

## State Estimation with Matlab®

### Introduction

In this practical course, state estimation is first dealt with in linear form. A simple test grid with measured variables is provided. The linear state estimation must then be completely programmed and examined for changes in the input variables. The second part of the practical course deals with a 3-node grid in which measured values are also specified. Here, the state estimation for energy grids is to be programmed in parts.

### Table of contents

<b>1 Basics.....</b>	<b>2</b>
1.1 General description .....	2
1.2 Derivation of the target function .....	3
1.3 State Estimation Algorithm and Preparation .....	4
<b>2 Programming the linear state estimation .....</b>	<b>7</b>
<b>3 Programming the state estimation for a 3 node grid .....</b>	<b>8</b>
<b>4 Overview of important Matlab constructs .....</b>	<b>11</b>

### Preparation

- Familiarize yourself with the equations for linear state estimation and state estimation for energy grids. (Lecture 5 and exercise 3)
- Basic knowledge of programming in Matlab is assumed. ("Getting Started" book about Matlab or the tutorial)

## 1 Basics

### 1.1 General description

State estimation can be used to

- calculate the current system state
- compensate for the influence of measurement errors
- calculate missing grid variables
- eliminate bad measurement data
- monitor the grid structure

i.e. create a complete and reliable data set of the system state.

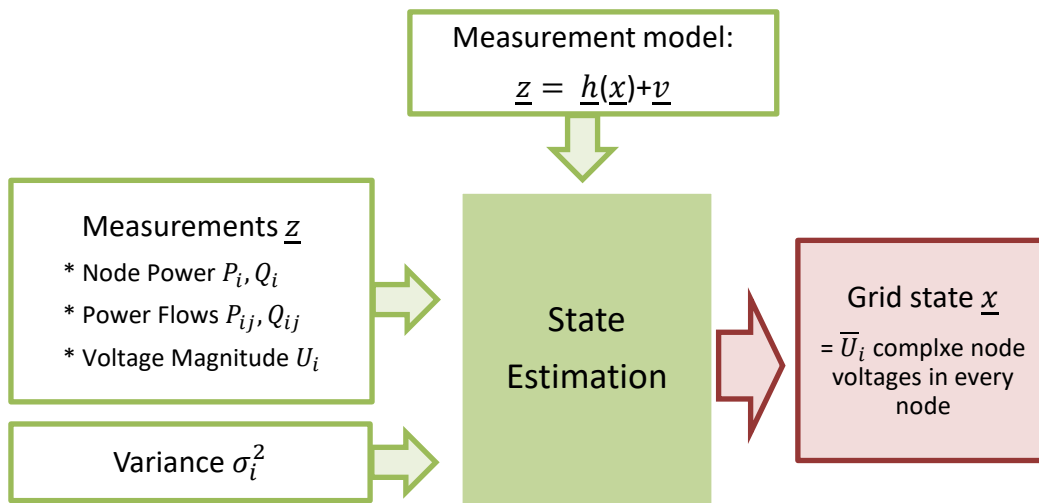


Figure 1.1: Input and output of the state estimation

The input parameters of the model can be divided into two categories:

The first category contains measured values that are recorded across the grid. As measured values are always subject to errors (tolerance ranges of the measuring devices, calibration errors, transmission errors, etc.), they can be weighted using variances. The following applies: the smaller the variance, the greater the weighting. If a measured value can be assumed to be very accurate, a small value is selected for the variance. The second category of input parameters includes the measurement model itself:  $\underline{z} = \underline{h}(\underline{x}) + \underline{v}$ . The faulty grid measurements  $\underline{z}$  are equated with the measurement model matrix  $\underline{h}(\underline{x})$  and the measurement error vector  $\underline{v}$ . For error-free measurements, the measurement error vector would correspond to a zero vector.



$\underline{x}$ : State vector  
(complexe node voltages)  
 $\underline{h}$ : Measurement model matrix  
(Grid topology, grid parameters)  
 $\underline{v}$ : Measurement fault vector  
 $\underline{z}$ : Measurement vector

Figure 1.2: Measurement model for calculating the state vector

## 1.2 Derivation of the target function

The following function is derived  $\frac{dJ(\underline{x})}{d\underline{x}} = 0$

### Inaccurate measurements and their effects

Assumption: Measurements  $\underline{z}$  are inaccurate!

