Nempy Technical Brief v1.0.0

Nicholas Gorman, Iain MacGill and Anna Bruce

Correspondence via n.gorman@unsw.edu.au

Introduction

Nempy is an open-source python package that can be used to model the dispatch procedure of the Australian National Electricity Market (NEM). We believe nempy will be a valuable tool for academic researchers and industry analysts exploring a wide variety of questions. The dispatch process is at the core of many market modelling projects. As the NEM evolves, constraints and ancillary service markets are becoming increasingly important in shaping dispatch outcomes. Significant changes to the dispatch process are also likely to occur soon. Nempy allows users to easily configure a dispatch model to fit the relevant research question. Furthermore, if extra functionality is needed, the python implementation, open-source licencing and planned ongoing support from developers make it possible to adapt nempy to your needs. The publication of this technical brief coincides with the release of nempy version 1.0.0. Version 1 will be a stable release and ongoing minor updates or patches will remain backwards compatible. However, we are interested in receiving feedback to inform ongoing maintenance, development and any future major updates.

Nempy is feature rich, flexible, can recreate historical dispatch with a high degree of accuracy, runs fast, has detailed documentation and has planned support until mid-2023.

Find nempy and more information at https://github.com/UNSW-CEEM/nempy.

Features

- **Energy bids**: between one and ten price quantity bid pairs can be provided for each generator or load bidding in the energy market
- **Loss factors**: loss factors can be provided for each generator and load
- **FCAS bids**: between one and ten price quantity bid pairs can be provided for each generator or load bidding in each of the eight FCAS markets
- **Ramp rates**: unit ramp rates can be set
- **FCAS trapezium constraints**: a set of trapezium constraints can be provided for each FCAS bid, these ensure FCAS is co-optimised with energy dispatch and would be technically deliverable
- **Fast start dispatch inflexibility profiles**: dispatch inflexibility profiles can be provided for unit commitment of fast-start plants
- **Interconnectors and losses**: interconnectors between each market region can be defined, non-linear loss functions and interpolation breakpoints for their linearisation can be provided
- **Generic constraints**: generic constraints that link across unit output, FCAS enablement and interconnector flows can be defined

- **Elastic constraints**: constraints can be made elastic, i.e. a violation cost can be set for constraints
- **Tie-break constraints**: constraints that minimise the difference in dispatch between energy bids for the same price can be enabled
- **Market clearing prices**: market prices are returned for both energy and FCAS markets, based on market constraint shadow prices
- **Historical inputs**: tools for downloading dispatch inputs from AEMO's NEMWeb portal and preprocessing them for compatibility with the nempy SpotMarket class are available
- **Input validation**: optionally check user inputs and raise descriptive errors when they do not meet the expected criteria
- **Adjustable dispatch interval**: a dispatch interval of any length can be used

Flexibility

Nempy is designed to have a high degree of flexibility, it can be used to implement very simple merit order dispatch models, highly detailed models that seek to re-create the real world dispatch procedure, or a model at the many levels of intermediate complexity. A set of examples demonstrating this flexibility are available on the package's documentation readthedocs page. Most inputs are passed to nempy as pandas DataFrame objects, which means nempy can easily source inputs from other python code, SQL databases, CSVs and other formats supported by the pandas' interface.

Accuracy

The accuracy with which nempy represents the NEM's dispatch process can be measured by re-creating historical dispatch results. This is done for a given dispatch interval by downloading the relevant historical inputs such as unit initial operating levels, bids and generic constraints, processing these inputs so they are compatible with the nempy SpotMarket class, and finally dispatching the spot market. The results can then be compared to historical results to gauge the model's accuracy. Figure 1 shows the results of this process for 1000 randomly selected dispatch intervals in 2019, comparing the modelled NSW energy price with historical prices. Here the model is configured to maximally reflect the NEM's dispatch procedure. The code to produce the results shown in this figure is available here. Figure 2 shows a similar comparison, but without FCAS markets or generic constraints. The code to produce the results shown in Figure 2 is available here. The simpler model produces a similar number of medianly priced intervals, however, outcomes for extreme ends of the price duration curve differ significantly from historical values.

