Journal of
open research software

Brown, T, et al. 2018 PyPSA: Python for Power System Analysis. *Journal of Open Research Software,* 6: 4.
DOI: https://doi.org/10.5334/jors.188

SOFTWARE METAPAPER

PyPSA: Python for Power System Analysis

Tom Brown, Jonas Hörsch and David Schlachtberger
Frankfurt Institute for Advanced Studies, Ruth-Moufang-Straße 1, 60438 Frankfurt am Main, DE
Corresponding author: Tom Brown (brown@fias.uni-frankfurt.de)

Python for Power System Analysis (PyPSA) is a free software toolbox for simulating and optimising modern electrical power systems over multiple periods. PyPSA includes models for conventional generators with unit commitment, variable renewable generation, storage units, coupling to other energy sectors, and mixed alternating and direct current networks. It is designed to be easily extensible and to scale well with large networks and long time series. In this paper the basic functionality of PyPSA is described, including the formulation of the full power flow equations and the multi-period optimisation of operation and investment with linear power flow equations. PyPSA is positioned in the existing free software landscape as a bridge between traditional power flow analysis tools for steady-state analysis and full multi-period energy system models. The functionality is demonstrated on two open datasets of the transmission system in Germany (based on SciGRID) and Europe (based on GridKit).

Keywords: Power system simulations; energy system simulations; Load flow calculations; optimal power flow; security-constrained optimal power flow; unit commitment; renewable energy
Funding statement: This research was conducted as part of the CoNDyNet project, which is supported by the German Federal Ministry of Education and Research under grant no. 03SF0472C. The responsibility for the contents lies solely with the authors.

(1) Overview

1. Introduction

Power system tools model the interactions between the electrical grid and the consumers and generators which use the grid. The importance of software modelling of the grid has risen in recent years given the increase in distributed and fluctuating wind and solar generation, and the increasing electrification of all energy demand. On the generation side, variable renewable generation causes loading in parts of the grid where it was never expected, and introduces new stochastic influences on the flow patterns. On the demand side, the need to decarbonise the transport and heating sectors is leading to the electrification of these sectors and hence higher electrical demand, replacing internal combustion engines with electric motors in the transport sector, and replacing fossil fuel boilers with heat pumps, resistive heaters and cogeneration for low-temperature space and water heating. In addition, the increasing deployment of storage technologies introduces many network users which are both consumers and generators of energy.

The increasing complexity of the electricity system requires new tools for power system modelling. Many of the tools currently used for power system modelling were written in the era before widespread integration of renewable energy and the electrification of transport and heating. They therefore typically focus on network flows in single time periods. Examples of such tools include commercial products like DIgSILENT PowerFactory [1], NEPLAN [2], PowerWorld [3], PSS/E [4] and PSS/SINCAL [5], and open tools such as MATPOWER [6], PSAT [7], PYPOWER [8] and pandapower [9] (see [10] for a full list of power system analysis tools).

The consideration of multiple time periods is important on the operational side for unit commitment of conventional generators and the optimisation of storage and demand side management, and on the investment side for optimising infrastructure capacities over representative load and weather situations. Several tools have subsets of these capabilities, such as calliope [11], manpower [12], MOST [13], oemof [14], OSeMOSYS [15], PLEXOS [16], PowerGAMA [17], PRIMES [18], TIMES [19] and urbs [20], but their representations of electrical grids are often simplified.

Python for Power System Analysis (PyPSA), the tool presented in this paper, was developed at the Frankfurt Institute for Advanced Studies to bridge the gap between power system analysis software and general energy system modelling tools. PyPSA can model the operation and optimal investment of the energy system over multiple periods. It has models for the unit commitment of conventional generators, time-varying renewable generators, storage units, all combinations of direct and alternating current electricity networks, and the coupling

of electricity to other energy sectors, such as gas, heating and transport. It can perform full load flow calculations and linearised optimal load flow, including under consideration of security constraints. It was written from the start with variable renewables, storage and sector-coupling in mind, so that it performs well with large networks and long time series.

Given the complexity of power system tools and the different needs of different users, it is crucial that such tools are both transparent in what they do and easily extendable by the user. To this end, PyPSA was released as free software under the GNU General Public Licence Version 3 (GPLv3) [21]. This means that the user is free to inspect, use and modify the code, provided that if they redistribute the software, they also provide the source code. Free software and open data also guarantee that research results can be reproduced by any third party, which is important given the large investment decisions that will need to be made on the basis of energy system modelling to reduce greenhouse gas emissions and combat global warming [22], [23].

PyPSA is available online in the Python Package Index (PyPI), on GitHub [24] and is archived on Zenodo [25]. Documentation and examples are available on PyPSA's website [26]. PyPSA is already used by more than a dozen research institutes and companies worldwide, 70 people are registered on the forum [27] and the website [26] has been visited by people from over 160 countries. As of October 2017 it has been used in six research papers [28, 29, 30, 31, 32, 33]. Users have already extended PyPSA for integer transmission expansion [28, 34] and in the grid planning tool open_eGo [35].

This paper describes version 0.11.0 of PyPSA [25]. In Section 2 the mathematical functionality of PyPSA is described, while in Section 3 the focus shifts to the implementation in software. Quality control is discussed in Section 4; the computational performance of PyPSA is described in Section 5; and then its functionality is compared with other software in Section 6. Several example applications are given in 7 before conclusions are drawn in 8.

2. Functionality

In this section the basic components, power flow, linear optimal power flow, energy system optimisation, unit commitment, contingency modelling and other functionality of PyPSA are described. The definitions of the main variables used in this section can be found in **Table 1**, along with units where applicable.

2.1. Components

PyPSA's representation of the power system is built by connecting the components listed in **Table 2**.

Buses are the fundamental nodes to which all other components attach. Their mathematical role is to enforce energy conservation at the bus at all times (essentially Kirchhoff's Current Law).

Loads, generators, storage units, stores and shunt impedances attach to a single bus and determine the power balance at the bus. Loads represent a fixed power demand; a generator's dispatch can be optimised within its power availaiblity; stores can shift power from one time to another with a standing loss efficiency for energy leakage; storage units behave like stores, but they can also have efficiency losses and power limits upon charging and discharging; finally shunt impedances have a voltage-dependent power consumption.

Lines and transformers connect two buses with a given impedance. Power flows through lines and transformers according to the power imbalances at the buses and the impedances in the network. Lines and transformers are referred to collectively as 'passive branches' to distinguish them from controllable link branches. The impedances of the passive branches are modeled internally using the equivalent PI model. The relation between the series impedance $z = r + jx$, the shunt admittance $y = g + jb$, the transformer tap ratio τ, the transformer phase shift θ^{shift}, and the complex currents I_0, I_1 and complex voltages V_0, V_1 at the buses labelled 0 and 1 is given by

$$\begin{pmatrix} I_0 \\ I_1 \end{pmatrix} = \begin{pmatrix} \frac{1}{z} + \frac{y}{2} & -\frac{1}{z}\frac{1}{\tau e^{-j\theta^{\text{shift}}}} \\ -\frac{1}{z}\frac{1}{\tau e^{j\theta^{\text{shift}}}} & \left(\frac{1}{z} + \frac{y}{2}\right)\frac{1}{\tau^2} \end{pmatrix} \begin{pmatrix} V_0 \\ V_1 \end{pmatrix} \quad (1)$$

(For lines, for which neither the tap ratio or the phase shift are relevant, set $\tau = 1$ and $\theta^{\text{shift}} = 0$ in this equation.) The equivalent circuit is shown in **Figure 1**. This circuit is for the case where the tap-changer is on the primary side; a similar equation and figure for the case where the tap-changer is on the secondary side is given in the documentation [26]. The line model defaults to the PI model, while the transformer model defaults to the more accurate T model, which is converted to the PI model using the standard delta-wye transformation. For convenience standard types for lines and transformers in networks at 50 Hz are provided following the conversion formula from nameplate parameters to impedances and the typical parameters provided in pandapower [9], so that the user does not have to input the impedances manually. The typical parameters in pandapower are based on [36, 37, 38].

Links connect two buses with a controllable active power dispatch that can be set by the user or optimised by PyPSA. Links can be used to represent point-to-point high voltage direct current (HVDC) lines, import-export capacities in transport models such as Net-Transfer-Capacity (NTC) models, or general energy conversion processes with a given efficiency, such as resistive heaters or heat pumps (from electricity to heat) or gas boilers (from gas to heat). Their efficiency can also be time-varying (e.g. to represent the ambient temperature dependence of a heat pump's coefficient of performance). Networks of links implement Kirchoff's Current Law (energy conservation at each bus), but not Kirchoff's Voltage Law, which is obeyed by networks of passive branches.

A generator can also be represented in terms of more basic components: a bus is added for the fuel source with a store to represent the amount of fuel available. It is then

Table 1: Nomenclature.