Effect: With inaccurate measurements, the grid equations  $P_i, Q_i, P_{ij}, Q_{ij}, U_i$  do not work out. This means that the grid state cannot be clearly determined. (Grid state:  $\bar{U}_i \forall i$ )

Solution: An auxiliary element is used, which ensures that the measured values can be assumed to be non-optimal. The auxiliary element is the measurement error vector  $\underline{v}$ , which is inserted in the measurement model  $\underline{z} = \underline{h}(\underline{x}) + \underline{v}$  so that the grid equations have a unique solution.

### Minimization of the assumed measurement error

Assumption for  $\underline{v}$ : As it can be assumed that the measured values correspond approximately to the actual values, the measurement error vector  $\underline{v}$  should be minimized. In addition, the measured values can be weighted if it is known which measuring devices are particularly accurate or inaccurate.

Solution: Minimization is performed using the square of the measurement error vector  $\underline{v}$  so that positive and negative values do not cancel each other out. The weighting by different accuracies is carried out by multiplication with the inverted covariance matrix  $R^{-1}$ . This results in the objective function  $J(\underline{x}) = \underline{v}^T R^{-1} \underline{v} = \min$ . Minimization is performed by deriving the objective function from the state vector  $\underline{x}$ :  $\frac{dJ(\underline{x})}{d\underline{x}} = 0$

### Non-linear grid equations as a problem

Problem: As the grid equations are non-linear equations, simple differentiation is not possible.

Solution: For this case, the state estimation was developed using the Taylor series expansion.

### State Estimation

Calculation: Using the iterations of the state estimation a voltage vector  $\hat{\underline{x}}$  is calculated that satisfies the grid equations and minimizes the measurement fault vector  $\underline{v}$ . If the state vector  $\hat{\underline{x}}$  is now inserted into the grid equations, an improved measurement vector  $\hat{\underline{z}} = \underline{h}(\hat{\underline{x}})$  is obtained.

### 1.3 State Estimation Algorithm and Preparation

The algorithm is first presented below. The individual elements used by the algorithm are then discussed.

