Power System Operation and Stability

Power flow programming with Matlab®

In this practical course the contents of power flow programming are deepened and partly extended. Here, the Newton-Raphson method is to be programmed in polar coordinates. In this practical course, the knowledge from the chapter of the fast decoupled power flow is also partly required. The programming is done in Matlab and if possible the advantages of Matlab regarding matrix oriented programming should be used. Only the main part of the calculation is to be implemented. The output as well as the grid data, the structure of the nodal admittance matrix and the function-names are already given. At the end of the programming work, the power flow is tested and extended in detail.

- Programming of the power flow via the operating equipment
- Programming the Newton-Raphson power flow in polar coordinates (without PU nodes).

Table of contents

Preparation

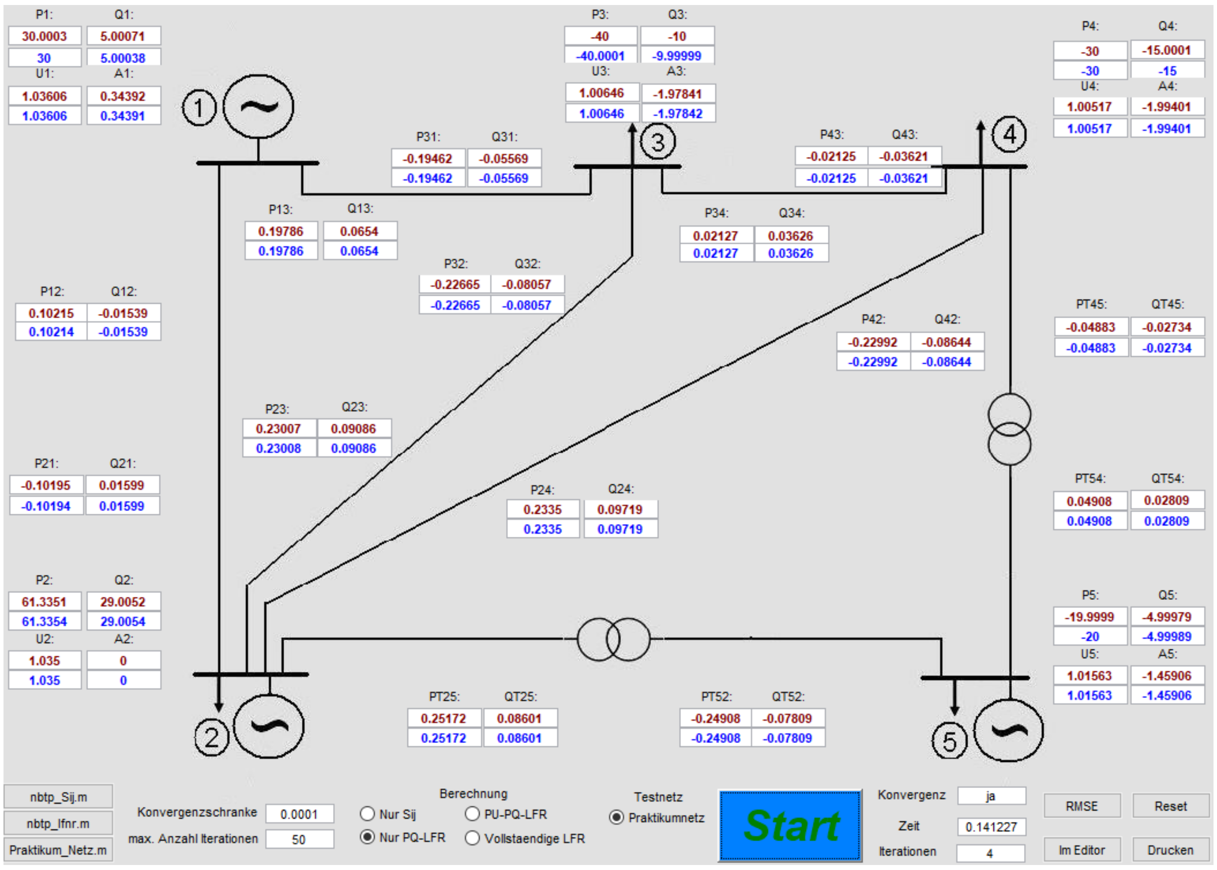
1. You should familiarize yourself with chapter 3 "Power Flow Calculation" of the lecture PSOS part 1.
2. Basic knowledge of the programming language Matlab is assumed. The familiarization with Matlab can be done by using the "Getting Started" book of Matlab or by using the tutorial.