Variable	Units	Definition
n, m		Bus labels
r		Generator energy carrier labels (e.g. wind, solar, gas, etc.)
s		Storage energy carrier labels (e.g. battery, hydrogen, etc.)
κ, ℓ		Branch labels
c		Cycle labels
t		Snapshot/time point labels
$e_{r/s}$	tCO_2eq/MWh_{th}	CO_2-equivalent emissions of energy carrier r or s
w_t	h	Weighting of snapshot in objective function
$g_{n,r,t}$	MW	Dispatch of generator at bus n with carrier r at time t
$G_{n,r}$	MW	Power capacity of generator n, r
$\bar{g}_{n,r,t}$	MW/MW	Power availability per unit of generator capacity
$\eta_{n,r}$	MW_{el}/MW_{th}	Efficiency of generator
$u_{n,r,t}$		On/off binary status for generator unit commitment
$T_{n,r}^{\min\_down}$	h	Generator minimum down time
$T_{n,r}^{\min\_up}$	h	Generator minimum up time
$ru_{n,r}$	(MW/MW)/h	Generator ramp up limit per unit of capacity
$rd_{n,r}$	(MW/MW)/h	Generator ramp down limit per unit of capacity
$c_{n,r}$	€/MW	Generator capital (fixed) cost
$o_{n,r}$	€/MWh	Generator operating (variable) cost
$suc_{n,r(,t)}$	€	Generator start up cost (in time t)
$sdc_{n,r(,t)}$	€	Generator shut down cost (in time t)
$h_{n,s,t}$	MW	Dispatch of storage at bus n with carrier s at time t
$H_{n,s}$	MW	Power capacity of storage n, s
$e_{n,s,t}$	MWh	Storage state of charge (energy level)
$E_{n,s}$	MWh	Storage energy capacity
$c_{n,s}$	€/MW	Storage power capacity cost
$\hat{c}_{n,s}$	€/MWh	Storage energy capacity cost
$o_{n,s}$	€/MWh	Storage dispatch cost
$d_{n,t}$	MW	Electrical load at bus n at time t
$\lambda_{n,t}$	€/MWh	Marginal price at bus n at time t
V_n	kV	Complex voltage at bus n
θ_n	rad	Voltage angle at bus n
I_n	kA	Complex current at bus n
P_n	MW	Total active power injection at bus n
Q_n	MVAr	Total reactive power injection at bus n
S_n	MVA	Total apparent power injection at bus n
$f_{\ell,t}$	MW	Branch active power flow
F_ℓ	MW	Branch active power rating
c_ℓ	€/MW	Branch capital cost
x_ℓ	Ω	Branch series reactance
r_ℓ	Ω	Branch series resistance

(contd.)

Variable	Units	Definition
z_ℓ	Ω	Branch series impedance
y_ℓ	S	Branch shunt admittance
τ_ℓ		Transformer tap ratio
$\theta_\ell^{\text{shift}}$	rad	Transformer phase shift
$\eta_{\ell,t}$	MW/MW	Efficiency loss of a link
$K_{n\ell}$		$N \times L$ incidence matrix
$C_{\ell c}$		$L \times (L - N + 1)$ cycle matrix
Y_{nm}	S	Bus admittance matrix
$B_{\ell k}$	S	Diagonal $L \times L$ matrix of branch susceptances
$BODF_{\ell k}$		Branch Outage Distribution Factor

Table 2: PyPSA components.

Network	Container for all other network components.
Bus	Fundamental nodes to which all other components attach.
Carrier	Energy carrier (e.g. wind, solar, gas, etc.).
Load	A consumer of energy.
Generator	Generator whose feed-in can be flexible subject to minimum loading or minimum down and up times, or variable according to a given time series of power availability.
Storage Unit	A device which can shift energy from one time to another, subject to efficiency losses.
Store	A more fundamental storage object with no restrictions on charging or discharging power.
Shunt Impedance	An impedance in shunt to a bus.
Line	A branch which connects two buses of the same voltage.
Transformer	A branch which connects two buses of different voltages.
Link	A branch with a controllable power flow between two buses.

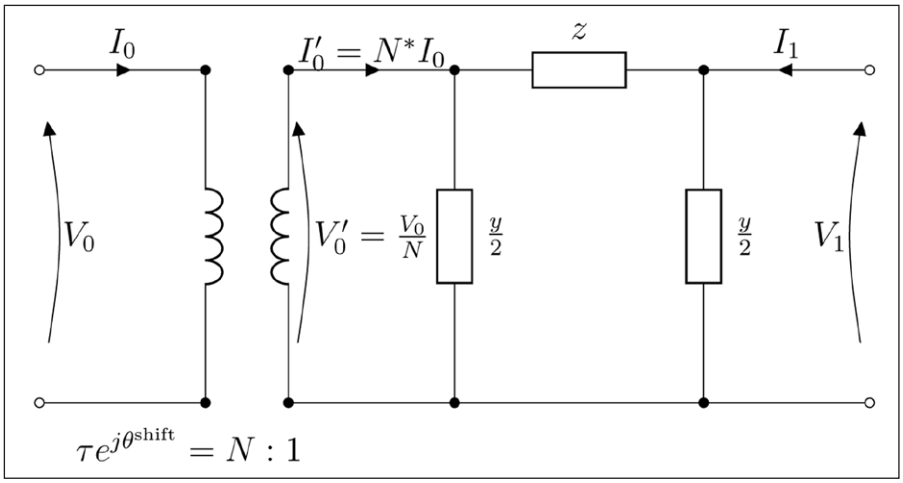

Figure 1: Electrical property definitions for passive branches (lines and transformers).

connected to the electricity bus with a link to represent the energy conversion loss. Similarly a storage unit can be represented with an additional bus for the storage medium with a store attached, and then two links connected to the electricity bus to represent charging and discharging.

Energy enters the model in generators; in storage units or stores with higher energy levels before than after the simulation; and in any components with efficiency greater than 1 (such as heat pumps). Energy leaves the model in loads; in storage units or stores with higher energy levels

after than before the simulation; and in lines, links or storage units with efficiency less than 1.

2.2. Power flow without optimisation

In a power flow calculation, the user specifies the power dispatch of all dispatchable components (loads, generators, storage units, stores and links) and then PyPSA computes the resulting voltages in the network and hence the power flows in passive branches (lines and transformers) based on their impedances.

2.2.1. Power flow equations for AC networks

A power flow calculation for an alternating current (AC) network ensures that for all buses labelled by n we have

$$S_n = V_n I_n^* = \sum_m V_n Y_{nm}^* V_m^* \tag{2}$$

where $S_n = P_n + jQ_n$ is the apparent power injected at the bus, I_n is the complex current and $V_n = |V_n| e^{j\theta_n}$ is the complex voltage, whose rotating angle is measured relative to a chosen slack bus. Y_{nm} is the bus admittance matrix, which is constructed for all buses based on the contributions from the individual branch admittance matrices from equation (1) and any shunt impedances at the nodes, following the example of MATPOWER [6].

The inputs and outputs for the buses are given as follows:

- For the chosen slack bus $n = 0$, it is assumed that the voltage magnitude $|V_0|$ and the voltage angle θ_0 are given. PyPSA must find the powers P_0 and Q_0.
- For *PQ* buses, P_n and Q_n are given; $|V_n|$ and θ_n are to be found.
- For *PV* buses, P_n and $|V_n|$ are given; Q_n and θ_n are to be found.

The non-linear equation system (2) is then solved using the Newton-Raphson algorithm [39] and, by default, an initial 'flat' guess of $\theta_n = \theta_0$ and $|V_n| = 1$ (per unit). The initial guess can also be specified ('seeded') by the user, using for example the linearised power flow solution.

2.2.2. Power flow equations for DC networks

A power flow calculation for a direct current (DC) network ensures that for all buses labelled by n we have

$$P_n = V_n I_n = \sum_m V_n G_{nm} V_m \tag{3}$$

where P_n is the active power injected at the bus and the voltage, current and the conductance matrix G_{ij} are now all real quantities. This non-linear equation is also solved with the Newton-Raphson algorithm.

2.2.3. Linearised power flow equations for AC networks

In some circumstances a linearisation of the AC power flow equations (2) can provide a good approximation to the full non-linear solution [40, 41]. The linearisation is restricted to calculating active power flows based on voltage angle differences and branch series reactances. It assumes that reactive power flow decouples from active power flow, that there are no voltage magnitude variations, voltage angles differences across branches are small enough that $\sin\theta \sim \theta$ and branch resistances are negligible compared to branch reactances. This makes it suitable for short overhead transmission lines close to their natural loading.

In this case it can be shown [6] that the voltage angles are related to the active power injections by a matrix

$$P_n = \sum_m \left(KBK^T\right)_{nm} \theta_m - \sum_\ell K_{n\ell} b_\ell \theta_\ell^{\text{shift}} \tag{4}$$

where K is the incidence matrix of the network, B is the diagonal matrix of inverse branch series reactances x_ℓ multiplied by the tap ratio τ_ℓ, i.e. $B_{\ell\ell} = b_\ell = \frac{1}{x_\ell \tau_\ell}$, and $\theta_\ell^{\text{shift}}$ is the phase shift for a transformer. The matrix KBK^T is singular with a single zero eigenvalue for a connected network and can be inverted by first deleting the row and column corresponding to the slack bus.