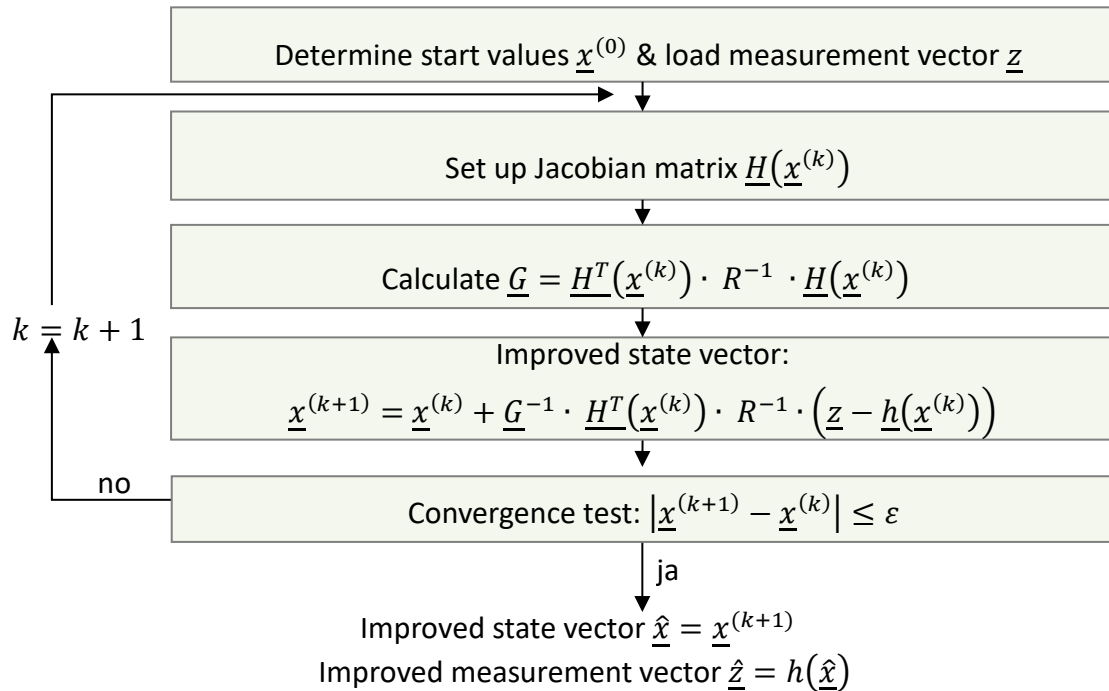


Figure 1.3: State Estimation Algorithm

Before the algorithm can be started, variables/vectors/matrices must be defined. The table shows all variables that are defined in the following.

Given	
Measurement vector (inaccurate)	$\underline{z}$
Inverted covariance matrix with standard deviations $\sigma$	$\underline{R}^{-1}$
Convergence threshold	$\varepsilon$
Angle resp. imaginary part of the voltage at node 1	$f_1 = 0$
Start values of the state vector	$\underline{x}^{(0)}$
Vector with grid equations	$\underline{h}(\underline{x}^{(k)})$
Jacobi matrix (Differentiated grid equations)	$\underline{H}(\underline{x}^{(k)})$
Searched	
State vector	$\hat{\underline{x}}$

Start values & convergence threshold

$$f_1 = 0 \quad \underline{x}^{(0)} = \begin{bmatrix} f_1 \\ f_2 \\ \vdots \\ e_1 \\ e_2 \\ \vdots \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 1 \\ 1 \\ \vdots \end{bmatrix} \text{ with } \bar{U}_1 = e_1 + j f_1 \quad \varepsilon = 10^{-4}$$

Measurement model:  $\underline{z} = \underline{h}(\underline{x}) + \underline{v}$

$$\underline{z} = \begin{bmatrix} P_i \\ Q_i \\ P_{ij} \\ Q_{ij} \\ U_i \end{bmatrix} \text{ measured values} \quad \underline{h}(\underline{x}) = \begin{bmatrix} P_i \\ Q_i \\ P_{ij} \\ Q_{ij} \\ U_i \end{bmatrix} \text{ with grid equations} \quad \underline{v} = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \end{bmatrix}$$

Part of the objective function:  $\underline{G} = \underline{H}^T(\underline{x}) \cdot R^{-1} \cdot \underline{H}(\underline{x})$

$$\underline{H}^T(\underline{x}) = \begin{bmatrix} \frac{dP_i}{df_i} & \frac{dP_i}{de_i} \\ \frac{dQ_i}{df_i} & \frac{dQ_i}{de_i} \\ \frac{dP_{ij}}{df_i} & \frac{dP_{ij}}{de_i} \\ \frac{dQ_{ij}}{df_i} & \frac{dQ_{ij}}{de_i} \\ \frac{dU_i}{df_i} & \frac{dU_i}{de_i} \end{bmatrix} = \begin{bmatrix} \frac{dP_1}{df_2} & \frac{dP_1}{df_3} & \frac{dP_1}{df_4} & \frac{dP_1}{de_1} & \frac{dP_1}{de_2} & \frac{dP_1}{de_3} \\ \frac{dP_2}{df_2} & \frac{dP_2}{df_3} & \frac{dP_2}{df_4} & \frac{dP_2}{de_1} & \frac{dP_2}{de_2} & \frac{dP_2}{de_3} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \frac{dQ_i}{df_2} & \dots & \dots & \frac{dQ_i}{de_1} & \dots & \dots \\ \frac{dP_{ij}}{df_2} & \dots & \dots & \frac{dP_{ij}}{de_1} & \dots & \dots \\ \frac{dQ_{ij}}{df_2} & \dots & \dots & \frac{dQ_{ij}}{de_1} & \dots & \dots \\ \frac{dU_i}{df_2} & \dots & \dots & \frac{dU_i}{de_1} & \dots & \dots \\ \frac{dU_i}{df_2} & \dots & \dots & \frac{dU_i}{de_1} & \dots & \dots \end{bmatrix} \quad R^{-1} = \begin{bmatrix} \frac{1}{\sigma_1^2} & 1 & \dots \\ 1 & \frac{1}{\sigma_2^2} & \dots \\ \vdots & \vdots & \ddots \end{bmatrix}$$