Thoroughly work through the following chapters of the experiment instructions and familiarize yourself with the following:
- Graphical user interface (GUI of the interface) and its buttons (buttons)
- Directory structure of the project
- Structure of the matrices in the file Praktikum_Netz.m

1 Description of the graphical interface and preparation for the tasks

Download the **Matlab files** from moodle and **unzip** the folder. Then open Matlab and **navigate** to the folder where the interface.m file is located. For easier and clearer processing of the experiment, the graphical user interface (GUI) shown in Figure 1.1 was added. To invoke the GUI, execute the **command „interface“** in the Matlab Command Window.

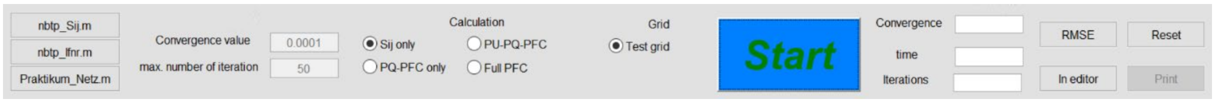
The interface contains the grid image of the test grid. Each element of the test grid is assigned to display elements in which the respective state variables of the grid components, i.e. complex voltages and powers in each node as well as power flows on each transmission element, are presented. Here, in each case, the **reference values are** displayed **in red** (the values calculated by the supervisor version of the power flow) in the **top line of the display field** and the **results of the power flow programmed by you are** displayed **in blue in the line below.**



P1:	30.0003	30
Q1:	5.00071	5.00038
U1:	1.03606	1.03606
A1:	0.34392	0.34391
P3:	-40	-40.0001
Q3:	-10	-9.99999
U3:	1.00646	1.00646
A3:	-1.97841	-1.97842
P4:	-30	-30
Q4:	-15.0001	-15
U4:	1.00517	1.00517
A4:	-1.99401	-1.99401
P31:	-0.19462	-0.19462
Q31:	-0.05569	-0.05569
P43:	-0.02125	-0.02125
Q43:	-0.03621	-0.03621
P13:	0.19786	0.19786
Q13:	0.0654	0.0654
P34:	0.02127	0.02127
Q34:	0.03626	0.03626
P32:	-0.22665	-0.22665
Q32:	-0.08057	-0.08057
P12:	0.10215	0.10214
Q12:	-0.01539	-0.01539
PT45:	-0.04883	-0.04883
QT45:	-0.02734	-0.02734
P42:	-0.22992	-0.22992
Q42:	-0.08644	-0.08644
P23:	0.23007	0.23008
Q23:	0.09086	0.09086
P21:	-0.10195	-0.10194
Q21:	0.01599	0.01599
PT54:	0.04908	0.04908
QT54:	0.02809	0.02809
P24:	0.2335	0.2335
Q24:	0.09719	0.09719
P2:	61.3351	61.3354
Q2:	29.0052	29.0054
U2:	1.035	1.035
A2:	0	0
P5:	-19.9999	-20
Q5:	-4.99979	-4.99989
U5:	1.01563	1.01563
A5:	-1.45906	-1.45906
PT25:	0.25172	0.25172
QT25:	0.08601	0.08601
PT52:	-0.24908	-0.24908
QT52:	-0.07809	-0.07809

Konvergenzschranke 0.0001
max. Anzahl Iterationen 50
Berechnung: Nur Sij [✓ Nur PQ-LFR] PU-PQ-LFR Vollstaendige LFR
Testnetz: [✓ Praktikumnetz]
Konvergenz ja
Zeit 0.141227
Iterationen 4

Figure 1.1: User interface of the power flow test



Convergence value 0.0001
max. number of iteration 50
Calculation: [✓ Sij only] PU-PQ-PFC PQ-PFC only Full PFC
Grid: [✓ Test grid]
Convergence
time
Iterations

The elements for controlling the calculation are explained below. The elements that become particularly important for this practical course are marked in color.