2.2.4. Linearised power flow equations for DC networks

For DC networks the equation (3) is linearised by positing $V_n = 1 + \delta V_n$ and assuming that δV_n is small. The resulting equations mirror the linearised AC approximation with the substitutions $\theta_n \to \delta V_n$ and $x_\ell \to r_\ell$.

2.3. Optimisation with linear power flow equations

PyPSA is a partial equilibrium model that can optimise both short-term operation and long-term investment in the energy system as a linear problem using the linear power flow equations.

PyPSA minimises total system costs, which include the variable and fixed costs of generation, storage and transmission, given technical and physical constraints. The objective function is given by

$$\min_{\substack{F_\ell, G_{n,r}, H_{n,s}, E_{n,s} \\ f_{\ell,t}, g_{n,r,t}, h_{n,s,t}, suc_{n,r,t}, sdc_{n,r,t}}} \Bigg[\sum_\ell c_\ell \cdot F_\ell + \sum_{n,r} c_{n,r} \cdot G_{n,r} + \sum_{n,r,t} \left(w_t \cdot o_{n,r} \cdot g_{n,r,t} + suc_{n,r,t} + sdc_{n,r,t} \right) + \sum_{n,s} c_{n,s} \cdot H_{n,s} + \sum_{n,s} \hat{c}_{n,s} \cdot E_{n,s} + \sum_{n,r,t} w_t \cdot o_{n,s} \cdot [h_{n,s,t}]^+ \Bigg] \tag{5}$$

It consists of the branch capacities F_ℓ for each branch ℓ and their annuitised fixed costs per capacity c_ℓ, the generator capacities $G_{n,r}$ at each bus n for technology r and their annuitised fixed costs per capacity $c_{n,r}$, the dispatch $g_{n,r,t}$ of the unit at time t and the associated variable costs $o_{n,r}$, the start up and shut down costs $suc_{n,r,t}$ and $sdc_{n,r,t}$ when unit commitment is activated, the storage unit power capacities $H_{n,s}$ and store energy capacities $E_{n,s}$ at each bus n for storage technology s and their associated fixed costs $c_{n,s}$ and $\hat{c}_{n,s}$, and finally the positive part of the storage dispatch $[h_{n,s,t}]^+$ and the associated variable costs $o_{n,s}$. The branch flows $f_{\ell,t}$ are optimisation variables but do not appear in the objective function. The optimisation is run over multiple time periods t representing different weather

and demand conditions. Each period can have a weighting w_t; the investment costs must then be annuitised for the total period $\sum_t w_t$ (typically a full year).

The dispatch of generators $g_{n,r,t}$ is constrained by their capacities $G_{n,r}$ and time-dependent availabilities $\tilde{g}_{n,r,t}$ and $\bar{g}_{n,r,t}$, which are given per unit of the capacities $G_{n,r}$:

$$\tilde{g}_{n,r,t} \cdot G_{n,r} \leq g_{n,r,t} \leq \bar{g}_{n,r,t} \cdot G_{n,r} \qquad \forall\, n,r,t \tag{6}$$

For conventional generators the availabilities are usually constant; a fully flexible generator would have $\tilde{g}_{n,r,t} = 0$ and $\bar{g}_{n,r,t} = 1$. For variable renewable generators such as wind and solar, $\bar{g}_{n,r,t}$ represents the weather-dependent power availability, while curtailment may also be limited by introducing a lower bound on the dispatch $\tilde{g}_{n,r,t}$.

The dispatch can also be limited by ramp rate constraints $ru_{n,r}$ and $rd_{n,r}$ per unit of the generator nominal power:

$$-rd_{n,r} \cdot G_{n,r} \leq \left(g_{n,r,t} - g_{n,r,t-1}\right) \leq ru_{n,r} \cdot G_{n,r} \qquad \forall\, n,r,t>0 \tag{7}$$

Unit commitment for conventional generators is described in Section 2.5.

The power capacity $G_{n,r}$ can also be optimised within minimum $\tilde{G}_{n,r}$ and maximum $\bar{G}_{n,r}$ installable potentials:

$$\tilde{G}_{n,r} \leq G_{n,r} \leq \bar{G}_{n,r} \qquad \forall\, n,r \tag{8}$$

The dispatch of storage units $h_{n,s,t}$, whose energy carriers are labelled by s, is constrained by a similar equation to that for generators in equation (6):

$$\tilde{h}_{n,s,t} \cdot H_{n,s} \leq h_{n,s,t} \leq \bar{h}_{n,s,t} \cdot H_{n,s} \qquad \forall\, n,s,t \tag{9}$$

except $\tilde{h}_{n,s,t}$ is now negative, since the dispatch of storage units can be both positive when discharging into the grid and negative when absorbing power from the grid. The power capacity $H_{n,s}$ can also be optimised within installable potentials.

The energy levels $e_{n,s,t}$ of all storage units have to be consistent between all hours and are limited by the storage energy capacity $E_{n,s}$

$$\begin{aligned} e_{n,s,t} &= \eta_{n,s,0}^{w_t} e_{n,s,t-1} \\ &\quad + \eta_{n,s,+} \cdot w_t \left[h_{n,s,t}\right]^+ - \eta_{n,s,-}^{-1} \cdot w_t \left[h_{n,s,t}\right]^- \\ &\quad + w_t \cdot h_{n,s,t,\text{inflow}} - w_t \cdot h_{n,s,t,\text{spillage}} \\ \tilde{e}_{n,s,t} \cdot E_{n,s} &\leq e_{n,s,t} \leq \bar{e}_{n,s,t} \cdot E_{n,s} \qquad \forall\, n,s,t \end{aligned} \tag{10}$$

Positive and negative parts of a value are denoted as $[\cdot]^+ = \max(\cdot, 0)$, $[\cdot]^- = -\min(\cdot, 0)$. The storage units can have a standing loss (self-discharging leakage rate) $\eta_{n,s,0}$, a charging efficiency $\eta_{n,s,+}$, a discharging efficiency $\eta_{n,s,-}$, inflow (e.g. river inflow in a reservoir) and spillage. The initial energy level can be set by the user, or it is assumed to be cyclic, i.e. $e_{n,s,t=0} = e_{n,s,t=T}$.

The store component is a more basic version of the storage unit: its charging and discharging power cannot be limited and there are no charging and discharging efficiencies $\eta_{n,s,+}$, $\eta_{n,s,-}$. The energy levels of the store can also be restricted by time series $\tilde{e}_{n,s,t}$, $\bar{e}_{n,s,t}$ given per unit of the energy capacity $E_{n,s}$; this allows the demand-side management model of [42] to be implemented in PyPSA. The energy capacity $E_{n,s}$ can also be optimised within installable potentials.

Global constraints related to primary energy consumption, such as emission limits, can also be implemented. For example, CO_2 emissions can be limited by a cap CAP_{CO_2}, implemented using the specific emissions e_r in CO_2-tonne-per-MWh_{th} of the fuel r and the efficiency $\eta_{n,r}$ of the generator:

$$\sum_{n,r,t} \frac{1}{\eta_{n,r}} w_t \cdot g_{n,r,t} \cdot e_r \leq \text{CAP}_{CO_2} \qquad \leftrightarrow \qquad \mu_{CO_2} \tag{11}$$

μ_{CO_2} is the shadow price of this constraint.

The (inelastic) electricity demand $d_{n,t}$ at each bus n must be met at each time t by either local generators and storage or by the flows $f_{\ell,t}$ from the branches ℓ

$$\begin{aligned} \sum_r g_{n,r,t} + \sum_s h_{n,s,t} + \sum_\ell \alpha_{\ell,n,t} \cdot f_{\ell,t} &= d_{n,t} \\ &\leftrightarrow w_t \cdot \lambda_{n,t} \quad \forall\, n,t \end{aligned} \tag{12}$$

where $\alpha_{\ell,n,t} = -1$ if ℓ starts at n, $\alpha_{\ell,n,t} = 1$ if ℓ is a line or transformer and ends at n, and $\alpha_{\ell,n,t} = \eta_{\ell,t}$ if ℓ is a link and ends at n (note that for lines and transformers, $\alpha_{\ell,n,t}$ is the incidence matrix of the network, $\alpha_{\ell,n,t} = K_{n\ell}$). $\eta_{\ell,t}$ represents an efficiency loss for a link (it can be time-dependent for efficiency that, for example, depends on the outside temperature, like for a heat pump). $\lambda_{n,t}$ is the marginal price at the bus. This equation implements Kirchhoff's Current Law (KCL), which guarantees energy conservation at each node.