Grid equations:

$$P_i = \sum_{j \neq i} P_{ij} = (e_i^2 + f_i^2) \sum_{j \neq i} G_{ii}^j + \sum_{j \neq i} ((e_i e_j + f_i f_j) G_{ij} - (e_i f_j - e_j f_i) B_{ij})$$

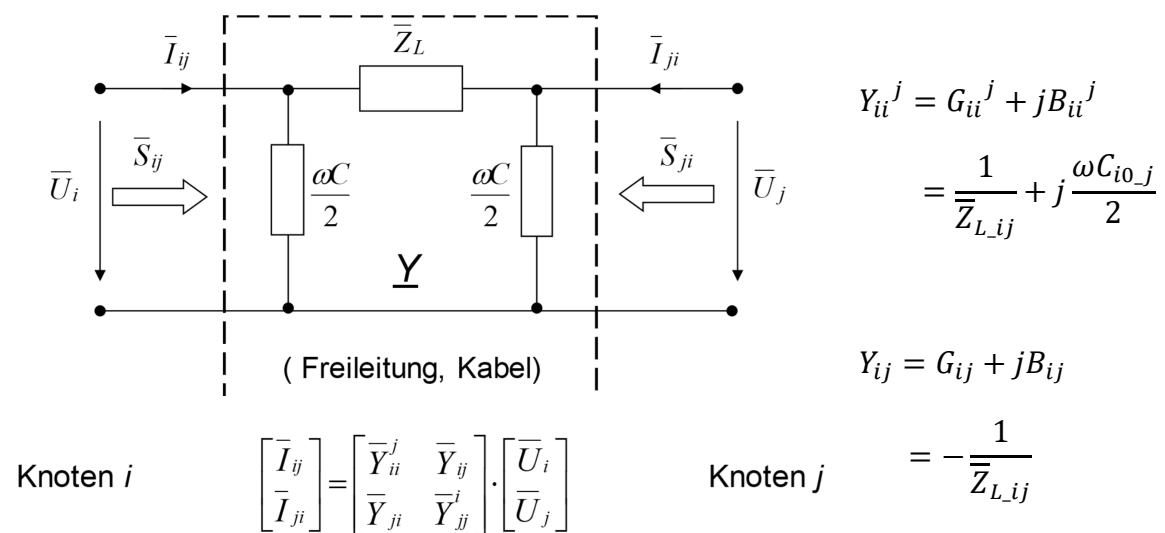
$$Q_i = \sum_{j \neq i} Q_{ij} = -(e_i^2 + f_i^2) \sum_{j \neq i} B_{ii}^j + \sum_{j \neq i} (-(e_i e_j + f_i f_j) B_{ij} - (e_i f_j - e_j f_i) G_{ij})$$

$$P_{ij} = (e_i^2 + f_i^2) G_{ii}^j + (e_i e_j + f_i f_j) G_{ij} - (e_i f_j - e_j f_i) B_{ij}$$

$$Q_{ij} = -(e_i^2 + f_i^2) B_{ii}^j - (e_i e_j + f_i f_j) B_{ij} - (e_i f_j - e_j f_i) G_{ij}$$

$$U_i = \sqrt{(e_i^2 + f_i^2)} \text{ mit } \bar{U}_i = e_i + j f_i$$

Attention: The two-port representation is decisive for the use of the line parameters in the equations (NOT AS IN THE LOAD FLOW CALCULATION!!):



Example: Line parameters in Matlab (cross elements neglected):

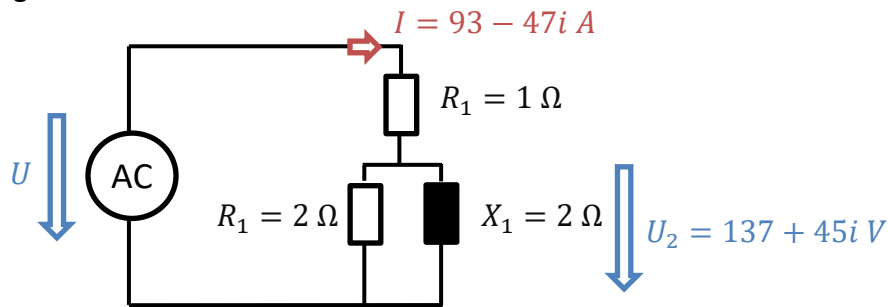
```
Zij = ZL = R + (1i*length*line_constant*2*pi()*50);
Yii_j = 1/Zij;
Yij = -1/Zij;
Gii_j = real(Yii_j);
Gij = real(Yij);
Bii_j = imag(Yii_j);
Bij = imag(Yij);
```