Start button	Start the selected calculation
	Convergence parameters
Convergence	Specification of the convergence value at which the power flow calculation procedure is to be aborted.
Max. Number of iterations	Maximum number of iterations after which the iterative calculation should be terminated in case of non-convergence.
	Option fields "Calculation“
Sij only	Calculation and display of power flows (chapter 2)
PQ-PFC only	Calculation of the power flow (LFR) with the reference node and the PQ nodes (all nodes are understood as PQ nodes).
PQ-PU-PFC	Calculation of the power flow with additional consideration of the PU nodes
Full PFC	Calculation of the power flow with conversion of PU to PQ nodes.
	Test grid" radio buttons
Praktikum_Netz	Calculations are performed for the five-node test grid. The results are displayed in the main window as well as in the Matlab Command Window.
IEEE grid	Calculations are performed for the IEEE RTS three-zone grid. The results are displayed ***only*** in the Matlab Command Window. The display elements of the main window are hidden.
	Advanced display
RMSE	Display of the deviations of the power flow results from the reference values in the Command Window and in a bar graph
In the editor	Display of power flow results as text file in the editor
Reset	Reset the display elements
Print	Output of the current window view (Praktikum_Netz) on a printer
Convergence	Information about the convergence behavior of the implemented Newton-Raphson algorithm
Time	Required time duration for the calculation of the power flow
Iterations	Number of iterations until convergence is reached

After pressing the Start button, the calculations are executed according to the flowchart shown in Figure 1.2. Figure 1.3 shows the directory structure of the folders including the files that have to be processed in this lab.

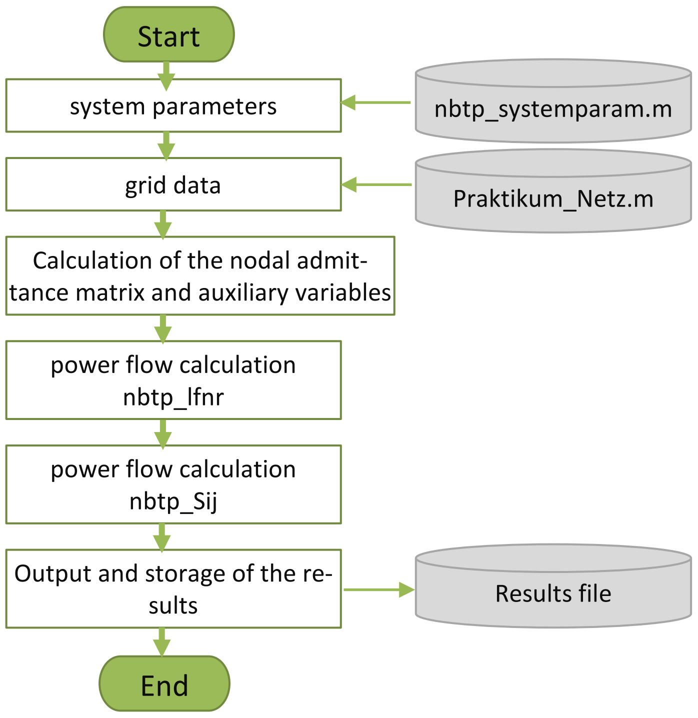
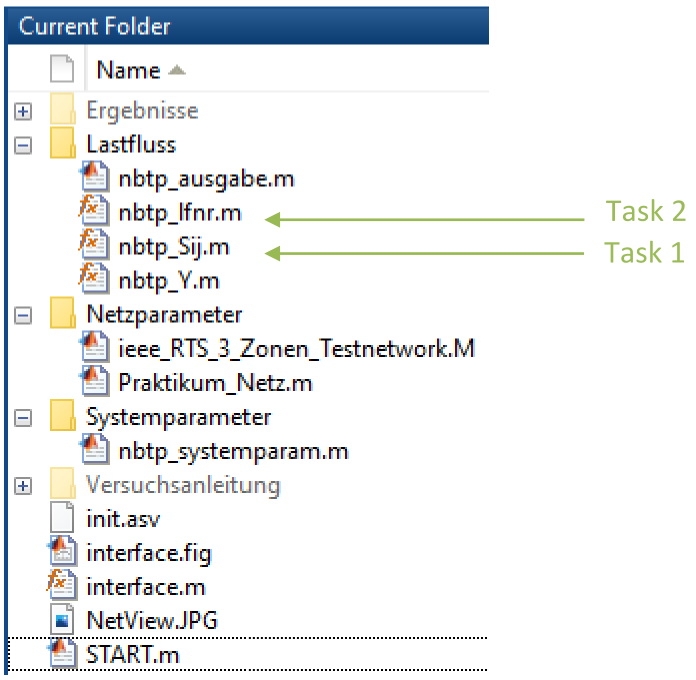