The flows in all passive branches are constrained by their capacities F_ℓ

$$\left|f_{\ell,t}\right| \leq F_\ell \qquad \forall\, \ell,t \tag{13}$$

For links, the flows can be more finely controlled with time-dependent per unit availabilities $\tilde{f}_{\ell,t}, \bar{f}_{\ell,t}$

$$\tilde{f}_{\ell,t} \cdot F_\ell \leq f_{\ell,t} \leq \bar{f}_{\ell,t} \cdot F_\ell \qquad \forall\, \ell,t \tag{14}$$

which allows, for example, time-dependent demand-side management schemes to be modelled [42]. For both passive branches and links, the upper and lower limits are associated with KKT multipliers $\bar{\mu}_{\ell,t}$ and $\underline{\mu}_{\ell,t}$.

The flows in links are fully controllable.

Power flows in networks of passive branches (lines and transformers) according to the linear power flow equations. It is assumed that the network is lossless, so that $\eta_{\ell,n,t} = 1$ for passive branches. To guarantee the physicality of the network flows, in addition to KCL, Kirchhoff's Voltage Law (KVL) must be enforced in each connected network. KVL states that the voltage differences around any closed cycle in the network must sum to zero. If each independent

cycle c is expressed as a directed combination of passive branches ℓ by a matrix $C_{\ell c}$ then KVL becomes the constraint

$$\sum_{\ell} C_{\ell c} \cdot x_{\ell} \cdot f_{\ell,t} = 0 \qquad \forall c,t \tag{15}$$

where x_{ℓ} is the series inductive reactance of branch ℓ. In a recent paper it is demonstrated that this formulation of the linear load flow using cycles solves up to 20 times faster than standard formulations using the voltage angles [30]; voltage angle and PTDF formulations are also implemented in PyPSA and deliver identical results.

Since branch capacities F_{ℓ} can be continuously expanded to represent the addition of new circuits to an aggregated transmission corridor ℓ, the impedances x_{ℓ} of the branches would also decrease. In principle this would introduce a bilinear coupling in equation (15) between the x_{ℓ} and the $f_{\ell,t}$. To keep the optimisation problem linear and therefore computationally fast, x_{ℓ} can be left fixed in each optimisation problem, updated and then the optimisation problem rerun, in up to 5 iterations to ensure convergence, following the methodology of [43]. Another author has implemented an integer transmission expansion in PyPSA [34] that bypasses the bilinearity with a disjunctive big-*M* relaxation [44]; this will be incorporated into the main code base of PyPSA soon.

2.4. Coupling to other energy sectors

PyPSA can also optimise operation and investment in other energy sectors, such as natural gas, heating and transport. These sectors can be modelled using a network of links with efficiencies for energy conversion losses; an example from a recent paper [29] is shown in **Figure 2**. For example, links from electricity to heat buses can represent resistive heaters and/or heat pumps (the latter can also be modelled with a time-dependent coefficient of performance, given the importance of capturing the dependence of heat pump performance on outside temperature [45]). Combined Heat and Power plants (CHPs) can also be modelled by adding additional constraints for the back pressure and fuel consumption (see the PyPSA examples [26]). Depletable resources such as natural gas are modelled with stores.

2.5. Unit Commitment

Unit commitment can be turned on for any generator. This introduces a times series of new binary status variables $u_{n,r,t} \in \{0, 1\}$, which indicates whether the generator is running (1) or not (0) in period t. The restrictions on generator output now become:

$$u_{n,r,t} \cdot \tilde{g}_{n,r,t} \cdot G_{n,r} \le g_{n,r,t} \le u_{n,r,t} \cdot \bar{g}_{n,r,t} \cdot G_{n,r} \qquad \forall n,r,t \tag{16}$$

so that if $u_{n,r,t} = 0$ then also $g_{n,r,t} = 0$.

If $T_{n,r}^{\min\_up}$ is the minimum up time then we have

$$\sum_{t'=t}^{t+T_{n,r}^{\min\_up}} u_{n,r,t'} \ge T_{n,r}^{\min\_up} \left(u_{n,r,t} - u_{n,r,t-1} \right) \qquad \forall n,r,t \tag{17}$$

(i.e. if the generator has just started up ($u_{n,r,t} - u_{n,r,t-1} = 1$) then it has to run for at least $T_{n,r}^{\min\_up}$ periods). Similarly for a minimum down time of $T_{n,r}^{\min\_down}$

$$\sum_{t'=t}^{t+T_{n,r}^{\min\_down}} \left(1 - u_{n,r,t'}\right) \ge T_{n,r}^{\min\_down} \left(u_{n,r,t-1} - u_{n,r,t} \right) \qquad \forall n,r,t \tag{18}$$

For non-zero start up costs $suc_{n,r}$ a new variable $suc_{n,r,t} \ge 0$ is introduced for each time period t and added to the objective function. The variable satisfies

$$suc_{n,r,t} \ge suc_{n,r} \left(u_{n,r,t} - u_{n,r,t-1} \right) \qquad \forall n,r,t \tag{19}$$

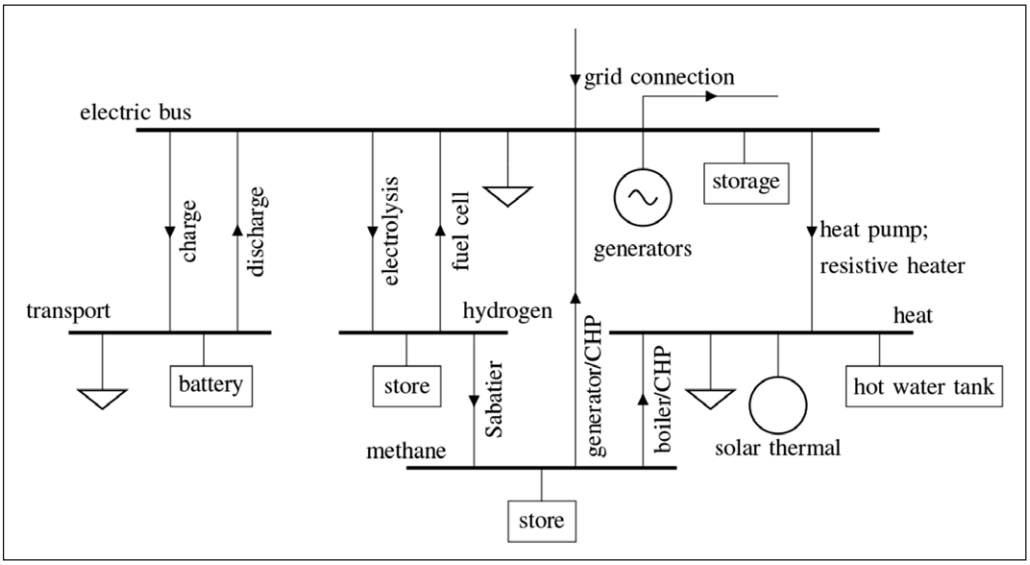

Figure 2: Example of the coupling in PyPSA between electricity (at top) and other energy sectors: transport, hydrogen, natural gas and heating. There is a bus for each energy carrier, to which different loads, energy sources and converters are attached.

so that it is only non-zero if $u_{n,r,t} - u_{n,r,t-1} = 1$, i.e. the generator has just started, in which case the inequality is saturated $suc_{n,r,t} = suc_{n,r}$. Similarly for the shut down costs $sdc_{n,r,t} \geq 0$ we have

$$sdc_{n,r,t} \geq sdc_{n,r}\left(u_{n,r,t-1} - u_{n,r,t}\right) \quad \forall\, n,r,t \tag{20}$$

The ramp-rate limits in equation (7) can also be suplemented by ramping limits at start-up and shut-down.

2.6. Security-Constrained LOPF

PyPSA has functionality to examine the steady state of the power system after outages of passive branches, based on an analysis of the linear power flow.

PyPSA calculates the Branch Outage Distribution Factor (BODF) from the Power Transfer Distribution Factors (PTDF) (see [46]). The BODF gives the change in linearised power flow on passive branch ℓ given the outage of passive branch κ

$$f_\ell^{(\kappa)} = f_\ell + BODF_{\ell\kappa} \cdot f_\kappa \tag{21}$$

Here f_ℓ is the flow before the outage and $f_\ell^{(\kappa)}$ is the flow after the outage of branch κ.

The BODF can then be used in Security-Constrained Linear Optimal Power Flow (SCLOPF). SCLOPF builds on the LOPF by including additional constraints that branches may not become overloaded after the outage of a selection of branches. For each potential outage of a branch κ, a set of constraints for all other branches ℓ is included, guaranteeing that they do not become overloaded beyond their capacity F_ℓ

$$\left|f_{\ell,t}^{(\kappa)}\right| = \left|f_{\ell,t} + BODF_{\ell\kappa} \cdot f_{\kappa,t}\right| \leq \left|F_\ell\right| \quad \forall\, \ell,t \tag{22}$$

2.7. Network clustering

PyPSA also implements a variety of network clustering algorithms to reduce the number of buses in a network while preserving important transmission lines. For example, the κ-means clustering algorithm was recently used in [32] to examine the effect of clustering on investment optimisation results.

