APPENDIX A

Code Manual

This appendix provides information on the functionalities of the tool and how to use it. Figure A.1 shows an overview over all files used for the program and its dependencies. The results.py file contains functions to extract results from the OPF solver and sort them according to the conventions. It is called by all scripts within the blue shaded area. In general, the scripts contain functions with sub-routines according to the different algorithms. This is to reduce the number of variables in the workspace and to be able to iterate different parameters (e.g. PV penetration, asymmetric loading). The variables in this appendix will be written as in the code, but they coincide with the variables introduced in the thesis. For a description of each script with information on how to extend or interpret the code, see the sections below.

To run a simulation the all files listed in Appendix B are required. The most important parameters to run a case study are the following:

- N_BUS: number of buses in the network, is used to import the correct case specific .csv-files in the */src/*-folder.
- N_PH: number of phases, must either be 1 or 3 for single- or three-phase representation, respectively.
- TIME_HORZ: planning horizon in hours.
- TIMESTEP: the time resolution in hours. The number of time steps for the study is obtained by division of the time horizon and time step.
- V_FCST: specify the forecast version. This version contains 2 versions (see description of *forecast.py*-file)
- Flags: Here, batteries, flexible loads, power curtailment etc. are enabled/disabled.
- UNBALANCE: degree of unbalanced loading. Three levels are introduced: symmetric, lightly, and heavily. The resulting *LOADSHARE*-parameter defines the fraction of peak load distributed to the respective phase.

