DC power flow – internal documentation

December 6, 2012

Rolando A Rodriguez and Sarah Becker with help from Gorm Andresen and Uffe Poulsen

The following document attempts to explain the problem of the constrained DC power flow, and the solutions implemented in the group's code, as well as this code's algorithm.

1 Power flow across a fully renewable Europe

We face the task of describing the flow of power across various nodes in a network. These nodes represent power grids (regions or countries) connected by high voltage transmission lines. For simplicity, we choose to model these flows considering only Kirchhoff's laws. This means that, based on the power generation and consumption (load) in each node, we look for the solution to the problem how power should be transmitted from sources to sinks which minimizes the necessary transmission capacities.

This approach neglects market influences, i.e. the fact that power exchanges from country to country are directly controlled using power electronic devices, such as FACTS or HVDC lines. We expect that such economic agreements between countries lead to shortest-path flows, which is not the optimal solution if one is interested in minimizing the transmission capacities needed.

In particular, we are interested in a fully renewable scenario, where the overall load is (on average) completely covered by generation of wind and solar energy. For a realistic model, we use actual weather data, which is converted into wind and solar energy generation and then normalized to match the average load. For details about the load and generation time series, see Dominik Heide's PhD thesis. The important parameters in this context are α, which is the percentage of wind energy in the mix, and γ, which is a global factor. Together with the load time series, we get from that a time series of power mismatches for each country, denoted by

$$\begin{aligned} \Delta &= \text{renewable generation} - \text{load} \\ &= \gamma\left(\alpha \cdot \text{wind power} + (1-\alpha) \cdot \text{solar power}\right) - \text{load}. \end{aligned} \tag{1.1}$$

From the power mismatches, we want to determine the power flows across Europe, subject to various outer conditions like transmission line capacities or different α and γ values.

2 The DC power flow as a simplification of the AC power flow

Power transmission in large networks can be precisely described by AC power flow, which accounts for voltage drops, phase differences, and active and reactive power. Each of the N buses (nodes) of the system is described by four variables: real power mismatch P, reactive power mismatch Q, the voltage V and the voltage phase angle δ. Depending on whether a bus is a generator bus, a load bus, or a reference (slack) bus, different variables are used in engineering. As capacitive lines require energizing before being able to transmit

power, and inductive loads consume reactive power, the values of P, Q, V, and δ will vary throughout the network. Solving the power flow and finding the power levels of buses and lines requires solving two sets of equations, one for the active and one for the reactive power.

$$P_i = \sum_{k=1}^{N} |V_i||V_k|(G_{ik}\cos\delta_{ik} + B_{ik}\sin\delta_{ik}) \tag{2.1}$$

$$Q_i = \sum_{k=1}^{N} |V_i||V_k|(G_{ik}\sin\delta_{ik} - B_{ik}\cos\delta_{ik}) \tag{2.2}$$

where $\delta_{ik} = \delta_i - \delta_k$ is the difference of the phase angles at nodes i and k, and G_{ik} and B_{ik} are the real and imaginary parts of the admittance of the links connecting nodes i and k, respectively. The consideration of reactive power and voltage is integral for transmission system operators (TSOs) in order to detect instability in transmission and distribution systems, but complicates the solution significantly. These equations are usually solved using the Newton-Raphson method.

For our purposes, we assume a stable system with stable voltage levels and no active or reactive power losses. This means that the power mismatch in each node is equal to its real part, i.e.

$$\Delta_i = P_i, \quad Q_i = 0$$

In this case, we can simplify the AC power flow to a DC power flow: Essentially, our assumptions imply that the voltage levels throughout the grid are constant, so that $V_i = V_k \equiv 1$, that there are no real components in the impedance of the lines (that is, they have resistances $R_{ik} = 0$ such that $G_{ik} = 0$), and the difference between the phase angles is sufficiently small so that $\sin(\delta_{ik}) \approx \delta_{ik}$. This means that we can forget about (**??**), and (**??**) is simplified to

$$\Delta_i = \sum_{k\neq i}^{N} B_{ik}(\delta_i - \delta_k) = \sum_{k\neq i}^{N} B_{ik}\delta_i - \sum_{k\neq i}^{N} B_{ik}\delta_k \equiv \sum L_{ik}\delta_k, \tag{2.3}$$

if we define

$$L_{ik} = \begin{cases} -B_{ik} & \text{if } i \neq k \\ \sum_{k\neq i} B_{ik} & \text{if } i = k \end{cases}$$