2.8. Planned new features

PyPSA is currently in version 0.11.0. PyPSA has been designed to be modular, so that it is possible to develop the code for many other types of calculations. Currently features being considered by the development team include, in rough order of priority:

- Integer transmission expansion, following an existing implementation in PyPSA [34] using the disjunctive big-*M* relaxation [44];
- Multi-horizon dynamic investment optimisation over several years, following for example the implementation in OSeMOSYS [15];
- Transient analysis using the Root-Mean-Square (RMS) values of phasor quantities, following the implementation in PSAT [7];
- An implementation of the non-linear power flow solution using analytic continuation in the complex plane [47], following the implementation in GridCal [48];
- Short-circuit analysis, following the implementation in pandapower [9];
- OPF with the full non-linear network equations, following the implementations in PYPOWER and MATPOWER;
- An interactive web-based GUI for analysing and manipulating the network topology.

3. Implementation and architecture

PyPSA was written in the Python programming language [49] because it is widely used in the modelling community, it is easy to learn and its implementation is also free. It is available for every major operating system, including GNU/Linux, Mac OSX and Windows. PyPSA has been tested with versions 2.7 and 3.5 of Python.

PyPSA stores all data about network components in the DataFrame objects of the Python library pandas [50]. This enables easy and efficient indexing of the data, while mantaining the fast calculation speeds of the underlying array objects of the Python library NumPy [51]. For each of the components listed in **Table 2** (except the overall Network container component) there is a DataFrame listing the static attributes (such as line impedance or capital cost) and a dictionary of DataFrames containing the time-varying attributes (such as wind power availability or consumer demand) that are in addition indexed by the list of snapshots. The specification of some attributes (such as generator maximum output) can be either static or time-varying; if the time series is not specified, then the static value is taken.

All matrix calculations and solutions of linear equation systems are carried out either with NumPy [51] or, in the case of sparse matrices, with SciPy [52]. These Python libraries interface with lower-level programming language libraries to benefit from the speed of strongly-typed languages.

Optimisation problems are formulated using the Python-based optimization modeling language Pyomo [53, 54], which is solver agnostic and intuitive to extend. The use of Pyomo also allows inter-operability with other energy system frameworks that use Pyomo, such as calliope [11], oemof [14] and urbs [20]. In PyPSA lower-level functions in Pyomo have been exploited to improve computational performance.

PyPSA has no graphical user interface, but integrates closely with the IPython [55] interactive notebooks, where networks and their properties can be visualised using the Python library Matplotlib [56] (see **Figures 4** and **5**) or the interactive JavaScript-based library plotly [57].

Internally PyPSA converts all power system quantities (voltage, power, current, impedances) to per unit values. Set points for loads and generation are stored separately from the power values which are actually dispatched.

4. Quality control

PyPSA comes with a large test suite that covers all of its major functionality. Tests are implemented using the Python library pytest [58]. Tests are also included

that compare PyPSA's results with other software such as PYPOWER [8] and pandapower [9]. Users can and do report bugs by raising issues in the GitHub repository [24] or on the forum [27].

5. Performance

In this section some examples of PyPSA's computational performance are given.

In **Figure 3** computation times are given for a full power flow on the MATPOWER [6] test cases (the IEEE standard cases as well as snapshots from the French TSO RTE and European networks [59]) using MATPOWER and PyPSA. In both cases the complete execution of the load flow function ('runpf' for MATPOWER and 'network.pf' for PyPSA) was timed on a computer with Intel Core i5-2520 M processors at 2.50 GHz each with a tolerance of 10^{-8} for the summed error in the apparent power S from equation (2). The timings were averaged over 10 attempts for each network. The computation times are similar, thanks to the fact that both MATPOWER and PyPSA (via the SciPy library [52]) use the same C library umfpack [60] for solving sparse linear equation systems, but PyPSA is in all cases slightly slower due to the overhead of preparing the admittance matrices in pure Python code. If the admittance matrix remains the same for several calculations, PyPSA has the option to avoid recalculating it, which can save some of this time; further acceleration is possible by using the just-in-time (jit) compiler numba [61], as has been done in the pandapower project [9] with success for larger networks.

For the linear optimal power flow (LOPF) the computation performance depends strongly on the choice of linear solver. To give an indication of typical calculation times, if dispatch in the SciGRID model of the German transmission network described in Section 7 (585 buses, 1423 generators including curtailable wind and solar at each node, 38 pump storage units, 852 lines, 96 transformers) is optimised over 4 snapshots, it takes 5 seconds using the COIN-OR Clp free solver on the computer described above. Extensive timings for different formulations of the LOPF problem can be found in [30].

6. Comparison to other power system tools

Given the proliferation of software tools available for modeling power systems, a guide is provided here that briefly compares PyPSA to other power system tools, with a particular focus on free software in the Python programming language. The advantages of Python are discussed above in Section 3.

Selected features for a selection of different software tools are compared in **Table 3**. Many of the tools have specialised features that are not shown in the table, so this table should only be treated as an indicative overview of their features in relation to PyPSA's features.

Many power system tools concentrate on steady-state, dynamic (i.e. short-term transient) and single-period OPF analysis of power networks. They neglect the multi-period unit commitment, investment optimisation and energy system coupling which PyPSA offers. In Python we focus our comparison on two tools: PYPOWER and pandapower.

PYPOWER [8] is a port of an older version of MATPOWER [6] from Matlab to Python. It does not make strong use of Python's object-oriented interface and structures data using NumPy arrays, which makes it difficult to track component attributes. It has no functionality to deal with multi-period OPF, which makes it unsuitable for unit commitment, storage optimisation or investment optimisation. This reflects the functionality of older versions of MATPOWER, but the latest version 6.0 of MATPOWER includes the MATPOWER Optimal Scheduling Tool (MOST) [13], which does multi-period OPF, but no investment optimisation. Unlike PyPSA, PYPOWER has the ability to do full non-linear OPF for single snapshots. Pandapower [9] provides a pandas [50] interface to PYPOWER [8], which makes it easier to use, and adds useful

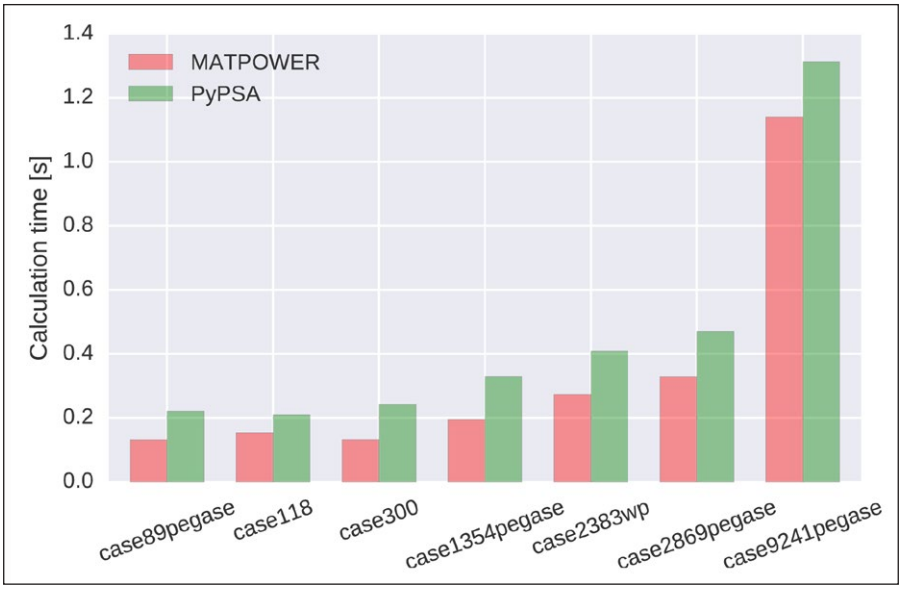

Figure 3: Calculation times for performing a full load flow on the MATPOWER [6] standard cases using MATPOWER versus PyPSA.

Table 3: A comparison of selected features of selected software tools that are similar to PyPSA.