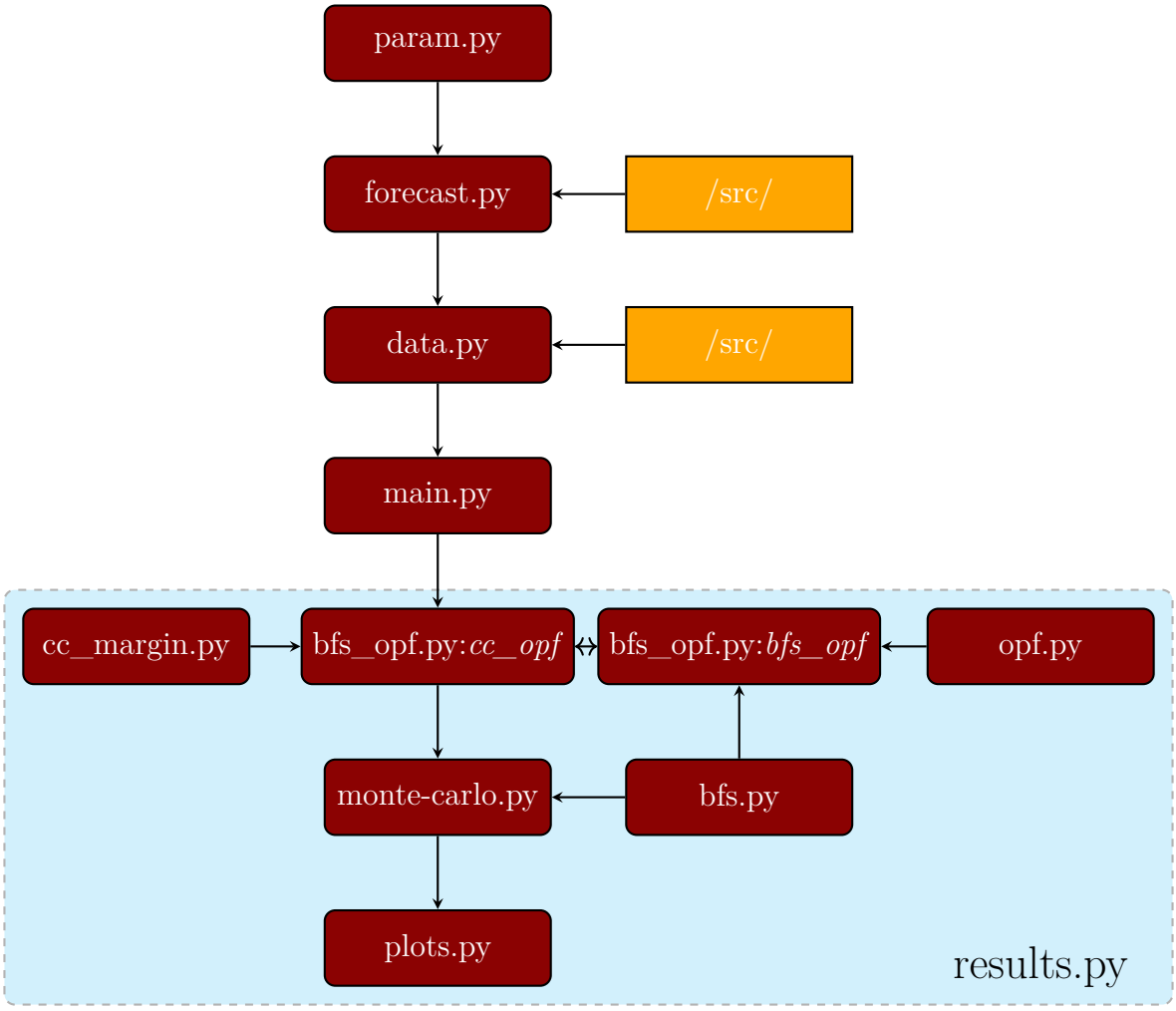

Figure A.1: Overview files with dependencies.

bfs.py

This script contains the BFS sub-routine. It imports the *param.py* and *data.py* script. The *bfs.py* script contains a function with the following input variables:

- sNet: net nodal apparent power from OPF
- time: time horizon. For BFS-OPF scheme: full time horizon of OPF. For Monte-Carlo: time = 1
- tau: OLTC transformer tap position

The script contains a for loop which runs *time* times. For each iteration the solutions are written into the output array. Power losses are calculated for the MATPOWER validation and are not further used in the tool. Finally, following variables are exported:

- bfs_vRe: real part bus voltage

- bfs_vIm: imaginary part bus voltage
- bfs_iRe: real part branch current
- bfs_iIm: imaginary part branch current
- t_BFS: runtime of BFS sweep for all time steps
- bfs_loss: active power losses calculated with result from BFS

bfs_opf.py

This script contains the DRCC-OPF scheme and the BFS-OPF sub-routine. The function for the **DRCC-OPF** scheme takes the following input variables:

- iLoad: iteration counter for load variation
- iPV: iteration counter for PV penetration variation

The function contains a while loop for the outer DRCC-OPF scheme. Inside, the BFS-OPF sub-routine (see below) is called to obtain the updated optimal solutions based on the uncertainty margins of the current iteration. The output of the DRCC-OPF function is a tuple containing the optimal set points from the BFS-OPF sub-routine. Furthermore, the results are exported with .pkl-files to the corresponding folder.

The BFS-OPF sub-routine takes the following inputs:

- sol_vRe: real part bus voltage from previous outer margin loop
- sol_vIm: imaginary part bus voltage from previous outer margin loop
- omI: uncertainty margin for branch current magnitude
- omV: uncertainty margin for bus voltage magnitude
- m: iteration counter outer margin loop
- iLoad,iPV: parameter variations

Note that the first two inputs are according to the initialization (e.g. flat start) for the first outer iteration. For subsequent iterations of the uncertainty loop the voltages are taken from the last BFS-OPF step before the uncertainty margin update. This way, faster convergence is achieved instead of starting with flat start each time the uncertainty margin is updated. The BFS-OPF passes the following variables to the outer DRCC-OPF loop:

- solOPF: optimal set points from OPF
- solBFS: solution from exact BFS run

cc_margin.py

This script calculates the updated uncertainty margins for branch current and bus voltage magnitude. Following variables are imported:

- solBFSOPF: optimal set points and operating point obtained from BFS-OPF scheme of current iteration m
- iLoad,iPV: iteration counter for parameter variation

The uncertainty margins are then updated and the output contains:

- omV_upd: updated uncertainty margin bus voltage
- omI_upd: updated uncertainty margin branch current

data.py

This script imports all external case specific data and generates all arrays used in all scripts. It is split into the following classes to reduce the number of variables in the workspace and for better readability.

