributed resources. IEEE Open Access J Power Energy 2024.
[19] Majumdar N, Kengkat P, Yermekbayev R, Hofmann L. Reliability parameterised distribution grid flexibility aggregation considering renewable uncertainties. In: 2023 58th international universities power engineering conference. UPEC, 2023, p. 1–6. http://dx.doi.org/10.1109/UPEC57427.2023.10294524.
[20] Ge S, Xu Z, Liu H, Gu C, Li F. Flexibility evaluation of active distribution networks considering probabilistic characteristics of uncertain variables. IET Gener Transm Distrib 2019;13(14):3148–57.
[21] Hui H, Bao M, Ding Y, Yan J, Song Y. Probabilistic integrated flexible regions of multi-energy industrial parks: Conceptualization and characterization. Appl Energy 2023;349:121521.
[22] Silva J, Sumaili J, Bessa RJ, Seca L, Matos MA, Miranda V, et al. Estimating the active and reactive power flexibility area at the TSO-DSO interface. IEEE Trans Power Syst 2018;33(5).
[23] Kalantar-Neyestanaki M, Cherkaoui R. Grid-cognizant TSO and DSO coordination framework for active and reactive power flexibility exchange: The swiss case study. Electr Power Syst Res 2024;235:110747. http://dx.doi.org/10.1016/j.epsr.2024.110747.
[24] Cali U, Kuzlu M, Pipattanasomporn M, Kempf J, Bai L. Introduction to the digitalization of power systems and markets. Digit Power Mark Syst using Energy Informatics 2021.
[25] Fan R, Yin T, Yang K, Lian J, Buckheit J. New data-driven approach to bridging power system protection gaps with deep learning. Electr Power Syst Res 2022;208:107863.

[26] Ahmad T, Madonski R, Zhang D, Huang C, Mujeeb A. Data-driven probabilistic machine learning in sustainable smart energy/smart energy systems: Key developments, challenges, and future research opportunities in the context of smart grid paradigm. Renew Sustain Energy Rev 2022;160:112128.

[27] Habib B, Isufi E, Breda Wv, Jongepier A, Cremer JL. Deep statistical solver for distribution system state estimation. IEEE Trans Power Syst 2023. http://dx.doi.org/10.1109/TPWRS.2023.3290358.

[28] Cremer JL, Strbac G. A machine-learning based probabilistic perspective on dynamic security assessment. Int J Electr Power Energy Syst 2021;128:106571.

[29] Di Silvestre ML, Favuzza S, Riva Sanseverino E, Zizzo G. How decarbonization, digitalization and decentralization are changing key power infrastructures. Renew Sustain Energy Rev 2018;93. http://dx.doi.org/10.1016/j.rser.2018.05.068.

[30] Thurner L, Scheidler A, Schäfer F, Menke J-H, Dollichon J, Meier F, et al. Pandapower—an open-source python tool for convenient modeling, analysis, and optimization of electric power systems. IEEE Trans Power Syst 2018;33(6).

[31] Milano F. An open source power system analysis toolbox. IEEE Trans Power Syst 2005;20(3).

[32] Zimmerman RD, Murillo-Sánchez CE, Thomas RJ. MATPOWER: Steady-state operations, planning, and analysis tools for power systems research and education. IEEE Trans Power Syst 2011;26(1). http://dx.doi.org/10.1109/TPWRS.2010.2051168.

[33] Xiang Y, Salemink P, Stoeller B, Bharambe N, van Westering W. Power grid model: a high-performance distribution grid calculation library. In: 27th international conference on electricity distribution, Vol. 2023. 2023, http://dx.doi.org/10.1049/icp.2023.0633.

[34] Johnston J, Henriquez-Auba R, Maluenda B, Fripp M. Switch 2.0: A modern platform for planning high-renewable power systems. SoftwareX 2019;10. http://dx.doi.org/10.1016/j.softx.2019.100251.

[35] Mirz M, Vogel S, Reinke G, Monti A. Dpsim—A dynamic phasor real-time simulator for power systems. SoftwareX 2019;10. http://dx.doi.org/10.1016/j.softx.2019.100253.

[36] Plietzsch A, Kogler R, Auer S, Merino J, de Muro AG, Lißе J, et al. PowerDynamics.jl—An experimentally validated open-source package for the dynamical analysis of power grids. SoftwareX 2022;17. http://dx.doi.org/10.1016/j.softx.2021.100861.

[37] Ramos A, Alvarez EF, Lumbreras S. openTEPES: open-source transmission and generation expansion planning. SoftwareX 2022;18.

[38] Brandl G. Sphinx documentation. 2010, URL http://sphinx-doc.org/sphinx.pdf.

[39] Bisong E, Bisong E. Google colaboratory. 2019, Building machine learning and deep learning models on google cloud platform: a comprehensive guide for beginners.

[40] Papazoglou GK, Forouli AA, Bakirtzis EA, Biskas PN, Bakirtzis AG. Estimating the feasible operating region of active distribution networks using the genetic algorithm. In: 2023 IEEE PES gTD international conference and exposition. GTD, IEEE; 2023, p. 182–7.

[41] Churkin A, Kong W, Mancarella P, Ceseña EAM. Quantifying phase unbalance and coordination impacts on distribution network flexibility. 2024, arXiv preprint arXiv:2408.06516.