Example: Grid equations in Matlab:

```
Pij = (ei^2+fi^2)*Gii_j + (ei*fj+fi*fj)*Gij - (ei*fj-ej*fi)*Bij;
Qij = -(ei^2+fi^2)*Bii_j - (ei*fj+fi*fj)*Bij - (ei*fj-ej*fi)*Gij;
Ui = sqrt(ei^2+fi^2);
```

## 2 Programming the linear state estimation

Given is the grid below:



The total current  $I$  and the voltage  $U_2$  are given as measured variables. The variances  $\sigma_i^2$  are assumed to be  $1 A^2$  and  $1 V^2$ .

### Task:

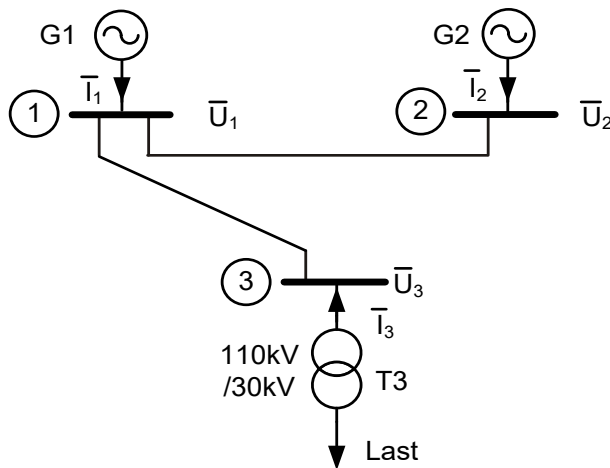
- 1) Think first before you start programming:
  - a. Set up the equations for  $I$  and  $U_2$ ? Remember that the quantity  $U$  you are looking for must be contained in both equations!
  - b. Set up the measurement model  $\underline{z} = \underline{H} \underline{x} + \underline{v}$  by defining the measurement vector  $\underline{z}$ , the vector with the measurement model  $\underline{H}$  and the fault vector  $\underline{v}$ .
- 2) Open Matlab and create a new script (Home → "New Script")
- 3) Program the linear state estimation
  - a. Define the vectors  $\underline{z}$  and  $\underline{H}$  and the inverted covariance matrix  $\underline{R}^{-1}$ . Also define the required variable  $\underline{x}$  with the syms function.
  - b. The objective function  $J(\underline{x})$  must then be set up and derived.
  - c. The system can be solved using the eqn function and the solve function.
  - d. The improved measured value  $\hat{\underline{z}} = \underline{H} \hat{\underline{x}}$  and the weighted error sum of squares  $J(\underline{x}) = \underline{v}^T \underline{R}^{-1} \underline{v}$  can then be calculated.

Covariance matrix with standard deviation $\sigma_i$  $\underline{R}^{-1} = \begin{bmatrix} \frac{1}{\sigma_1^2} & 0 \\ 0 & \frac{1}{\sigma_2^2} \end{bmatrix}$	<b>Helpful Matlab functions</b>	
Objective function ( $\underline{H}$ in this case is the measurement model vector)  $J(\underline{x}) = \underline{v}^T \underline{R}^{-1} \underline{v}$ $= (\underline{z} - \underline{H} \underline{x})^T \underline{R}^{-1} (\underline{z} - \underline{H} \underline{x})$	Generate the searched quantity  Derivation of the objective function	syms x  diff(J)  eqn = (dJ == 0)  x_searched = solve(eqn,x)
	Objective function = 0  Calculation of the searched quantity	

- 4) Now run the script. Which value is calculated for  $U$ ?
- 5) What happens when the actual values are used? ( $I = 92 - 46i$  and  $U_2 = 138 + 46i$ )  
Does the algorithm calculate  $U = 230 + 0i$ ?
- 6) What happens if the accuracy is changed (change standard deviation)?
- 7) What happens if the measured values are changed (e.g. increase the real part of I)?