- bus: slack bus voltage, maximal voltage unbalance factor, bounds voltage magnitudes
- branch: impedance matrix, bounds branch current magnitudes
- sets: lists with sets of flexible loads, batteries. Used to formulate constraints and other auxiliary arrays
- load: function to generate load profile from input data with specified resolution and time horizon, nodal power demand, limits for load shifting and shedding. Load profiles 1-100 were downloaded from here. The load profile is generated for *TIME_HORZ* hours around midday.
- bess: installed capacity, efficiency, soc-limits, bounds for charging and discharging
- gen: installed capacity of PV and bounds for power ratio
- inverter: function to place inverter to phase specified in input file, calculation of inverter capacity of PV and battery according to asymmetric loading
- cost: arrays with costs for all terms in objective function
- oltc: specifications of OLTC transformer
- pv: import forecast, calculation of covariance matrix, create array with PV forecast per phase

- cc: evaluation of inverse distribution function at confidence level

Also, the static BIBC and BCBV matrices for the BFS sub-routine are generated.

Finally, functions to import and export results are defined here. To change the name convention for results, change the strings in functions *sol_export()* and *sol_import()*.

forecast.py

Different forecast versions are implemented and specified in the parameter file. All forecast versions export the mean, standard deviation, and sample data to a .pkl-file to the */src/fcst/* sub-folder containing version, season and number of time steps in the filename. Out-of-sample and in-sample analyses can be performed according to the forecast.

(1) Gaussian samples. Number of samples, peak PV, standard deviation can be specified. Two sets of samples are generated for out-of-sample analysis

(2) Gaussian samples. One sample set is generated for in-sample analysis.

If real measurement data is available, forecasts can be generated according to historical data. The dimensions for mean and standard deviations must be a 1 dimensional numpy array with the number of time steps. The samples must be sorted in a 2D numpy array with dimensions number of time steps for axis 0, and number of samples for axis 1.

main.py

To run the program, *main.py* has to be executed. It imports the *param.py*, *bfs_opf.py* and *monte_carlo.py* files and iterates through the parameter variations specified in the parameter file. It contains print commands to print the progress and display some key values, such as voltage mismatches or runtimes.

monte_carlo.py

This file runs the Monte-Carlo simulations. Following variables are imported:

- solBFS_OPF: optimal set points from DRCC-OPF.
- iLoad,iPV: iteration counters for parameter variations

Within the function the set points are extracted and the samples imported. A for-loop iterates through all samples to run power flow calculations with ”real” PV infeed and optimal set points. The net nodal apparent power is calculated for each realization and passed to the BFS sub-routine. Violations are summed up and finally exported. The function saves the results in a .pkl-file in the */rlt/* sub-folder, containing following elements:

- mc_vMag: voltage magnitudes for all samples and time steps
- mc_iMag: current magnitudes for all samples and time steps
- n_iUp: number of current violations per phase
- n_vLow: number of lower bound voltage magnitude violations per phase
- n_vUp: number of upper bound voltage magnitude violations per phase

opf.py

The opf.py file contains the OPF problem formulation and builds the model. Here, the gurobipy interface is used. A minimal example is showed as follows:

```
import gurobipy as gp
m = gp.Model('example')
z = m.addVars(dim1,dim2, lb, ub, name)
m.addConstrs((z[x,y] == a[x,y]\
                        for x in range(dim1) for y in range(dim2)),\
                        name='z')
m.setObjective(obj)
m.optimize()
```

Listing A.1: Minimal example gurobipy model.