Start
system parameters
nbtp_systemparam.m
grid data
Praktikum_Netz.m
Calculation of the nodal admittance matrix and auxiliary variables
power flow calculation nbtp_lfnr
power flow calculation nbtp_Sij
Output and storage of the results
Results file
End

Figure 1.2: Flow chart of the power flow calculation



Current Folder

Name

- Ergebnisse
- Lastfluss
  - nbtp_ausgabe.m
  - nbtp_lfnr.m ← Task 2
  - nbtp_Sij.m ← Task 1
  - nbtp_Y.m
- Netzparameter
  - ieee_RTS_3_Zonen_Testnetwork.M
  - Praktikum_Netz.m
- Systemparameter
  - nbtp_systemparam.m
- Versuchsanleitung
- init.asv
- interface.fig
- interface.m
- NetView.JPG
- START.m

Figure 1.3: Directory structure of the Matlab project

2 Task 1: Programming the power flow via the operating equipment

As a "warm-up" regarding the programming language Matlab and in preparation for the actual power flow, the power flows over all lines and transformers are to be calculated with an existing result of a power flow. Together with the results of the voltages of all nodes, which are derived from the power flow to be implemented later and are given here, these quantities are important for the evaluation of the overall state of the grid. Figure 2.1 describes the equivalent image of a line.

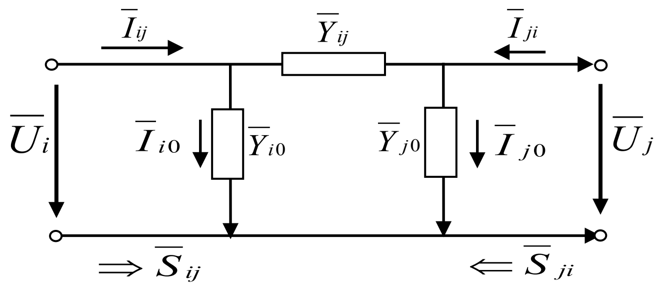


Figure 2.1 Equivalent circuit diagram of a device

The power flows Sij and Sji are calculated with the following formulas. Note that the **complex-conjugate** quantities are given here!

$$\overline{S}_{ij}^{*} = \overline{U}_i^{*}\,\overline{I}_{ij} = P_{ij} - jQ_{ij} = \overline{U}_i^{*}\,(\overline{U}_i - \overline{U}_j)\overline{Y}_{ij} + U_i^2\,\overline{Y}_{i0}$$

$$\overline{S}_{ji}^{*} = \overline{U}_j^{*}\,\overline{I}_{ji} = P_{ji} - jQ_{ji} = \overline{U}_j^{*}\,(\overline{U}_j - \overline{U}_i)\overline{Y}_{ij} + U_j^2\,\overline{Y}_{j0}$$

1. Open the file nbtp_Sij.m and familiarize yourself with the function and the elements. Connection_Data and Tct_Data can be found in the grid file Praktikum_Netz.m, where Connection_Data represents the line connections and Tct_Data the transformer connections. The vector Ua contains the 5 complex node voltages.

2. Program the power flows using the above formulas and the input data for the line connections (Sij, Sji) on the one hand and the transformer connections (Sij_t, Sji_t) on the other hand.

   Tip 1: i and j are selected so that they correspond to the node number from Connection_Data and Tct_Data. Example: Sij = [S12; S13; S23; S24; S34] and Sij_t = [S25; S45].
   Tip 2: First create the vectors Ui, Ui*, Uj, Yij, Yi0 and then use element-wise multiplication. Example: Vector1 .* Vector2
   Tip 3: In case of errors: Use the debugging function of Matlab. Click on the line number to the left of your code or, depending on the Matlab version, on the line next to the numbers. This creates a breakpoint at which the script execution pauses. Now start the calculation with the start button of the GUI and use the "Step" button in the selection bar above your code to execute your script step by step. In the "Workspace" you can display the elements by double clicking on them.

