DC power flow – internal documentation

January 26, 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 (2.2), and (2.1) 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 (2.3). 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 (2.3) in order to find the flows using (2.4) 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 (3.1) and (3.2). In this section, we will restate the problem, and in the next section, we will include (3.1) and (3.2).

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. (2.3) 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. (3.3) 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. (3.3)

together with the minimization is indeed the same as the one of Eq. (2.3) 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. (3.4) 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. (2.3) (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 the problem must be stated for the `CVXOPT` module for Python. Other implementations are quite similar. Along the way, we will come across the precise definitions of curtailment and balancing.

4 Implementing balancing and curtailment

The quadratic programming solver `CVXOPT` (from ConVeX OPTimisation) requires the information to be stored in a specifically shaped matrices. The form and values of these matrices help define the scenario and problem that one wants to optimize. The standard format of a quadratic programming problem is

$$C = xPx + qx - \text{the cost function to be minimized} \tag{4.1}$$
$$Ax = b - \text{the equality constraint} \tag{4.2}$$
$$Gx \leq h - \text{the inequality constraint} \tag{4.3}$$

where x is the variable that is varied and the other entities are parameters of the specific problem (don't confuse the cost matrix P with the power we talked about earlier!). In our case, x does not only consist of the flows F_l, but we also want to minimize balancing B_n

and curtailment R_n at each node. This means that our x looks like

$$x = \begin{pmatrix} F_1 \\ \vdots \\ F_L \\ B_1 \\ \vdots \\ B_N \\ R_1 \\ \vdots \\ R_N \end{pmatrix}$$

Let's first take a look at the cost function. The quadratic parameter P is a $(L+2N) \times (L+2N)$ matrix. It is simply chosen to be diagonal, since there is no distinguished link or node in the network and we do not want to introduce any "synergy effects" between flows, balancing and curtailment. We thus have:

$$P = \begin{pmatrix} P_F & 0 & 0 \\ 0 & P_B & 0 \\ 0 & 0 & P_R \end{pmatrix}$$

Upon examination, we found that in Dominik Heide's code, curtailment is often shared between nodes. That is, in the event of a global positive mismatch where node i has an overproduction of R and node j is perfectly balanced, Dominik's algorithm will have node i export some of its overproduction to node j, and both will curtail amounts $0.5 \cdot R$. This behavior is of course not desired since it leads to additional flows which are completely superfluous. The reason for these results is that Dominik minimized the square of the balancing and treated curtailment just as negative balancing. If you do this, however, an even distribution of curtailment is favored, leading to transport of power which is eventually shedded. This is why we treat curtailment and balancing separately, and this is also the reason why the quadratic parameter P_R is set to zero.

The cost of curtailment is instead accounted for in the linear term. Again, since no nodes or links are distinguished, it is chosen to look like

$$q = \begin{pmatrix} q_F \\ q_B \\ q_R \end{pmatrix}$$

Next, we build the equality constraint matrix A. It should incorporate the DC power flow equation Eq. (3.3), which must now be modified to include balancing and curtailment. It should now express

$$\text{renewable generation} + \text{balancing} = \text{load} + \text{curtailment} + \text{net outflow}.$$

Since load and generation are, in fact, our input parameters, they form the outer constraint on our system, and we have

$$(\text{renewable generation} - \text{load})_n = \Delta_n = \sum_l F_{ln} - B_n + R_n.$$

This means that A is given by

$$A = \begin{pmatrix} K & -\mathbb{I}_{N\times N} & \mathbb{I}_{N\times N}. \end{pmatrix}$$

and $b = \Delta$ by the nodal mismatch of generation and load.

Finally, the inequality constraints must follow the form $Gx \leq h$. Remembering that flows are directed and that the link capacity can have different values in either direction, the flow component of the constraints must have the form

$$\begin{pmatrix} 1 & 0 & \dots \\ -1 & 0 & \dots \\ 0 & 1 & \dots \\ 0 & -1 & \dots \\ \vdots & \vdots & \ddots \end{pmatrix} \begin{pmatrix} F_1 \\ F_2 \\ \vdots \\ \vdots \end{pmatrix} \leq \begin{pmatrix} h_1 \\ h_{-1} \\ h_2 \\ h_{-2} \\ \vdots \end{pmatrix}, \tag{4.4}$$

where h_l is the capacity flowing in the positive direction along link l and h_{-l} is the capacity flowing in the opposite direction, with $h_l, h_{-l} \geq 0$.