	Software	Version	Citation	Free Software	Grid Analysis			Economic Analysis							
					Power Flow	Continuation Power Flow	Dynamic Analysis	Transport Model	Linear OPF	SCLOPF	Nonlinear OPF	Multi-Period Optimisation	Unit Commitment	Investment Optimisation	Other Energy Sectors
Power system tools	MATPOWER	6.0	[6]	✓	✓	✓		✓	✓		✓				
	NEPLAN	5.5.8	[2]		✓		✓	✓	✓	✓	✓				✓
	pandapower	1.4.0	[9]	✓	✓			✓	✓		✓				
	PowerFactory	2017	[1]		✓		✓		✓	✓	✓				
	PowerWorld	19	[3]		✓		✓	✓	✓	✓	✓				
	PSAT	2.1.10	[7]	✓	✓	✓	✓		✓		✓	✓	✓		
	PSS/E	33.10	[4]		✓		✓		✓	✓	✓				
	PSS/SINCAL	13.5	[5]		✓		✓				✓				✓
	PYPOWER	5.1.2	[8]	✓	✓			✓	✓		✓				
	PyPSA	0.11.0		✓	✓			✓	✓	✓		✓	✓	✓	✓
Energy system tools	calliope	0.5.2	[11]	✓				✓				✓		✓	✓
	minpower	4.3.10	[12]	✓				✓	✓			✓	✓		
	MOST	6.0	[13]	✓	✓	✓		✓	✓	✓	✓	✓	✓		
	oemof	0.1.4	[14]	✓				✓				✓	✓	✓	✓
	OSeMOSYS	2017	[15]	✓				✓				✓		✓	✓
	PLEXOS	7.400	[16]					✓	✓	✓		✓	✓	✓	✓
	PowerGAMA	1.1	[17]	✓				✓	✓			✓			
	PRIMES	2017	[18]					✓	✓			✓	✓	✓	✓
	TIMES	2017	[19]					✓	✓			✓	✓	✓	✓
	urbs	0.7	[20]	✓				✓				✓	✓	✓	✓

functionality such as standard types (on which PyPSA's standard types are based), short circuit calculations, state estimation, and modelling of switches and three-winding transformers. The last four functions are currently missing in PyPSA, along with non-linear OPF, but like PYPOWER, pandapower does not have multi-period OPF functionality. pandapower is under active development and the PyPSA team stays in contact with the pandpower team to exchange tips and features, which is a clear benefit for both developers and users of free software.

PyPSA differs from more general energy system models such as calliope [11], oemof [14], OSeMOSYS [15] and urbs [20] by offering more detailed modelling of power networks, in particular the physics of power flow according to the impedances in the network. PyPSA can model a more general energy network using link components (see Section 2.4), but cannot, for example, yet do the multi-year dynamic investment that OSeMOSYS does. The non-free PLEXOS software [16] comes the closest to matching PyPSA's functionality, but PLEXOS is missing non-linear power flow.

These differences with other software tools are the reason that it was decided to develop a new tool rather than to extend an existing one. Existing tools for power flow such as PYPOWER did not have the internal code and data structures for economic optimisation over multiple time periods with many inter-temporal actors, whereas the energy system tools were missing the tight integration with power flow analysis that we believe is necessary for future research.

7. Demonstration of features on the SciGRID and GridKit datasets

On the PyPSA website [26] a large number of examples of code using PyPSA is linked for reference and to help users just starting out with the software. These range from basic small-scale networks demonstrating the features of PyPSA, to a one-node-per-country model of the European power system with high shares of renewables [62], to full transmission network models available as open data from the SciGRID [63] and GridKit projects [64], [65] which we demonstrate here.

The SciGRID model of Germany provides geo-referenced data for substations and transmission lines (220 kV and above). In one code example, data from openly-available sources on power plant locations and capacities, load distribution and time series are added to the SciGRID data so that load flow calculations can be carried out. The results of one such simulation for Germany with nodal pricing is shown in **Figure 4**. In this snapshot there was a large amount of zero-marginal-cost wind feed-in suppressing the locational marginal prices ($\lambda_{n,t}$ from equation (12)) in the North of Germany. Transmission bottlenecks in the middle of Germany prevent the transportation of this cheap electricity to the South, where more expensive conventional generators set the price. The linearly-optimised dispatch was then fed into a full non-linear power flow calculation where each bus was set to maintain nominal voltage; the resulting reactive power feed-in is also shown in **Figure 4**.

The data quality for the transmission grid in OpenStreetMap outside Germany is not of uniform quality, so for the European grid, an extract of the ENTSO-E interactive map [66] was made [64] using GridKit [65]. The details of how load, conventional power plants and renewable generation time series and expansion potentials were added to the grid data are provided in a forthcoming paper [67]. The result of generation and storage investment optimisation for a clustering of the network from 5000 buses down to 256 buses, allowing no grid expansion and assuming a CO_2 reduction of 95% compared to 1990 levels, is shown in **Figure 5**. The lack of grid expansion forces some balancing of renewable variability locally with storage. Short-term battery storage (grey) combines with solar power (yellow) in Southern Europe, while longer-term hydrogen storage (purple) pairs with wind power (blue) in Northern Europe. This system has an average cost of € 82/MWh. If the grid is optimally expanded, much of the storage can be eliminated and costs are as low as € 65/MWh [32].

8. Conclusions

In this paper a new toolbox has been presented for simulating and optimising power systems. Python for Power System Analysis (PyPSA) provides components to model variable renewable generation, conventional power plants, storage units, coupling to other energy

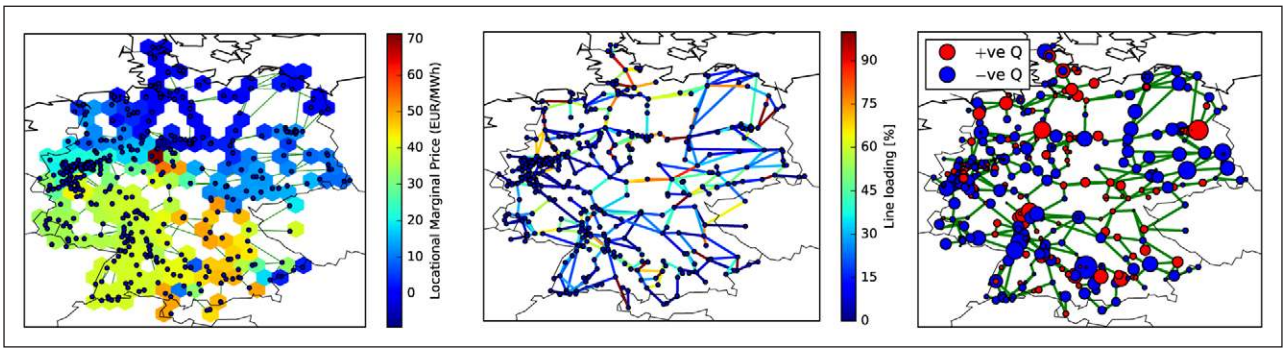

Figure 4: *Left:* Locational marginal prices ($\lambda_{n,t}$ from equation (12)) for Germany in an hour with high wind and low load; *Middle:* Line loading during this hour: highly loaded lines in the middle of Germany prevent the transport of cheap wind energy to consumers in the South; *Right:* Reactive power feed-in (positive in red, negative in blue) necessary to keep all buses at unit nominal voltage.

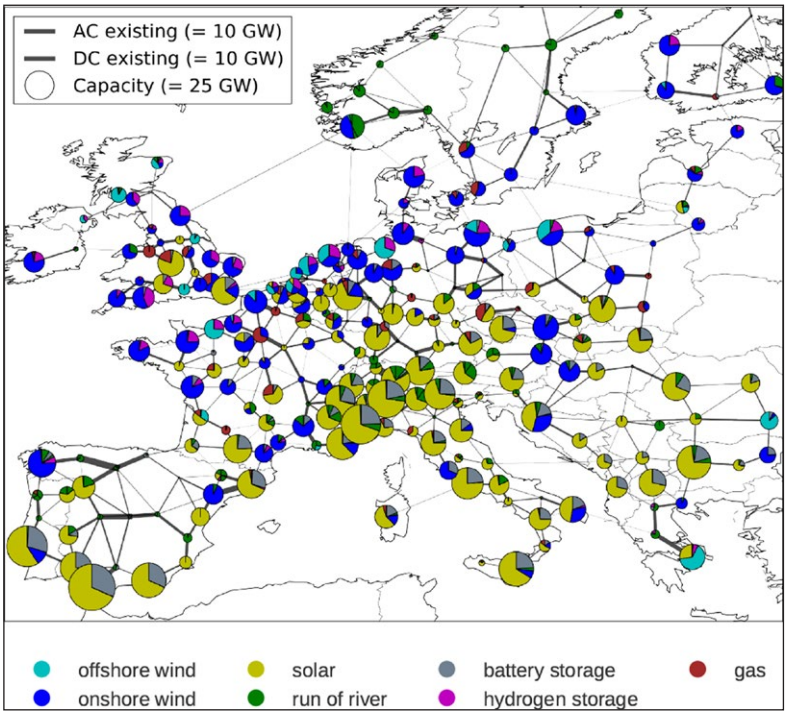

Figure 5: Results of optimisation of generation and storage capacities in Europe to reduce CO_2 emissions in the European electricity sector by 95% compared to 1990 levels [32]. The grid topology is based on the GridKit network for Europe, clustered from 5000 buses to 256 buses.

sectors and multiply-connected AC and DC networks over multiple periods for the optimisation of both operation and investment. Tools are also provided for steady-state analysis with the full load flow equations. PyPSA's performance for large datasets, comparisons with other software packages and several example applications are demonstrated.

As free software, the code of PyPSA can easily be inspected and extended by users, thereby contributing to further research and also transparency in power system modelling. Given that public acceptance of new infrastructure is often low, it is hoped that transparent modelling can contribute to public understanding of the various options we face when designing a sustainable energy system.

(2) Availability

Operating system

GNU/Linux, Mac OSX, Windows and any other operating systems running Python.