In our case where $B_{ik} = 1$, the L matrix is equal to the Laplace matrix

$$L = KK^T$$

where K is the incidence matrix of the network with the elements

$$K_{nl} = \begin{cases} 1 & \text{if link } l \text{ starts at node } n \\ -1 & \text{if link } l \text{ ends at node } n \\ 0 & \text{otherwise} \end{cases}$$

The values of δ_n are now obtained from (**??**). They define the flows between two nodes, as the power flow F_l along link l that connects nodes i and k is given by

$$F_l = B_{ik}(\delta_i - \delta_k),$$

which, in the case where all the lines have $B_{ik} = 1$, is reduced to

$$F = K^T \delta \tag{2.4}$$

To sum up, the problem is now to solve for δ in (**??**) in order to find the flows using (**??**) for given Δ_i and network topology. As B does has one eigenvalue zero, this is only possible for Δs that obey the constraint $\sum_i \Delta_i = 0$. Additionally, the solution is only unique up to a global phase, i.e. one can fix e.g. $\delta_0 = 0$. Alternatively, the Moore-Penrose pseudo inverse can be obtained for L, to solve directly, the latter option saving calculation time in very large systems. Confer Dominik Heide's dissertation for more details.

3 The minimum dissipation principle

We now want to include constraints on the flows that model the finite transmission capacity of a power line, i.e. inequalities of the form

$$F = K^T \delta \leq h \tag{3.1}$$

Additionally, we want to treat cases in which

$$\sum_i \Delta_i \neq 0. \tag{3.2}$$

This is obviously not possible in our setting so far, since there are no solutions at all if $\sum_i \Delta_i \neq 0$ (no surprise here – that would violate the conservation of energy), and if there are solutions, they are unique up to a global phase and do not allow for further constraints on the flows. What we need here is a reformulation of the problem that allows for the inclusion of (**??**) and (**??**). In this section, we will restate the problem, and in the next section, we will include (**??**) and (**??**).

While we have toyed with the idea of changing the reactances of individual lines to regulate the transmission limits, we found that the problem is easier to solve if we state the power in the nodes directly as a function of the flows, and forget about δ. Eq. (**??**) then simply becomes

$$KF = \Delta \tag{3.3}$$

We have now passed from a system in N (number of nodes) variables to a system of L (number of links) variables. For a generic power network, L will larger than N, and therefore, the solution of this problem is no longer unique. Additional solutions arise because Eq. (**??**) can be satisfied by flows that have, in addition to the "net transporting current" that takes power from node to node, "circular components" that flow in a round. Fortunately, it is possible to make it unique and identical to the DC power flow situation by requiring that the square of the flows be minimal. This eliminates the circular flows. It is called the minimum dissipation principle. We show that the solution to Eq. (**??**)

together with the minimization is indeed the same as the one of Eq. (??) using a vector of N Lagrange multipliers λ. We are looking for the minimal flows under the constraint that the power is transported from the sources to the sinks.

$$\min_F F^T F - \lambda^T (KF - \Delta). \tag{3.4}$$

It's easy to see the minimum by completing the squares in F :

$$\left(F - \frac{1}{2}K^T\lambda\right)^T \left(F - \frac{1}{2}K^T\lambda\right) = F^T F - \lambda^T KF + \frac{1}{4}\lambda^T KK^T\lambda$$

so that Eq. (??) becomes

$$\min_F \left(F - \frac{1}{2}K^T\lambda\right)^T \left(F - \frac{1}{2}K^T\lambda\right) - \frac{1}{4}\lambda^T KK^T\lambda + \lambda^T\Delta$$

whose minimum evidently lies at $F = 1/2K^T\lambda$, i.e. the minimizing flows fulfill an equation of the same form as Eq. (??) (the flow is a so-called potential flow). This means that, as long as we minimize the square of the flows $F^T F$ while ensuring that $KF = \Delta$, we are describing the DC power flow, or, in other words, the two formulations are equivalent.

We now turn to the extension of the problem to cases where $\sum_i \Delta \neq 0$. In order to achieve that, we introduce *balancing*, i.e. extra power generation when there is less renewable generation than load, and *curtailment*, i.e. the shedding of overproduction. At the same time, we include the constraints on the flows. Computationally speaking, this is a quadratic programming optimization problem. Numerical solutions are implemented in a number of packages, developed for MATLAB, C and Python. We describe in the following how it is done with C modules generated by `cvxgen`.