3. Now select "Sij only" in the GUI and press the start button. To check the correctness of the results, you can compare the correct values displayed in red with the blue values calculated by you OR to get a quick overview, you can display the results by calling the RMSE view of the GUI.

3 Task 2: Programming the Newton-Raphson method in polar coordinates

The Newton-Raphson algorithm is to be programmed as indicated in figure 3.1. For this purpose parts of the algorithm are already given in the file nbtp_lfnr.m, which have to be completed by you.

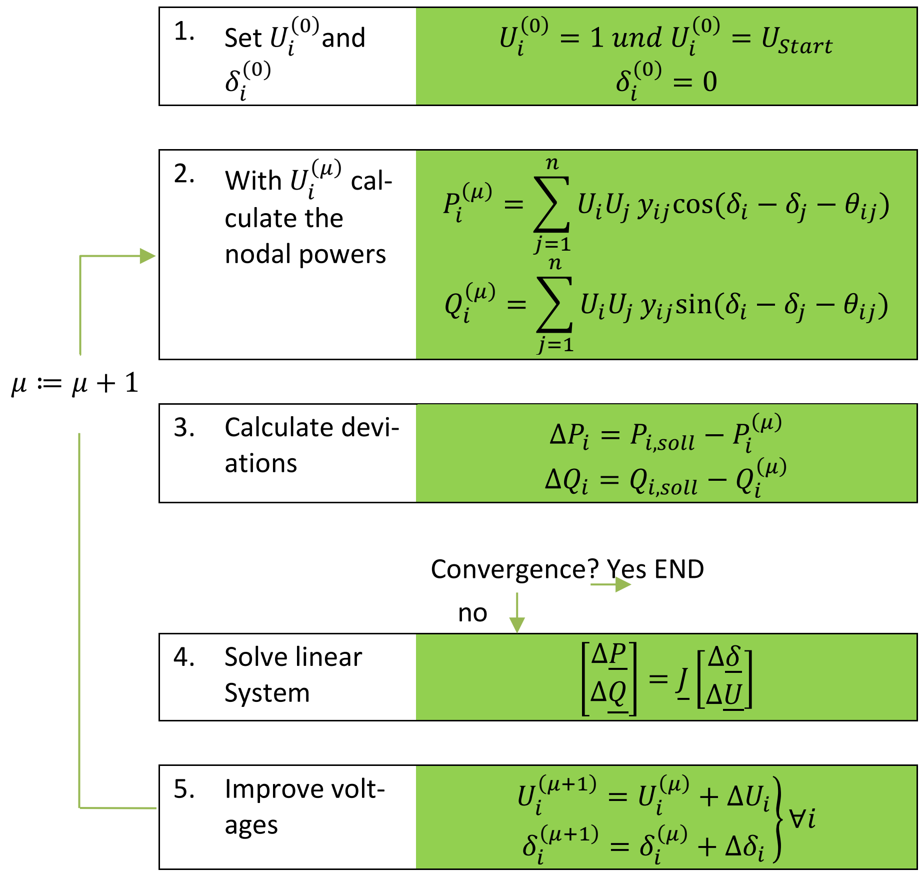

1. Set $U_i^{(0)}$ and $\delta_i^{(0)}$
$U_i^{(0)} = 1 \; und \; U_i^{(0)} = U_{Start}$
$\delta_i^{(0)} = 0$
2. With $U_i^{(\mu)}$ calculate the nodal powers
$P_i^{(\mu)} = \sum_{j=1}^{n} U_i U_j \, y_{ij} \cos(\delta_i - \delta_j - \theta_{ij})$
$Q_i^{(\mu)} = \sum_{j=1}^{n} U_i U_j \, y_{ij} \sin(\delta_i - \delta_j - \theta_{ij})$
$\mu := \mu + 1$
3. Calculate deviations
$\Delta P_i = P_{i,soll} - P_i^{(\mu)}$
$\Delta Q_i = Q_{i,soll} - Q_i^{(\mu)}$
Convergence? Yes END
no
4. Solve linear System
$\begin{bmatrix} \Delta \underline{P} \\ \Delta \underline{Q} \end{bmatrix} = \underline{J} \begin{bmatrix} \Delta \underline{\delta} \\ \Delta \underline{U} \end{bmatrix}$
5. Improve voltages
$\left.\begin{matrix} U_i^{(\mu+1)} = U_i^{(\mu)} + \Delta U_i \\ \delta_i^{(\mu+1)} = \delta_i^{(\mu)} + \Delta \delta_i \end{matrix}\right\} \forall i$

Figure 3.1: Flowchart of the Newton-Raphson method in polar coordinates

In the following, the necessary formulas for setting up and calculating the Jacobi matrix are given.