Programming language

Python. PyPSA has been tested with versions 2.7 and 3.5 of Python.

Additional system requirements

None.

Dependencies

PyPSA is written in pure Python and is available in the Python Package Index (PyPI). PyPSA depends on the following Python libraries that are not in the Python standard library, but all of which are available in PyPI:

- NumPy [51]
- SciPy [52]
- pandas [50] (version 0.18 or later)
- Pyomo [53, 54]
- networkx (optional for some graph topology algorithms; version 1.10 or later)
- pytest (optional for testing)
- matplotlib (optional for plotting)
- plotly (optional for interactive plotting)

List of contributors

The exact code contributions of each person to version 0.11.0 of PyPSA can be found in the GitHub repository [24].

- Tom Brown, Frankfurt Institute for Advanced Studies
- Jonas Hörsch, Frankfurt Institute for Advanced Studies
- David Schlachtberger, Frankfurt Institute for Advanced Studies
- João Gorenstein Dedecca, Delft University of Technology
- Nis Martensen, Energynautics GmbH
- Konstantinos Syranidis, Forschungszentrum Jülich

Software location

Archive

Name: Zenodo

Persistent identifier: https://doi.org/10.5281/zenodo.1034551

Licence: GPLv3 [21]

Publisher: Zenodo

Version published: 0.11.0

Date published: 21/10/17

Code repository
Name: GitHub
Persistent identifier: https://github.com/PyPSA/PyPSA
Licence: GPLv3 [21]
Date published: 21/10/17

Language
English

(3) Reuse potential

Modelling of the electrical power system is becoming increasingly important thanks to the liberalisation of the power system, the rise of variable renewable energy to combat global warming, and the electrification of transport and heating. PyPSA provides a modular, object-oriented framework for simulating power systems that can be used for research and case studies, and also easily extended beyond its existing functionality. To maximise its reuse potential, PyPSA is written as abstractly as possible, making no assumptions about network topology, infrastructure parameters or asset technologies. Judging by traffic on the forum [27], the website [26] and private communications, PyPSA is already being used by more than a dozen research institutes. Users have already extended it for integer transmission expansion [28, 34] and in the grid planning tool open_eGo [35].

Support for new users is provided on the PyPSA website [26] in the form of documentation and extensive usage examples, as well as on the PyPSA forum [27].

Users can contribute towards the code by raising issues or making pull requests on the GitHub repository [24], or by interacting with the PyPSA developers on the PyPSA forum [27].

Acknowledgements
We thank Stefan Schramm for supporting the development of PyPSA. We thank the community of PyPSA users for bug reports, improvement suggestions and their general friendly support and encouragement for the further development of PyPSA.

Competing Interests
The authors have no competing interests to declare.

References

1. **DIgSILENT GmbH** 2017 "PowerFactory." http://digsilent.de/.
2. **NEPLAN AG** 2017 "PowerFactory." http://www.neplan.ch/.
3. **PowerWorld Corporation** 2017 "PowerWorld." https://www.powerworld.com/.
4. **Siemens AG** 2017 "PSS/E." http://w3.siemens.com/smartgrid/global/en/products-systems-solutions/software-solutions/planning-data-management-software/planning-simulation/Pages/PSS-E.aspx.
5. **Siemens AG** 2017 "PSS/SINCAL." http://w3.siemens.com/smartgrid/global/en/products-systems-solutions/software-solutions/planning-data-management-software/planning-simulation/pss-sincal/pages/pss-sincal.aspx.
6. **Zimmerman, R D, Murillo-Sanchez, C E** and **Thomas, R J** 2011 "MATPOWER: Steady-state operations, planning and analysis tools for power systems research and education." *IEEE Trans. Power Syst.*, 26: 12. [Online]. DOI: https://doi.org/10.1109/TPWRS.2010.2051168
7. **Milano, F** "An open source power system analysis toolbox." *IEEE Transactions on Power Systems*, 20(3): 1199–1206. Aug 2005. [Online]. DOI: https://doi.org/10.1109/TPWRS.2005.851911
8. **Lincoln, R** 2017 "PYPOWER." https://github.com/rwl/PYPOWER.
9. **Thurner, L, Scheidler, A, Schäfer, F, Menke, J-H, Dollichon, J, Meier, F, Meinecke, S** and **Braun, M** 2017 pandapower – an Open Source Python Tool for Convenient Modeling, Analysis and Optimization of Electric Power Systems. Preprint. [Online]. Available: https://arxiv.org/abs/1709.06743.
10. **Open Electrical Wiki** 2017 "Power Systems Analysis Software." http://www.openelectrical.org/wiki/index.php?title=Power_Systems_Analysis_Software.
11. **Pfenninger, S** 2017 "Dealing with multiple decades of hourly wind and PV time series in energy models: A comparison of methods to reduce time resolution and the planning implications of inter-annual variability." *Applied Energy*, 197: 1–13. [Online]. DOI: https://doi.org/10.1016/j.apenergy.2017.03.051
12. **Greenhall, A** and **Christie, R** "Minpower: A power systems optimization toolkit." In: *2012 IEEE Power and Energy Society General Meeting*, 1–6. July 2012. [Online]. DOI: https://doi.org/10.1109/PESGM.2012.6344667
13. **Murillo-Sanchez, C E, Zimmerman, R D, Anderson, C L** and **Thomas, R J** 2013 "Secure planning and operations of systems with stochastic sources, energy storage and active demand." *IEEE Transactions on Smart Grid*, 4: 2220–2229. [Online]. DOI: https://doi.org/10.1109/TSG.2013.2281001
14. **Hilpert, S, Günther, S, Kaldemeyer, C, Krien, U, Plessmann, G, Wiese, F** and **Wingenbach, C** 2017 "Addressing energy system modeling challenges: The contribution of the open energy modelling framework (oemof)." *Preprints*. [Online]. DOI: https://doi.org/10.20944/preprints201702.0055.v1
15. **Howells, M, Rogner, H, Strachan, N, Heaps, C, Huntington, H, Kypreos, S, Hughes, A, Silveira, S, DeCarolis, J, Bazillian, M** and **Roehrl, A** 2011 "Osemosys: The open source energy modeling system: An introduction to its ethos, structure and development." *Energy Policy*, 39(10): 5850–5870. sustainability of biofuels. [Online]. DOI: https://doi.org/10.1016/j.enpol.2011.06.033
16. **Energy Exemplar** 2017 "PLEXOS." http://energyexemplar.com/.
17. **Svendsen, H G** and **Spro, O C** 2016 "PowerGAMA: A new simplified modelling approach for analyses of large interconnected power systems, applied to a 2030 Western Mediterranean case study." *Journal of Renewable and Sustainable Energy*, 8(5): 055501. [Online]. DOI: https://doi.org/10.1063/1.4962415