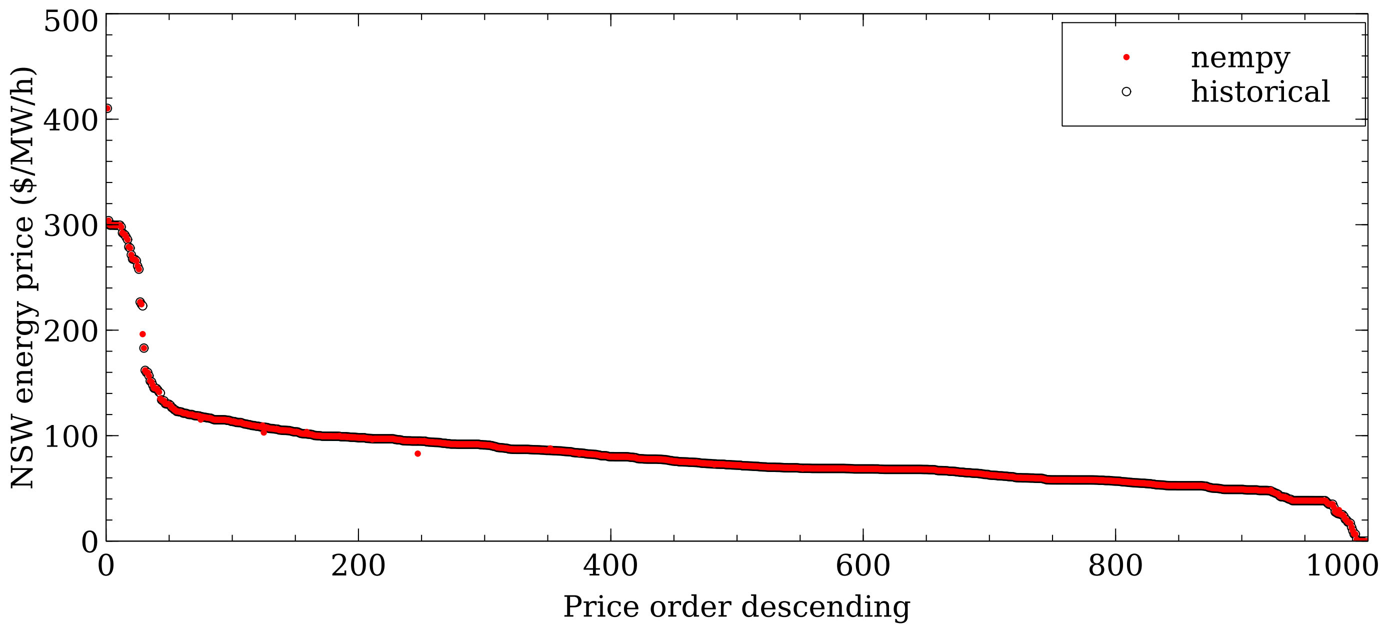

Figure 1: A comparison of the historical NSW reference node price, prior to scaling or capping, with the price calculated using nempy. The nempy model was configured to maximally replicated the NEM dispatch process and 1000 randomly selected intervals were used.

