Appendix A
Solving Systems of Nonlinear Equations

Chapter 4 of this book describes and analyzes the power flow problem. In its ac version, this problem is a system of nonlinear equations. This appendix describes the most common method for solving a system of nonlinear equations, namely, the Newton-Raphson method. This is an iterative method that uses initial values for the unknowns and, then, at each iteration, updates these values until no change occurs in two consecutive iterations.

For the sake of clarity, we first describe the working of this method for the case of just one nonlinear equation with one unknown. Then, the general case of n nonlinear equations and n unknowns is considered.

We also explain how to directly solve systems of nonlinear equations using appropriate software.

A.1 Newton-Raphson Algorithm

The Newton-Raphson algorithm is described in this section.

A.1.1 One Unknown

Consider a nonlinear function $f(x) : \mathbb{R} \rightarrow \mathbb{R}$. We aim at finding a value of x so that:

$$f(x) = 0. \tag{A.1}$$

To do so, we first consider a given value of x, e.g., $x^{(0)}$. In general, we have that $f\left(x^{(0)}\right) \neq 0$. Thus, it is necessary to find $\Delta x^{(0)}$ so that $f\left(x^{(0)} + \Delta x^{(0)}\right) = 0$.

A.J. Conejo, L. Baringo, *Power System Operations*, Power Electronics and Power Systems, https://doi.org/10.1007/978-3-319-69407-8

Using Taylor series, we can express $f\left(x^{(0)}+\Delta x^{(0)}\right)$ as:

$$f\left(x^{(0)}+\Delta x^{(0)}\right)=f\left(x^{(0)}\right)+\Delta x^{(0)}\left(\frac{\mathrm{d}f\left(x\right)}{\mathrm{d}x}\right)^{(0)}+\frac{\left(\Delta x^{(0)}\right)^{2}}{2}\left(\frac{\mathrm{d}^{2}f\left(x\right)}{\mathrm{d}x^{2}}\right)^{(0)}+\ldots \tag{A.2}$$

Considering only the first two terms in Eq. (A.2) and since we seek to find $\Delta x^{(0)}$ so that $f\left(x^{(0)}+\Delta x^{(0)}\right)=0$, we can approximately compute $\Delta x^{(0)}$ as:

$$\Delta x^{(0)}\approx-\frac{f\left(x^{(0)}\right)}{\left(\frac{\mathrm{d}f\left(x\right)}{\mathrm{d}x}\right)^{(0)}}. \tag{A.3}$$

Next, we can update x as:

$$x^{(1)}=x^{(0)}+\Delta x^{(0)}. \tag{A.4}$$

Then, we check if $f\left(x^{(1)}\right)=0$. If so, we have found a value of x that satisfies $f(x)=0$. If not, we repeat the above step to find $\Delta x^{(1)}$ so that $f\left(x^{(1)}+\Delta x^{(1)}\right)=0$ and so on.

In general, we can compute $x^{(\nu)}$ as:

$$x^{(\nu+1)}=x^{(\nu)}-\frac{f\left(x^{(\nu)}\right)}{\left(\frac{\mathrm{d}f\left(x\right)}{\mathrm{d}x}\right)^{(\nu)}}, \tag{A.5}$$

where ν is the iteration counter.

Considering the above, the Newton-Raphson method consists of the following steps:

- **Step 0:** initialize the iteration counter ($\nu=0$) and provide an initial value for x, i.e., $x=x^{(\nu)}=x^{(0)}$.
- **Step 1:** compute $x^{(\nu+1)}$ using Eq. (A.5).
- **Step 2:** check if the difference between the values of x in two consecutive iterations is lower than a prespecified tolerance ϵ, i.e., check if $\left|x^{(\nu+1)}-x^{(\nu)}\right|<\epsilon$. If so, the algorithm has converged and the solution is $x^{(\nu+1)}$. If not, continue at **Step 3**.
- **Step 3:** update the iteration counter $\nu\leftarrow\nu+1$ and continue at **Step 1**.