Both the curtailment and the balancing are expressed in positive values, so that both have to limited with the condition that $B_n, R_n \geq 0$. This implies that the constraint G matrix in the balancing and section is defined as

$$-\mathbb{I}B \leq 0 \tag{4.5}$$

For the curtailment, we have the additional constraint that there should be no transported curtailment at a node, i.e. it must be limited to the local excess power. Together with the positivity, this means for the curtailment part of G:

$$\begin{pmatrix} 1 & 0 & \dots \\ -1 & 0 & \dots \\ 0 & 1 & \dots \\ 0 & -1 & \dots \\ \vdots & \vdots & \ddots \end{pmatrix} \begin{pmatrix} R_1 \\ R_2 \\ \vdots \\ \vdots \end{pmatrix} \leq \begin{pmatrix} b_1 \\ 0 \\ b_2 \\ 0 \\ \vdots \end{pmatrix} \tag{4.6}$$

5 Different balancing strategies

Given that our solution vector x contains information about not only the flows, but also about the balancing and curtailment activities of each node, the optimization is not only minimizing the square of the flows, but also the mismatch correction at each node. The standard QP cost function

$$C = xPx + qx$$

reads in detail

$$C = \sum_l \left[P_F F_l^2 + q_F F_l\right] + \sum_n \left[P_B B_n^2 + q_B B_n + P_R R_n^2 + q_R R_n\right]$$

P_F , P_B and P_R are the flow, balancing and curtailment prizing in the quadratic component of C and q_F , q_B and q_R are linear ones. As we are interested in minimizing the square of the flows and not in minimizing the flow along any one link in particular, the linear cost of F is set to zero ($q_F = 0$). Likewise, since we don't want to minimize the square of the curtailment (as explained above), we set P_R to zero as well. This leaves us with

$$C = \sum_l P_F F_l^2 + \sum_n \left[P_B B_n^2 + q_B B_n + q_R R_n \right]$$

By adjusting the remaining parameters, we can motivate one kind of behavior over another. If, for instance, local balancing is cheaper than flows, countries will prefer to balance their own loads, even if their neighbors have excess power. Here, the parameters take different values, depending on which scenario we wish to model.

5.1 Weakly independent nodes

This models the case where the nodes have to do their own balancing, and receive no help from their neighbors apart from receiving their renewables. The rules to follow are (in descending priority):

1. Sharing of renewable surplus
2. Own balancing
3. Own curtailment

To remove the motivation to share balancing, P_B is set to zero. In order ensure that the renewable surplus is shared, the marginal cost of transmitting one unit of energy between any two nodes i and j has to be lower than the marginal cost of producing it at the destination with balancing plus the cost of curtailing it at the origin. In other words

$$\frac{\partial C}{\partial F_{ij}} < \frac{\partial C}{\partial B_j} + \frac{\partial C}{\partial R_i} \tag{5.1}$$

which leads to

$$2P_F(N-1)\Delta_0 < q_B + q_R,$$

where the term $(N-1)\Delta_0$ represents the largest possible flow in the network, built from the largest amount of mismatch Δ_0 transferred at a given time and the maximum number of links that this flow could travel through. In the quadratic parameters, we usually set $P_F = 1$. Theoretically, it shouldn't matter how q_B and q_R are chosen in order to satisfy the inequality, as any balancing will automatically imply curtailment if transfer could have been an option. To play safe, we set both of them equal to the left side of the inequality.

5.2 Weakly dependent nodes

This models the case where nodes share the balancing duties whenever there is a global negative mismatch, as opposed to leaving the countries with a negative mismatch to balance

it alone. While the total balancing energy used will be the same in both cases, the total balancing power installed in Europe will be lower in this case (though perhaps higher for an individual, stable node than it would be in an uncooperative scenario). The set of rules is then