4 Balancing and flow calculation with `cvxgen`

To calculate balancing needs and occurring flows, we employ the physical DC approximation to the full AC power flow. It is valid as long as the network is in a stable state, i.e. no significant voltage or phase shifts between the nodes, and for a network in which line resistances can be neglected. It has the obvious advantage that it minimizes the total dissipation in the network, i.e. the power losses, and the necessary flows.

We first take a look at a situation in which there is no global mismatch between renewable energy generation and load.

$$\sum_{i=1}^{N} \Delta_i = 0 \tag{4.1}$$

For a network with unconstrained line capacities ("copper flow"), the DC power flow is then given as that set of flows F_l that fulfills

$$KF = \Delta$$
$$\text{and minimizes } \sum_{l=1}^{L} F_l^2. \tag{4.2}$$

Due to condition (**??**), the existence of such a solution is guaranteed, and the minimization (**??**) makes it unique.

Now we generalize this to the case where we have a global mismatch, which has to be compensated by balancing. The total balancing can be expressed as[1]

$$B_{\text{tot}} = \sum_{i=1}^{N} (\Delta_i - (K \cdot F)_i)_- = \sum_{i=1}^{N} B_i$$

The balancing need B_i is what is potentially left of a negative mismatch Δ_i after it has been reduced by the net imports $-(K \cdot F)_i$. We use the flows to minimize B_{tot}. In other words, we make the system use as much of the renewable generation as possible. This is our top priority.

As a secondary objective, we still want to minimize dissipation. We do so in a second step, where we minimize the flows keeping the total balancing at its minimal value found in the first step:

$$\min_{\sum_{i=1}^{N} (\Delta_i - (K \cdot F)_i)_- \leq B_{\min}} \sum_{l=1}^{L} F_l^2$$

This ensures that we arrive at the minimal flows that make optimal sharing of renewables possible.

If the line capacities are constrained by (possibly direction dependent) capacities, $h_{-l} \leq F_l \leq h_l$, these constraints are included in the two minimizations, and we have

$$\text{Step 1:} \qquad \min_{h_{-l} \leq F_l \leq h_l} \qquad \sum_{i=1}^{N} (\Delta_i - (K \cdot F)_i)_- = B_{\min}$$

$$\text{Step 2:} \qquad \min_{\substack{h_{-l} \leq F_l \leq h_l \\ \sum_{i=1}^{N} (\Delta_i - (K \cdot F)_i)_- \leq B_{\min}}} \sum_{l=1}^{L} F_l^2$$

5 Balancing, flow and storage calculation with `gurobi`

...comes as soon as I understand it.

6 Other tested implementations

`CPLEX` – commercial, apparently a predecessor of gurobi, slower

`cvxopt` – python-internal module, easy, but very slow. In addition, defining a hierarchy of goal just by giving the higher priority cost function a higher prefactor wasn't such a good idea – in the end, the routine just ignored the secondary targets because their influence on the cost function was too low.

[1] $(x)_- = \max\{-x, 0\}$ denotes the negative part of a quantity x.

7 Code walk-through and instructions

The following is a walk-through of the zdcpf.py script, as far as the version of December 6, 2012 is concerned. The code works with data stored on the group's Pepsi server, which requires a user name and password. Access is currently only for group members. Contact us for more information, and maybe we can come to an agreement. In order to access the database from outside the institute, just open an ssh tunnel to the database by typing

```
ssh -L5432:localhost:5432 USERNAME@pepsi.imf.au.dk
```

into your terminal. Note that due to a brute force attack on the server, it was necessary to switch off the password identification when you login from a computer outside the Aarhus mathematics institute (IMF). You have to generate an ssh key pair, e.g. by running

```
ssh-keygen -t rsa -b 4096
```

and put the public key on pepsi with

```
scp  /.ssh/id_rsa.pub <username>@pepsi.imf.au.dk:.ssh/authorized_keys
```

(obviously, you have to do so from a computer which can still reach pepsi by password). Note that this overwrites any existing authorized keys. You can optionally specify a passphrase for your private key. Your (local) private key `/.ssh/id_rsa` is then required to log in, so copy it to any other computers you need to ssh to pepsi from.