### 3 Programming the state estimation for a 3 node grid

The following grid is given with the specified line parameters. Cross elements of the lines are neglected in this example.



#### Line parameters

Length of line 1-2 = 54.881km

Length of line 1-3 = 62.198km

$r' = 0.08 \text{ Ohm/km}$

$l' = 1.206e-3 \text{ H/km}$

#### Task:

The document "PSOS\_SE\_3nodalgrid\_student.m" contains parts of the state estimation that need to be completed. The document is divided into sections for this purpose:

#### Preparation sections

0. Searched variables  $e_1, e_2, e_3, f_2, f_3$  for  $\underline{x}$
1. Measurements  $P_i, Q_i, P_{ij}, Q_{ij}, U_i$  for  $\underline{z}$
2. Line parameters
3. Grid equations  $P_i, Q_i, P_{ij}, Q_{ij}, U_i$
4. Derivation of the grid equations  $P_i, Q_i, P_{ij}, Q_{ij}, U_i$  by  $e_1, e_2, e_3, f_2, f_3$

#### State Estimation

5. Measurement vector  $\underline{z}$ , measurement model matrix  $\underline{h}(\underline{x})$ , Jacobian matrix  $\underline{H}(\underline{x})$  and standard deviation  $R^{-1}$
6. Iteration algorithm

#### Evaluation

7. Command window and plots

Parts must now be added to the sections. Please follow the instructions below, which will guide you step by step through the code.



**Instruction:**

Open the Matlab file "PSOS\_SE\_3nodalgrid\_student.m" and familiarize yourself with the structure of the script (sections).

0. In section 0 the required variables must be defined. These are used later when creating the power flow equations.

To use variables in Matlab, the "syms" function can be used. By entering "syms x1 x2", 2 variables x1 and x2 can be created so that a function  $y(x1, x2)$  can be defined and solved. Now apply the syms function to the required variables  $e_1$ ,  $e_2$ ,  $e_3$ ,  $f_2$ ,  $f_3$  and set  $f_1 = 0$ .

**Hint:** Matlab function: syms variable1 variable2

1. The measured values are given in section 1. There is nothing to do here.
2. In section 2, the admittances Z12 and Z13 as well as some conductances G and susceptances B are already given. However, the missing conductance values G and susceptances B still need to be programmed.

Follow the two-port representation and the equations given in the basic chapter.

**Hint:** Note: Pay attention to the minus characters in the two-port representation!

3. The grid equations are to be programmed in section 3.  
The grid equations contain the conductances and susceptances of the two-port representation and the variables that have already been generated with "syms". Use the equations given in the basic chapter.

**Hint:** Example:  $f_{P12} = (e_1^2 + f_1^2) * G_{11\_2} + (e_1 * e_2 + f_1 * f_2) * G_{12} - (e_1 * f_2 - e_2 * f_1) * B_{12}$ ;

4. In section 4, all grid equations are derived according to each variable (syms).  
The individual grid equations from the previous section are now derived individually according to  $e_1$ ,  $e_2$ ,  $e_3$ ,  $f_2$ ,  $f_3$ . For each grid equation, the new equations can be saved in a row vector, which makes it easier to create  $H(x)$ .

**Hint:** Structure of row vector: rowP12 = [dP12f2, dP12f3, dP12e1, dP12e2, dP12e3];

5. The vectors z and h(x) and the matrices H(x) and R\_inv need to be defined in section 5.

- a. The measurement vector z consists of the measurements from section 1.
- b. The vector h(x) should now contain all the grid equations. Make sure that the order is the same as that of the measurement vector z.

**Hint:** Matlab function: h(searchedSize1, ...) = [function 1; function 2; ...]

- c. When creating the Jacobian matrix H(x), the row vectors from section 4 can be accessed. Make sure that the order is the same as that of the measurement vector.

**Hint:** Matlab function: H(searchedSize1, ...) = [function 1; function 2; ...]

- d. The inverted covariance matrix R\_inv is a diagonal matrix and should initially be filled with all entries = 1. Later, the measured values are weighted differently.