Jacobi matrix	$\begin{bmatrix} \vdots \\ \Delta P_i \\ \vdots \\ \Delta Q_i \\ \vdots \end{bmatrix} = \begin{bmatrix} \underline{H} & \underline{N} \\ \underline{M} & \underline{L} \end{bmatrix} \begin{bmatrix} \vdots \\ \Delta \delta_i \\ \vdots \\ \Delta U_i \\ \vdots \end{bmatrix}$
H	$\frac{\partial P_i}{\partial \delta_i} = h_{ii} = -\sum_{j \neq i} U_i U_j y_{ij} \sin(\delta_i - \delta_j - \theta_{ij})$ $\frac{\partial P_i}{\partial \delta_j} = h_{ij} = U_i U_j y_{ij} \sin(\delta_i - \delta_j - \theta_{ij})$
M	$\frac{\partial Q_i}{\partial \delta_i} = m_{ii} = \sum_{j \neq i} U_i U_j y_{ij} \cos(\delta_i - \delta_j - \theta_{ij})$

	$\frac{\partial Q_i}{\partial \delta_j} = m_{ij} = -U_i U_j y_{ij} \cos(\delta_i - \delta_j - \theta_{ij})$
N	$\frac{\partial P_i}{\partial U_i} = n_{ii} = 2U_i y_{ij} \cos(\theta_{ii}) + \sum_{j \neq i} U_j y_{ij} \cos(\delta_i - \delta_j - \theta_{ij})$ $\frac{\partial P_i}{\partial U_j} = n_{ij} = U_i y_{ij} \cos(\delta_i - \delta_j - \theta_{ij})$
L	$\frac{\partial Q_i}{\partial U_i} = \tilde{l}_{ii} = l_{ii} U_i = 2U_i^2 y_{ii} \sin(-\theta_{ii}) - h_{ii}$ $\frac{\partial Q_i}{\partial U_j} = \tilde{l}_{ij} = l_{ij} U_j = h_{ij}$

1. Open the function nbtp_lfnr.m and add the start values under "Task 1: Initialization".

   Tip 1: Uses the Matlab functions: zeros(), abs(), angle()
   Tip 2: Ssoll corresponds to the node powers, since we only calculate with PQ nodes in the PQ-PFC.

2. Add the convergence query. You can insert it directly under the definition of epsilon.

   Tip 3: The if function terminates when a "break" is used.

3. Complete the Jacobian matrix with the help of the above formulas. First create the vector h, m, n and l and then correct the "wrong" value at the position (in,in) in the following code line.

   Tip 4: Example for h with in=1: h(1,:) = [h11, h12, h13, h14, h15]. With the next code line h(in,in) the "wrong-calculated value" h11 is then replaced by the correct calculation. This procedure is intended to reduce the programming effort.
   Tip 5 Calculation $\boldsymbol{h_{ij}, m_{ij}}$ **and** $\boldsymbol{n_{ij}}$ **:** Clarify how h_{ii}, m_{ii} and n_{ii} are calculated! Use the vectors calculated in the previous step (h_{ij}, m_{ij} and n_{ij}).
   Tip 6: Uses the Matlab function sum() and the element-wise multiplication .*
   Tip 7: [1;1;1] + 4 = [5;5;5]

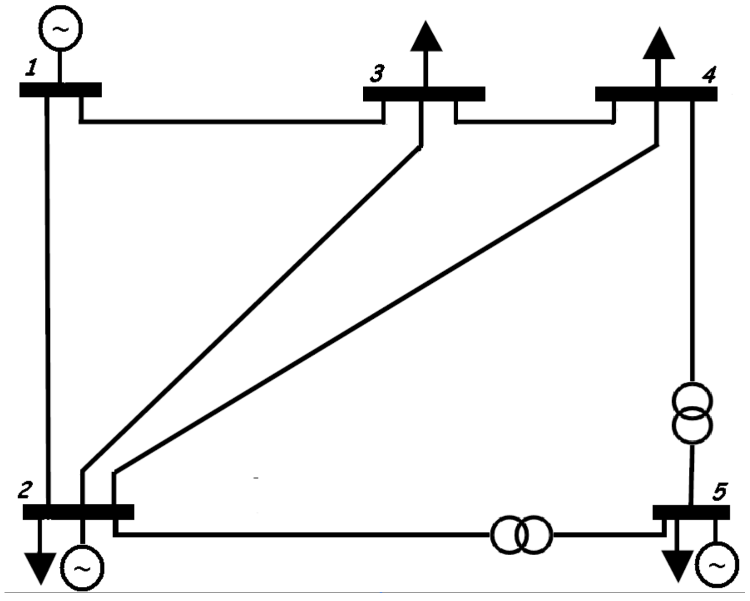
4. Now remove the reference node from the Jacobian matrix.