1. Sharing of renewable surplus
2. Sharing of balancing
3. Own balancing
4. Own curtailment

In this scenario, we wish to reduce the square of the balancing required in total, so that two countries balancing a mismatch $M/2$ is preferable to a single one balancing M. This implies bringing back the quadratic term P_B. The condition that flow must be cheaper than balancing still holds, so the inequality in (5.1) still applies to this case. In addition, we want to arrange it so that marginal cost of balancing at a node i which is already doing a large amount of balancing is higher than the marginal cost of producing and bringing balancing from a less used node j. That is

$$\begin{aligned} \frac{\partial C}{\partial B_i} &> \frac{\partial C}{\partial B_j} + \frac{\partial C}{\partial F_{ij}} \\ \text{where } \frac{\partial C}{\partial B_i} &= 2P_B B_i + q_B \\ \text{and } \frac{\partial C}{\partial B_j} + \frac{\partial C}{\partial F_{ij}} &= 2P_B B_j + q_B + \sum_l 2P_F \Delta_0 \leq 2P_B B_j + q_B + 2P_F(N-1)\Delta_0 \end{aligned}$$

where, again, $(N-1)\Delta_0$ represents the maximal transferable mismatch along the longest possible route. The inequality in (4.6) can be simplified to

$$\begin{aligned} 2P_B B_i + q_B &> 2P_B B_j + q_B + 2P_F(N-1)\Delta_0 \\ \Leftrightarrow P_B &> \frac{(N-1)P_F\Delta_0}{B_i - B_j} \end{aligned} \tag{5.2}$$

This means that, the more uneven the current balancing productions between nodes i and j are, the (relatively) cheaper it will be to transport power from j to i in comparison to balancing at i. This term explodes as the balancing production in the two nodes equals out, and $B_i = B_j$, so some tolerance level must be introduced. We set this to $\Delta B = B_i - B_j \approx 1\,\text{MW}$.

We found that ensuring that inequality (5.1) holds makes the system behave as a weakly independent grid, and by further asserting inequality (5.2), we can make the system behave as a weakly dependent grid.

6 Code walk-through and instructions

The following is a walk-through of the zdcpf.py script, as far as the version of January 26, 2012 is concerned. The code works with data stored on the group's Pepsi server, which

requires a user name and password. Anders Søndergaard is responsible for issuing new users, and can be contacted at `andersas@gmail.com`. 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.

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 greatly simplifies the process of running scripts.

First, you need to fetch the data you need from the server. You will find the necessary functions for this in the file `datamgmt.py`, which in turn calls functions from `Database_v1.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 DONG), 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 second most important element, the `generatemat()` function, which generates the matrices needed for the QP solver, `cvxopt`.

First it calls the function `AtoKh`, which generates the incidence matrix K and the flow capacity vector h (as of today) from the data in `admat.txt`. Note that, as cvxopt requires

linear independence in the equality constraint matrix, the first line in K represents a dummy node, connected only to the first real node (Norway, in this case). 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.

The further matrices are generated based on the incidence matrix K and the constraints vector h. The first matrix produced is the diagonal P matrix as described in Eq. (4.1), consisting of a series of elements P_F, P_B, and P_R, corresponding to the flow, balancing and curtailment variables.

As said in the previous section, the default value for P_F is 1, and although P_B may take other values, it is initially set, together with P_R, to zero, or, to be precise, as cvxopt does not allow for linearly dependent matrices, the values are set instead to very small. Next, the q vector for the linear terms, is generated by a number of zeroes equal to the number of variables. The values of q_B and q_R can then be individually modified later on. The equality constraints matrix A is then built as explained in the previous section. To keep the linear independence, the first row is eliminated, corresponding to the dummy node. The right hand side of the equation, the mismatch vector b, will be determined later on depending on the local mismatches. As explained in section 4, the inequality constraint matrix G has a different size, shape and construction, according to the different parts of the solution – namely equations (4.4), (4.5), and (4.6). We assemble G in this way as a block-diagonal matrix by setting in first the constraints on flows, then balancing and finally curtailment. At last, the inequality vector h is built based on the transmission capacities, and also leaving space for the variable constraints required by (4.6). Finally, the function

```
N, F, lF = zdcpf(Nodes,admat,path_to_admat,coop,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, `coop` which is a switch which decides whether balancing is shared (see Sec. 5), `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, an artificial mismatch vector b (defaults to the actual mismatch) and a line capacity constraint vector h (defaults to today's capacities). It returns a set of Nodes `N`, which now contains the balancing and curtailment time series, as well as a list of flows `F` and a list of flow names `lF` to facilitate the mapping from the flow vector to the two countries which are linked by the flow.

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. The bottom half of the script is devoted to plotting functions, in order to better visualize the results.

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 datamgmt.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()
zdcpf(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()
ResNodes = Nodes()
gammas=[0.25,0.5,0.75]
for i in range(len(gammas)):
  N.setgammas(N,gammas[i])
  ResNodes, ResFlows, lF = zdcpf(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`.