**Hint:** Matlab function: eye(length(z))

6. Section 6 contains the iterating algorithm. The start values  $\underline{x}^{(k)}$  and some other necessary variables are already defined in the code.
- As can be seen in Figure 3.1,  $\underline{H}(\underline{x}^{(k)})$  must be calculated first.  
**Hint:** Matlab function:  $H\_calculated = H(x\_k1, \dots, x\_k5)$   
**Hint:** Matlab function:  $X\_rounded = vpa(X, \text{decimal places})$
  - Now calculate  $\underline{G}$ , round  $\underline{G}$  and calculate the inverse of  $\underline{G}$ .  
**Hint:** Matlab function: Transpose a matrix  $M$ :  $M.'$   
**Hint:** Matlab function:  $G\_inverse = inv(G)$
  - First calculate  $\underline{h}(\underline{x}^{(k)})$ , round the vector and then calculate the improved state vector  $\underline{x}^{(k+1)}$   
**Hint:** Matlab function:  $h\_calculated = h(x\_k1, \dots, x\_k5)$

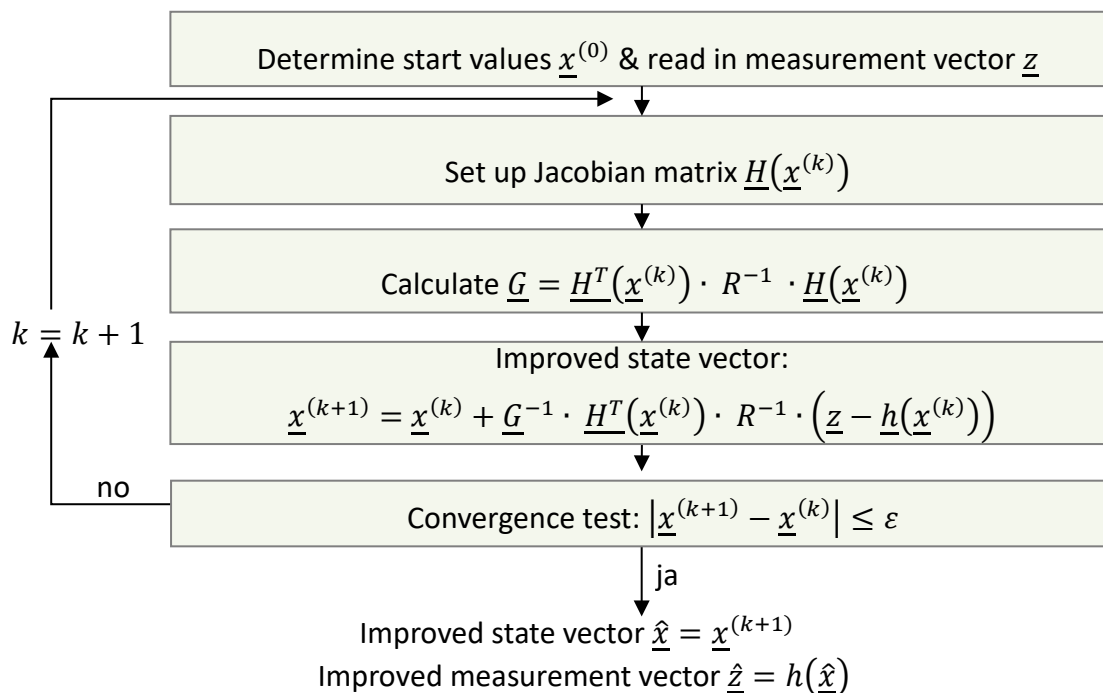


Figure 3.1: : State Estimation Algorithm

- The state estimation is now fully programmed. This should be tested. Use the "Run" button to start the simulation. What can be seen in the plots?
- The voltage value U2 should now not provide optimum measured values. U2 is 95% of the original value.
- In order to include the inaccuracy of the measuring device for U2 in the calculation, the weighting for U2 must be adjusted in the inverted covariance matrix. Set the value to 0.5 instead of 1.
- What happens if the value from exercise 9 is set to 0.000001 instead of 0.5?

## 4 Overview of important Matlab constructs

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