The opf function takes the following inputs:

- bfs_vRe: real part voltage from previous BFS step
- bfs_vIm: imaginary part voltage from previous BFS step
- omI: uncertainty margin branch current for current iteration m
- omV: uncertainty margin bus voltage for current iteration m
- iLoad,iPV: iteration counters for parameter variation

The input arrays are first pre-processed to the correct dimensions. Then, auxiliary arrays, such as square of voltage magnitude or voltage drop for each line are generated. After pre-processing is done, the actual model building starts. After initialization, all variables are defined. To improve modularity and readability all constraints are formulated in functions which are called all at once. This way, individual models can be expanded. As an example, the model for power losses is shown below:

```
### POWER LOSSES ###
def power_losses():
    # active
    m.addConstrs((pLoss[i,t] ==\
                  iRe[i,t]*bfs_dVRe[i,t] +\
                  iIm[i,t]*bfs_dVIm[i,t]\
```

```
                    for i in range(dimL) for t in range(pm.T)),\
                name='pLoss')

    # reactive
    m.addConstrs((qLoss[i,t] ==\
                iRe[i,t]*bfs_dVIm[i,t] -\
                iIm[i,t]*bfs_dVRe[i,t]\
                    for i in range(dimL) for t in range(pm.T)),\
                name='qLoss')

    # absolute active power losses
    m.addConstrs((pLossAbs[i,t] ==\
                gp.abs_(pLoss[i,t])\
                    for i in range(dimL) for t in range(pm.T)),\
                name='pLossAbs')
```

Listing A.2: Constraints for power losses.

The output of the OPF script is the solver output containing optimal values for decision variables and objective function value. Documentations for the solver interface can be found at the support portal, or the website directly. On the website, there are also some examples and video tutorials

param.py

The parameter file defines all parameters used in the program. All parameters are implemented with capital letters. All lines are commented and the file is split into different sections:

- Gurobi parameters: solver parameters
- General: number of buses to import corresponding source files, number of phases, base power, time horizon and time step, convergence criteria, maximum numbers of iteration counters, forecast version
- Flags: battery,load shedding, load shifting, OLTC, load profile constant or from external file, chance constraints
- Parameter variation: enable/disable parameter variations, season, lists with values for parameter variations (e.g. PV penetration)
- Chance constraints: settings for smooth margin update, predefined power ratio

plots.py

The plots file imports the results from the corresponding results folder and extracts tuples from the .pkl-files. Plots are generated with the matplotlib.pyplot package. The

function to import the .pkl-files is defined in the *data.py* script. An example of how to extract variables from the solver output and plot them is given as follows:

```
import data as dt
import results as

# import solution from OPF
solOPF = (dt.sol_import('opf'))

# active power curtailment per node, phase, time step
opf_aCurt = rlt.read_out_phase('aCurt', dt.n, solOPF)

# plot for all buses
for p in range(pm.N_PH):
    fig,ax = plt.subplots(tight_layout=True)
    for i in range(dt.n):
        ax.plot(opf_aCurt[i,p,:], label='Bus %s'%i)
    ax.set_xlabel('Time Step')
    plt.title('Curtailment Phase %s'%p)
    plt.legend()
    plt.show()
    plt.close()
```

Listing A.3: Example extract and sort solution from solver output.

results.py

This script contains functions to extract results from the solver of BFS scheme and sorts arrays depending on their use. Following functions are used throughout the tool:

read_out(var,dim,sol) Export solver output of variable *var* to dimension *dim*×t, from solution list *sol*

read_out_3ph(var,dim,sol) Export solver output of variable *var* to dimension *dim*×3×t, from solution list *sol*. Results in 3D array

phase_sort(dim,data) Creates a 3D array with *dim*×3×t from *data*

APPENDIX B

Input Files

This appendix lists all external input files required to run case studies with the tool. They are presented in alphabetical order with the file name, usage of the file, and descriptions of all columns. For Table B.6: Diagonal elements contain self-impedances, off-diagonal elements represent mutual impedances.