5. Now select "PQ-PFC only" in the GUI and press the Start button. To check the correctness of the results, you can compare the correct values displayed in red with the blue values calculated by you OR to get a quick overview, you can display the results by calling the RMSE view of the GUI.

4 Appendix

1. Test grid image
2. Overview of important Matlab constructs

4.1 Test grid image



4.2 Overview of important Matlab constructs

help *[something*]	%Help for command/function 'something'**.**
A=3;	%Matrix A consists of a single element with value 3 (All variables are represented as matrices in Matlab).
A1=3+i*6;	%A1 is complex: Re{A1}=3, Im{A1}=6
A2=3*exp (j*45)	%A2 is complex: magnitude=3, angle=45°
B=[1,2,3];	%B is a vector with 3 elements
C=[1,2,3;4,5,6];	%C is a matrix with 2 rows and 3 columns
D=[];	%D is an empty matrix
x=inf;	%x is equal to infinity
x=nan;	%Value of x is undefined e.g. result of a division by 0
x=B(i);	%x has the value of the i-th element of the vector B
x=C(i,j);	%x has the value of the element in the i-th row and the j-th column of the matrix C
x=C(i,:);	%x is a vector and contains all values of the i-th row of C
x=C(i,k:l);	%x is a vector and contains all values of the i-th row that belong to the columns k to l
E=zeros(M,N);	% (M x N) zero matrix (M x M if no N specified), i.e. E(i,j)=0
E=ones(M,N);	% (M x N) matrix (M x M if no N specified) with E(i,j)=1
E=[1:1:10];	% corresponding to E=[1,2,3,4,5,6,7,8,9,10] (general E=[a:n:b] and a, b, n real numbers)
[n,m]=size(C);	%Dimensions of the matrix C
l=length(B);	%length of the vector/matrix B
3*B;	%all values of B are multiplied by 3
B'	%Transpose a matrix/vector
C=B1 .* B2;	%C(i,j) = B1(i,j)*B2(i,j) (Necessary condition: Dim(B1) = Dim(B2))
x^3;	% x^3
C=B.^3;	%C(i,j) = B(i,j)^3
B^-1;	%Inverse matrix of B
conj(U);	%Conjugate complex matrix to U
abs(z);	%amount of the complex number z
angle(z);	%angle of the complex number z
real(z);	%real part of the complex number z
imag(z);	%Imaginary part of the complex number z
mod(x,y);	%modulo(x,y) (in C++ syntax: x%y)
x=sum(B);	% $x=\sum_i b_i$
x=find(B>1);	%x is a vector which contains all indices of those elements of B which satisfy the condition bi>1 (for matrices also notation (z,s)=find(C>1) is possible).
[R1,R2,...] =BelFun(matrix,vector,variable,const...);	%Function call of a function BelFun(...); the elements R1, R2 etc. are assigned the return of the function.
function[R1,R2...]=BelFun(Matrix,Vector,Variable,const)	%Function declaration
if (x~=1)...	% if-else statement (condition true if x is not equal to 1)
elseif(x>=1|x==4&h<4)	%(condition true if (x >=1) or (x=4 and h<4))
... else ... end;	%termination/termination of the statement
for (i = 1:n)...end;	%count loop: i is incremented by 1 in each iteration, i runs here from 1:n
while(...)...end;	%While loop
diag(ones(n,1));	%creates a unit matrix with dimension n x n
sort(C);	%sorts elements of C
isempty(C);	%true if C contains no elements
sin(A),cos(a) ...	%sin, cos... of a number or each element of the matrix