Illustrative Example A.1 *Newton-Raphson algorithm for a one-unknown problem*

We consider the following quadratic function:

$$f\left(x\right)=x^{2}-3x+2,$$

whose first derivative is:

$$\frac{\mathrm{d}f(x)}{\mathrm{d}x} = 2x - 3.$$

The Newton-Raphson algorithm proceeds as follows:

- **Step 0:** we initialize the iteration counter ($\nu = 0$) and provide an initial value for x, e.g., $x^{(\nu)} = x^{(0)} = 0$.
- **Step 1:** we compute $x^{(1)}$ using the equation below:

$$x^{(1)} = x^{(0)} - \frac{\left(x^{(0)}\right)^2 - 3x^{(0)} + 2}{2x^{(0)} - 3} = 0 - \frac{0^2 - 3 \cdot 0 + 2}{2 \cdot 0 - 3} = 0.6667.$$

- **Step 2:** we compute absolute value of the difference between $x^{(1)}$ and $x^{(0)}$, i.e., $|0.6667 - 0| = 0.6667$. Since this difference is not small enough, we continue at **Step 3**.
- **Step 3:** we update the iteration counter $\nu = 0 + 1 = 1$ and continue at **Step 1**.
- **Step 1:** we compute $x^{(2)}$ using the equation below:

$$x^{(2)} = x^{(1)} - \frac{\left(x^{(1)}\right)^2 - 3x^{(1)} + 2}{2x^{(1)} - 3} = 0.6667 - \frac{0.6667^2 - 3 \cdot 0.6667 + 2}{2 \cdot 0.6667 - 3} = 0.9333.$$

- **Step 2:** we compute the absolute value of the difference between $x^{(2)}$ and $x^{(1)}$, i.e., $|0.9333 - 0.6667| = 0.2666$. Since this difference is not small enough, we continue at **Step 3**.
- **Step 3:** we update the iteration counter $\nu = 1 + 1 = 2$ and continue at **Step 1**.

This iterative algorithm is repeated until the difference between the values of x in two consecutive iterations is small enough. Table A.1 summarizes the results. The algorithm converges in four iterations for a tolerance of $1 \cdot 10^{-4}$.

Note that the number of iterations needed for convergence by the Newton-Raphson algorithm is small.

□

Table A.1 Illustrative Example A.2: results

Iteration	x
0	0
1	0.6667
2	0.9333
3	0.9961
4	1.0000

A.1.2 Many Unknowns

The Newton-Raphson method described in the previous section is extended in this section to the general case of a system of n nonlinear equations with n unknowns, as the one described below:

$$\begin{cases} f_1(x_1, x_2, \ldots, x_n) = 0, \\ f_2(x_1, x_2, \ldots, x_n) = 0, \\ \quad \vdots \\ f_n(x_1, x_2, \ldots, x_n) = 0, \end{cases} \tag{A.6}$$

where $f_i(x_1, x_2, \ldots, x_n) : \mathbb{R}^n \to \mathbb{R}$, $i = 1, \ldots, n$, are nonlinear functions.

The system of equations (A.6) can be rewritten in compact form as:

$$\mathbf{f}(\mathbf{x}) = \mathbf{0}, \tag{A.7}$$

where:

- $\mathbf{f}(\mathbf{x}) = [f_1(\mathbf{x}) \; f_2(\mathbf{x}) \; \ldots \; f_n(\mathbf{x})]^\top = \mathbf{0}$: $\mathbb{R}^n \to \mathbb{R}^n$,
- $\mathbf{x} = [x_1 \; x_2 \; \ldots \; x_n]^\top$,
- $\mathbf{0} = [0 \; 0 \; \ldots \; 0]^\top$, and
- $\top$ denotes the transpose operator.

Given an initial value for vector $\mathbf{x}$, i.e., $\mathbf{x}^{(0)}$, we have, in general, that $\mathbf{f}\left(\mathbf{x}^{(0)}\right) \neq \mathbf{0}$. Thus, we need to find $\Delta\mathbf{x}^{(0)}$ so that $\mathbf{f}\left(\mathbf{x}^{(0)} + \Delta\mathbf{x}^{(0)}\right) = \mathbf{0}$. Using the first-order Taylor series, $\mathbf{f}\left(\mathbf{x}^{(0)} + \Delta\mathbf{x}^{(0)}\right)$ can be approximately expressed as:

$$\mathbf{f}\left(\mathbf{x}^{(0)} + \Delta\mathbf{x}^{(0)}\right) \approx \mathbf{f}\left(\mathbf{x}^{(0)}\right) + \mathbf{J}^{(0)}\Delta\mathbf{x}^{(0)}, \tag{A.8}$$

where $\mathbf{J}$ is the $n \times n$ Jacobian:

$$\mathbf{J} = \begin{bmatrix} \dfrac{\partial f_1(\mathbf{x})}{\partial x_1} & \dfrac{\partial f_1(\mathbf{x})}{\partial x_2} & \cdots & \dfrac{\partial f_1(\mathbf{x})}{\partial x_n} \\ \dfrac{\partial f_2(\mathbf{x})}{\partial x_1} & \dfrac{\partial f_2(\mathbf{x})}{\partial x_2} & \cdots & \dfrac{\partial f_2(\mathbf{x})}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \dfrac{\partial f_n(\mathbf{x})}{\partial x_1} & \dfrac{\partial f_n(\mathbf{x})}{\partial x_2} & \cdots & \dfrac{\partial f_n(\mathbf{x})}{\partial x_n} \end{bmatrix}. \tag{A.9}$$