The code is written in Python and requires a number of modules to be installed for proper execution. These are: `numpy`, `sqlalchemy`, `psycopg2`, `cvxopt`, `pylab`, `matplotlib`, and `scipy`. Additionally, the group has been working under `ipython`, which somewhat simplifies the process of running scripts. This makes tests easier, but slows python down a bit and there is a known memory leak when generating a lot of histograms, what we actually do (see `http://old.nabble.com/memory-usage-%28leakage-%29-in-ipython-interactive-mode-td22345056.html`). So don't use it generally.

First, you need to fetch the data you need from the server. You will find the necessary functions for this in the file `Database_v2.py`. There you find for example the function `get_ISET_country_data`, which can be used to load data (wind and solar energy generation, load etc.) of a specific country. It first attempts to access the data locally by looking for a `data` folder, and if the data are not found, it tries to download them from the Pepsi server. These files are in numpy's own binary format, and contain the 70128 data points of load time series, as well as normalized values for offshore and onshore wind and solar generation for each region.

Once you have the data, you will want to determine the flow time series in a specific setting. The bulk of the code for this purpose can be found in `zdcpf.py`. To execute, it needs additional input data in a folder which is by default called `settings`: `admat.txt`, in which the line capacities between the European countries are listed as a matrix (data from ENTSO-E), and `ISET2ISO_country_codes.npy`, which is a binary file that contains the mapping of the ISET and ISO country codes to each other.

Before running the code, we take a look at `zdcpf.py`. In the core of the algorithm stands the node class, defined in the first lines of the file. This is a type of object which we define to contain several vectors, matrices, and methods. When you declare a variable to be of

type node, you must provide it with a path to a file, the file name, and an ID number. For example, defining a node `Norway` using data from N.npy would be done simply by saying

`Norway=node('./data','N.npy',0)`.

The `Norway` variable now holds information on Norway's load, wind and solar production and values for gamma and alpha. Though it is technically possible to modify or extract the information directly from this object (for example, by typing `print Norway.wind` or `Norway.alpha=0.5`), doing so will bring up problems since when some variables like e.g. α are changed, other variables like e.g. the total renewable generation has to be changed as well to keep the data consistent. Instead, it is better to use the functions that we have defined for these purposes, and which can be looked up in the class definition (for example `print Norway.getwind()` and `Norway.setalpha(0.5)`).

For convenience, there is also a class `Nodes`, which holds the set of all nodes in a specific calculation. Its default behavior is to include all European nodes upon construction.

Next in the script, we meet the function `AtoKh`, which generates the incidence matrix K and the flow capacity vector h (as of today) from the data in `admat.txt`. The node setup may be changed by changing the `admat.txt` file. Note, however, that you still must provide a data file for the each node and add it to the list of nodes in the script, or else the code will fail.

```
F = sdcpf(Nodes,admat,path_to_admat,copper,lapse,b,h)
```

executes the actual calculations for each point in the time series. The function receives Nodes, which is an instance of the `Nodes` class, the name of the admat file as well as its location, `copper` which is a switch to turn Europe into a copper plate, i.e. consider the transmission capacities as unlimited, `lapse` which is the number of time steps to consider, `b`, the scaling factor for the line capacities (default is one), and a line capacity constraint vector h (defaults to today's capacities). It modifies the Nodes to contain the balancing and curtailment as calculated and returns the list of flows `F`. With these building blocks, one can simulate a series of scenarios, create loops that cycle through different α or γ values, or constraints, or all.

While this guide is not an user's guide, and does not aim at giving a full explanation of every line, we outline a simple example. One would, after opening an ssh tunnel to Pepsi, open ipython and cd to the directory where all the files are stored, then type

```
%run Database_v2.py
get_ISET_country_data('DK')
```

This will check whether the data file for Denmark is available and, if it is not, download the data files for all European countries.

Then one can load all the functions for the power flow by

```
%run zdcpf.py
```

and then type

```
N=Nodes()
F=sdcpf(N)
```

to start a default run. Alternatively, one can set some of the arguments of zdcpf to other values to modify the run to what one is interested is. For example, to sweep over various γ values, one could type

```
N=Nodes()
gammas=[0.25,0.5,0.75]
for i in range(len(gammas)):
  N.set_gammas(gammas[i])
  F = sdcpf(N)
```

Note that each iteration overwrites the previous 70128 points in the Nodes and F variables, and so they must be saved at every iteration. For doubts, questions, comments, complaints and threats, email us @ `rar@imf.au.dk`, or `becker@fias.uni-frankfurt.de`