Table B.1: Input file: battery data.

Column Name	Unit	Description
bus_i	-	bus number
becc_ic	MWh	installed capacity of battery
bess_efficiency	-	battery efficiency
soc_min	-	minimum state of charge
soc_max	-	maximum state of charge
soc_init	-	initial and end state of charge
e2p	-	energy-to-power ratio battery

Table B.2: Input file: branch data.

Column Name	Unit	Description
branch_i	-	branch number
config	-	branch type used for impedance matrix calculation
fbus	-	bus number start point branch
tbus	-	bus number end point branch
ampacity	A	ampacity of branch, thermal limit
length	km	length of branch in km
cc_violations	-	allowable violation probability per branch

Table B.3: Input file: bus data.

Column Name	Unit	Description
bus_i	-	bus number
v_base	kV	base voltage for each bus
v_min	p.u.	minimal bus voltage in per unit system
v_max	p.u.	maximal bus voltage in per unit system
v_slack	p.u.	voltage magnitude slack bus in per unit system
cc_violations	-	allowable violation probability per bus
vuf_max	-	maximum allowable voltage unbalance factor

Table B.4: Input file: cost data.

Column Name	Unit	Description
bus_i	-	bus number
curt	€/MWh	active power curtailment costs
pv	€/MWh	dispatch costs PV
bat	€/MWh	dispatch costs batteries
shed	€/MWh	load shedding costs
shift	€/MWh	load shifting costs
qSupport	€/MVarh	reactive power support inverters costs
loss	€/MWh	active power losses costs
slack_revenue	€/MWh	revenue for exporting power to upper level grid
slack_cost	€/MWh	costs of electricity from upper level grid
slack_q	€/MVarh	reactive power exchange cost with upper level grid

Table B.5: Input file: generator data.

Column Name	Unit	Description
bus_i	-	bus number
icPV	MW	installed capacity of PV
pf_max	-	minimal power factor of inverter

Table B.6: Input file: impedance data.

Column Name	Unit	Description
config	-	branch type specified in branchData
R1	Ω/km	resistance of phase 1
X1	Ω/km	reactance of phase 1
R2	Ω/km	resistance of phase 2
X2	Ω/km	reactance of phase 2
R3	Ω/km	resistance of phase 3
X3	Ω/km	reactance of phase 3

Table B.7: Input file: inverter data.

Column Name	Unit	Description
bus_i	-	bus number
cap_PV	-	multiplication factor of installed capacity of PV to obtain inverter capacity
cap_Bat	-	multiplication factor of installed capacity of battery to obtain inverter capacity
ph_PV	-	used to select phase to which PV inverter is connected: 3 = all phases, 0 = phase a, 1 = phase b, 2 = phase c
ph_Bat	-	used to select phase to which battery inverter is connected: 3 = all phases, 0 = phase a, 1 = phase b, 2 = phase c

Table B.8: Input file: load data.

Column Name	Unit	Description
bus_i	-	bus number
s_peak	kVar	peak apparent load per bus
power_factor	-	power factor for each load
flex_min	-	fraction of peak apparent load that can be reduced
flex_max	-	fraction of peak apparent laod that can be increased
load_profile	-	number of load profile in load profile folder, specified for each bus

Table B.9: Input file: OLTC data.

Column Name	Unit	Description
bus_i	-	bus number
oltc_min	p.u.	reduction of bus voltage in p.u. for minimal tap position
oltc_max	p.u.	increase of bus voltage in p.u. for maximal tap position
oltc_steps	-	number of tap positions
oltc_n_shift	-	allowable number of tap actions in time horizon
symmetric	-	boolean indicating symmetric or asymmetric tap positions for all phases: 1 = symmetric, 0 = asymmetric