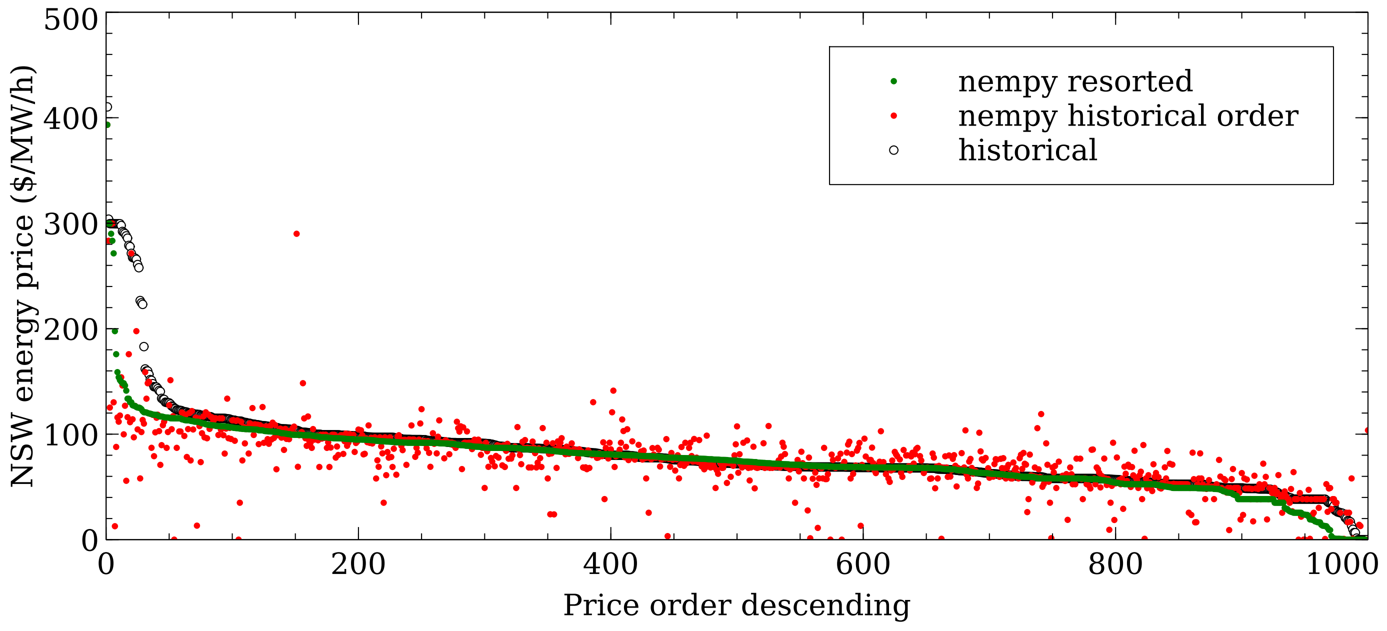

Figure 2: A comparison of the historical NSW reference node price, prior to scaling or capping, with the price calculated using nempy. The nempy model was configured without FCAS markets or generic constraints and 1000 randomly selected intervals were used.

Run-time

The run-time for nempy to calculate dispatch depends on several factors, the complexity of the model implemented, time taken to load inputs, the mixed-integer linear solver used and of course the hardware. Run-times reported here used an Intel® Xeon(R) W-2145 CPU @ 3.70 GHz. For the model results shown in Figure 1, including time taken to load inputs from the disk and using the open-source solver CBC, the average run-time per dispatch interval was 2.54 s. When the proprietary solver Gurobi was used, a run-time of 1.84 s was achieved. For the results shown in Figure 2, the run-times with CBC and Gurobi were 1.02 s and 0.98 s respectively, indicating that for simpler models the solver used has a smaller impact on run-time. For the simpler model, the time to load inputs is increased significantly by the loading of historical NEMDE input/output XML files which takes approximately 0.4 s. Importantly, this means it will be possible to speed up simpler models by sourcing inputs from different data storage formats.

Documentation

Nempy has a detailed set of documentation, mainly comprising of two types: examples and reference documentation. The examples aim to show how nempy can be used and how it works in a practical manner. A number of simple examples focus on demonstrating the use of subsets of the package's features in isolation in order to make them easier to understand. The more complex examples show how features can be combined to build models more suitable for analysis. The reference documentation aims to cover all the package's public APIs (the classes, methods and functions accessible to the user), describing their use, inputs, outputs and any side effects. Find the documentation on our readthedocs page.

Support

Nempy's development is being led by Nick Gorman as part of his PhD candidature at the Collaboration on Energy and Environmental Markets at the University of New South Wales' School of Photovoltaics and Renewable Energy Engineering. As part of this project we plan to engage with and support software users, this can be facilitated through the PhD until mid-2023. If nempy is used sufficiently broadly we would look to continue support beyond this timeframe.

Ongoing work

Ongoing work is likely to focus on greater support for time-sequential and dynamic models through the creation of tools that dynamically create inputs for the dispatch process. This would likely include tools for dynamically generating participant bids and regional FCAS requirements. Additionally, many historical dispatch generic constraints are potentially unsuitable for dynamic modelling as their right-hand sides contain line flow values and unit operating statuses that would be expected to change under many scenarios in a dynamic model. Thus, another avenue for future work is the addition of tools for creating constraint inputs appropriate for dynamic modelling.