18. "The PRIMES Model." 2009 http://www.e3mlab.ntua.gr/, NTUA, Tech. Rep.
19. **Loulou, R, Remne, U, Kanudia, A, Lehtila, A** and **Goldstein, G** 2005 "Documentation for the TIMES Model – PART I 1–78." *ETSAP, Tech. Rep.* [Online]. Available: http://iea-etsap.org/index.php/documentation.
20. **Dorfner, J, Dorfner, M, Candas, S, Müller, S, Özsahin, Y, Zipperle, T** and **Herzog, S** Icedkk, and WYAUDI, "urbs: v0.6." Aug. 2016. [Online]. DOI: https://doi.org/10.5281/zenodo.60484
21. "GNU General Public Licence." Free Software Foundation. [Online]. Available: http://www.gnu.org/licenses/gpl.html.
22. **Pfenninger, S, DeCarolis, J, Hirth, L, Quoilin, S** and **Staffell, I** 2017 "The importance of open data and software: Is energy research lagging behind?" *Energy Policy*, 101: 211–215. [Online]. DOI: https://doi.org/10.1016/j.enpol.2016.11.046
23. **Pfenninger, S** 2017 "Energy scientists must show their workings." *Nature*, 542: 393. [Online]. DOI: https://doi.org/10.1038/542393a
24. "Python for Power System Analysis (PyPSA) GitHub Repository." [Online]. Available: https://github.com/PyPSA/PyPSA.
25. **Brown, T, Hörsch, J** and **Schlachtberger, D** "Python for Power System Analysis (PyPSA) Version 0.11.0." Apr. 2017. [Online]. DOI: https://doi.org/10.5281/zenodo.1034551
26. **Brown, T, Hörsch, J** and **Schlachtberger, D** "Python for Power System Analysis (PyPSA) Website." [Online]. Available: https://pypsa.org/.
27. "Python for Power System Analysis (PyPSA) Forum." [Online]. Available: https://groups.google.com/forum/#!forum/pypsa.
28. **Dedecca, J G, Hakvoort, R A** and **Herder, P M** 2017 "Transmission expansion simulation for the European Northern Seas offshore grid." *Energy*, 125: 805–824. [Online]. DOI: https://doi.org/10.1016/j.energy.2017.02.111
29. **Brown, T, Schlachtberger, D, Kies, A** and **Greiner, M** 2016 "Sector coupling in a simplified model of a highly renewable European energy system." In: *Proceedings of 15th Wind Integration Workshop*.
30. **Hörsch, J, Ronellenfitsch, H, Witthaut, D** and **Brown, T** "Linear Optimal Power Flow Using Cycle Flows." *ArXiv e-prints*. Apr. 2017. [Online]. Available: https://arxiv.org/abs/1704.01881.
31. **Schlachtberger, D, Brown, T, Schramm, S** and **Greiner, M** 2017 "The benefits of cooperation in a highly renewable European electricity network." *Energy*, 134: 469–481. [Online]. DOI: https://doi.org/10.1016/j.energy.2017.06.004
32. **Hörsch, J** and **Brown, T** 2017 "The role of spatial scale in joint optimisations of generation and transmission for European highly renewable scenarios." In: *Proceedings of 14th International Conference on the European Energy Market (EEM 2017)*. [Online]. Available: https://arxiv.org/abs/1705.07617.
33. **Groissböck, M** and **Gusmao, A** "Impact of high renewable penetration scenarios on system reliability: two case studies in the Kingdom of Saudi Arabia." *ArXiv e-prints*. Sep. 2017. [Online]. Available: https://arxiv.org/abs/1709.03761.
34. **Dedecca, J G** 2017 "Mixed integer modification of the PyPSA package." https://github.com/jdedecca/MILP_PyPSA.
35. "open eGo GitHub Repository." 2017. [Online]. Available: https://github.com/openego.
36. **Heuck, K, Dettmann, K-D** and **Schulz, D** 2013 *Elektrische Energieversorgung*, 9th ed. Berlin Heidelberg New York: Springer-Verlag. DOI: https://doi.org/10.1007/978-3-8348-2174-4
37. **Oswald, B** and **Oeding, D** 2004 *Elektrische Kraftwerke und Netze.*, 6th ed. Berlin Heidelberg New York: Springer-Verlag.
38. **Oswald, B** 2005 "Vorlesung Elektrische Energieversorgung I: Skript Transformatoren." [Online]. Available: http://antriebstechnik.fh-stralsund.de/1024x768/Dokumentenframe/Versuchsanleitungen/EMA/Trafo.pdf.
39. **Grainger, J J** and **Stevenson, W D, Jr.** 1994 *Power System Analysis*. New York: McGraw-Hill.
40. **Purchala, K, Meeus, L, Dommelen, D V** and **Belmans, R** "Usefulness of DC power flow for active power flow analysis." In: *IEEE Power Engineering Society General Meeting*, 1: 454–459. June 2005. [Online]. DOI: https://doi.org/10.1109/PES.2005.1489581
41. **Stott, B, Jardim, J** and **Alsac, O** 2009 "Dc power flow revisited." *IEEE Trans. Power Syst.*, 24(3): 1290. [Online]. DOI: https://doi.org/10.1109/TPWRS.2009.2021235
42. **Kleinhans, D** "Towards a systematic characterization of the potential of demand side management." *ArXiv e-prints*. Jan. 2014. [Online]. Available: https://arxiv.org/abs/1401.4121.
43. **Hagspiel, S, Jägemann, C, Lindenburger, D, Brown, T, Cherevatskiy, S** and **Tröster, E** 2014 "Cost-optimal power system extension under flow-based market coupling." *Energy*, 66: 654–666. [Online]. DOI: https://doi.org/10.1016/j.energy.2014.01.025
44. **Bahiense, L, Oliveira, G C, Pereira, M** and **Granville, S** "A mixed integer disjunctive model for transmission network expansion." *IEEE Transactions on Power Systems*, 16(3): 560–565. Aug 2001. [Online]. DOI: https://doi.org/10.1109/59.932295
45. **Petrović, S N** and **Karlsson, K B** 2016 "Residential heat pumps in the future Danish energy system." *Energy*, 114: 787–797. [Online]. DOI: https://doi.org/10.1016/j.energy.2016.08.007
46. **Wollenberg, B** and **Wood, A** 1996 *Power Generation, Operation, and Control*. John Wiley & Sons.
47. **Trias, A** "The holomorphic embedding load flow method." In: *Power and Energy Society General Meeting, 2012 IEEE*, 1–8. July 2012. DOI: https://doi.org/10.1109/PESGM.2012.6344759
48. **Vera, S P** 2017 "Gridcal." https://github.com/SanPen/GridCal.

49. "Python programming language, version 3.5." 2017 https://www.python.org/.
50. **McKinney, W** 2010 "Data structures for statistical computing in Python." In: *Proceedings of the 9th Python in Science Conference*, 51–56. [Online]. Available: http://conference.scipy.org/proceedings/scipy2010/mckinney.html.
51. **van der Walt, S, Colbert, S C** and **Varoquaux, G** 2011 "The NumPy array: A structure for efficient numerical computation." *Computing in Science & Engineering*, 13: 22–30. [Online]. DOI: https://doi.org/10.1109/MCSE.2011.37
52. **Jones, E, Oliphant, T, Peterson, P,** et al. 2001 "SciPy: Open source scientific tools for Python." [Online]. Available: http://www.scipy.org/.
53. **Hart, W E, Watson, J-P** and **Woodruff, D L** 2011 "Pyomo: modeling and solving mathematical programs in Python." *Mathematical Programming Computation*, 3(3): 219–260. [Online]. DOI: https://doi.org/10.1007/s12532-011-0026-8
54. **Hart, W E, Laird, C, Watson, J-P** and **Woodruff, D L** 2012 Pyomo–optimization modeling in python. *Springer Science & Business Media*, 67. [Online]. DOI: https://doi.org/10.1007/978-1-4614-3226-5
55. **Pérez, F** and **Granger, B E** 2007 "IPython: A System for Interactive Scientific Computing." *Computing in Science & Engineering*, 9: 21–29. [Online]. DOI: https://doi.org/10.1109/MCSE.2007.53
56. **Hunter, J D** 2007 "Matplotlib: A 2d graphics environment." *Computing in Science & Engineering*, 9: 90–95. [Online]. DOI: https://doi.org/10.1109/MCSE.2007.55
57. **P. T. Inc.** 2015 Collaborative data science. Montréal, QC. [Online]. Available: https://plot.ly.
58. **Krekel, H,** et al. 2017 "pytest." [Online]. Available: https://pytest.org/.
59. **Josz, C, Fliscounakis, S, Maeght, J** and **Panciatici, P** "Data in MATPOWER and QCQP Format: iTesla, RTE Snapshots, and PEGASE." *ArXiv e-prints*. Mar. 2016. [Online]. Available: https://arxiv.org/abs/1603.01533.
60. **Davis, T A** "Algorithm 832: Umfpack v4.3—an unsymmetric-pattern multifrontal method." *ACM Trans. Math. Softw.*, 30(2): 196–199. Jun. 2004. [Online]. DOI: https://doi.org/10.1145/992200.992206
61. **Lam, S K, Pitrou, A** and **Seibert, S** 2015 "Numba: A llvm-based python jit compiler." In: *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC*, *ser.*, 7: 1–7: 6 LLVM '15. New York, NY, USA: ACM. [Online]. DOI: https://doi.org/10.1145/2833157.2833162
62. **Schlachtberger, D, Brown, T, Schramm, S** and **Greiner, M** "Supplementary Data: The Benefits of Cooperation in a Highly Renewable European Electricity Network." Jun. 2017. [Online]. DOI: https://doi.org/10.5281/zenodo.804338
63. **Matke, C, Medjroubi, W** and **Kleinhans, D** "SciGRID – An Open Source Reference Model for the European Transmission Network (v0.2)." Jul. 2016. [Online]. Available: http://www.scigrid.de.
64. **Wiegmans, B** "GridKit extract of ENTSO-E interactive map." Jun. 2016. [Online]. DOI: https://doi.org/10.5281/zenodo.55853
65. **Wiegmans, B** 2015 "Improving the topology of an electric network model based on open data." Master's thesis, Energy and Sustainability Research Institute, University of Groningen. [Online]. Available: https://www.scigrid.de/publications/16_1_BWiegmans_Master_Thesis_2015.pdf.
66. "ENTSO-E Interactive Transmission System Map." Jan. 2016. [Online]. Available: https://www.entsoe.eu/map/Pages/default.aspx.
67. **Hörsch, J, Hofmann, F, Schlachtberger, D** and **Brown, T** 2017 *PyPSA-Eur: An open optimization model of the European transmission system*, in preparation.

How to cite this article: Brown, T, Hörsch, J and Schlachtberger, D 2018 PyPSA: Python for Power System Analysis. *Journal of Open Research Software,* 6: 4. DOI: https://doi.org/10.5334/jors.188

Submitted: 02 August 2017 **Accepted:** 12 November 2017 **Published:** 16 January 2018

]u[*Journal of Open Research Software* is a peer-reviewed open access journal published by Ubiquity Press

OPEN ACCESS