Since we seek $\mathbf{f}\left(\mathbf{x}^{(0)} + \Delta\mathbf{x}^{(0)}\right) = \mathbf{0}$, from Eq. (A.8) we can compute $\Delta\mathbf{x}^{(0)}$ as:

$$\Delta\mathbf{x}^{(0)} \approx -\left[\mathbf{J}^{(0)}\right]^{-1} \mathbf{f}\left(\mathbf{x}^{(0)}\right). \tag{A.10}$$

Then, we can update vector $\mathbf{x}$ as:

$$\mathbf{x}^{(1)} = \mathbf{x}^{(0)} + \Delta\mathbf{x}^{(0)}. \tag{A.11}$$

In general, we can update vector $\mathbf{x}$ as:

$$\mathbf{x}^{(\nu+1)} = \mathbf{x}^{(\nu)} - \left[\mathbf{J}^{(\nu)}\right]^{-1} \mathbf{f}\left(\mathbf{x}^{(\nu)}\right), \tag{A.12}$$

where ν is the iteration counter.

Considering the above, the Newton-Raphson algorithm consists of the following steps:

- **Step 0:** initialize the iteration counter ($\nu = 0$) and provide an initial value for vector $\mathbf{x}$, i.e., $\mathbf{x} = \mathbf{x}^{(\nu)} = \mathbf{x}^{(0)}$.
- **Step 1:** compute the Jacobian $\mathbf{J}$ using (A.9).
- **Step 2:** compute $\mathbf{x}^{(\nu+1)}$ using matrix equation (A.12).
- **Step 3:** check every element of the absolute value of the difference between the values of vector $\mathbf{x}$ in two consecutive iterations is lower than a prespecified tolerance $\boldsymbol{\epsilon}$, i.e., check if $\left|\mathbf{x}^{(\nu+1)} - \mathbf{x}^{(\nu)}\right| < \boldsymbol{\epsilon}$. If so, the algorithm has converged and the solution is $\mathbf{x}^{(\nu+1)}$. If not, continue at **Step 4**.
- **Step 4:** update the iteration counter $\nu \leftarrow \nu + 1$ and continue at **Step 1**.

For the sake of clarity, this iterative algorithm is schematically described through the flowchart in Fig. A.1.

Illustrative Example A.2 *Newton-Raphson algorithm for a two-unknown problem*

We consider the following system of two equations and two unknowns:

$$\begin{cases} f_1(x, y) = x + xy - 4, \\ f_2(x, y) = x + y - 3. \end{cases}$$

We aim at finding the values of x and y so that $f_1(x, y) = 0$ and $f_2(x, y) = 0$. To do so, we use the Newton-Raphson method.

First, we compute the partial derivatives:

$$\begin{cases} \dfrac{\partial f_1(x, y)}{\partial x} = 1 + y, \\ \dfrac{\partial f_1(x, y)}{\partial y} = x, \\ \dfrac{\partial f_2(x, y)}{\partial x} = 1, \\ \dfrac{\partial f_2(x, y)}{\partial y} = 1. \end{cases}$$

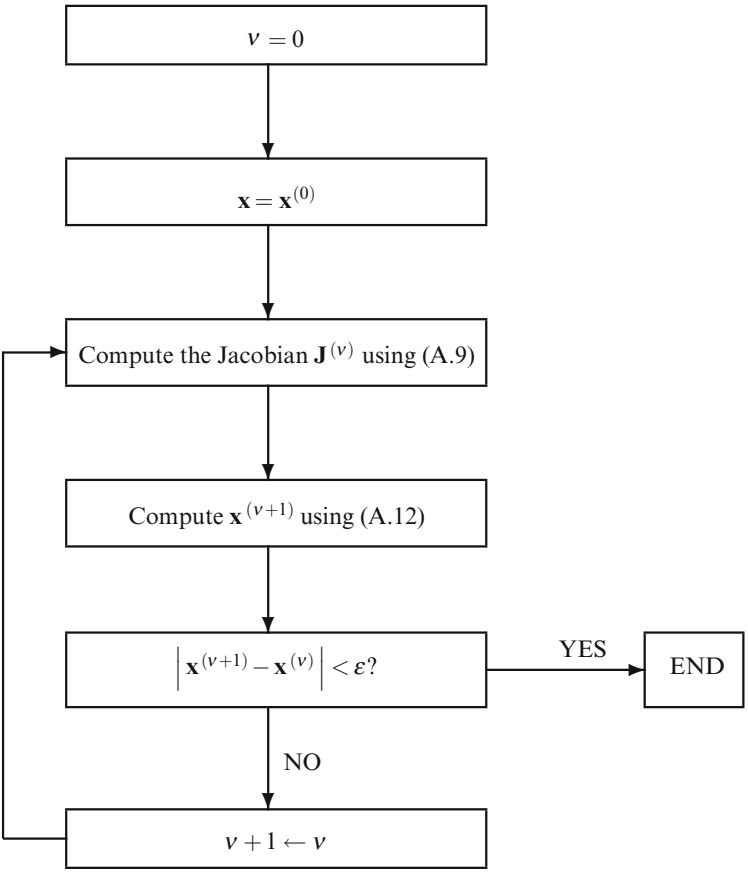

Fig. A.1 Algorithm flowchart for the Newton-Raphson method

Second, we build the Jacobian matrix:

$$\mathbf{J} = \begin{bmatrix} 1+y & x \\ 1 & 1 \end{bmatrix}.$$

Then, we follow the iterative procedure described above:

- **Step 0:** we initialize the iteration counter ($\nu = 0$) and provide initial values for variables x and y, e.g., $x^{(\nu)} = x^{(0)} = 1.98$ and $y^{(\nu)} = y^{(0)} = 1.02$, respectively.
- **Step 1:** we compute the Jacobian matrix $\mathbf{J}$ at iteration $\nu = 0$:

$$\mathbf{J}^{(0)} = \begin{bmatrix} 1+y^{(0)} & x^{(0)} \\ 1 & 1 \end{bmatrix} = \begin{bmatrix} 1+1.02 & 1.98 \\ 1 & 1 \end{bmatrix} = \begin{bmatrix} 2.02 & 1.98 \\ 1 & 1 \end{bmatrix}.$$

Table A.2 Illustrative Example A.2: results

Iteration	x	y
0	1.9800	1.0200
1	1.9900	1.0100
2	1.9950	1.0050
3	1.9975	1.0025
4	1.9987	1.0013
5	1.9994	1.0006
6	1.9997	1.0003
7	1.9998	1.0002
8	1.9999	1.0001
9	2.0000	1.0000

- **Step 2:** we compute $x^{(1)}$ and $y^{(1)}$ using the matrix equation below:

$$\begin{bmatrix} x^{(1)} \\ y^{(1)} \end{bmatrix} = \begin{bmatrix} x^{(0)} \\ y^{(0)} \end{bmatrix} - \begin{bmatrix} 1 + y^{(0)} & x^{(0)} \\ 1 & 1 \end{bmatrix}^{-1} \begin{bmatrix} x^{(0)} + x^{(0)} y^{(\nu)} - 4 \\ x^{(0)} + y^{(0)} - 3 \end{bmatrix}$$

$$= \begin{bmatrix} 1.98 \\ 1.02 \end{bmatrix} - \begin{bmatrix} 2.02 & 1.98 \\ 1 & 1 \end{bmatrix}^{-1} \begin{bmatrix} -4 \cdot 10^{-4} \\ 0 \end{bmatrix} = \begin{bmatrix} 1.9900 \\ 1.0100 \end{bmatrix}.$$

- **Step 3:** we compute the difference between $x^{(1)}$ and $x^{(0)}$, i.e., $|1.9900 - 1.98| = 0.01$, as well as the differences between $y^{(1)}$ and $y^{(0)}$, i.e., $|1.0100 - 1.02| = 0.01$. Since these differences are not small enough, we continue with **Step 4**.
- **Step 4:** we update the iteration counter $\nu = 0 + 1 = 1$ and continue with **Step 1**.

This iterative algorithm is repeated until the differences between the values of x and y in two consecutive iterations are small enough. Table A.2 provides the evolution of the values of these unknowns. The algorithm converges in nine iterations for a tolerance of $1 \cdot 10^{-4}$.

Note that the number of iterations needed by the Newton-Raphson algorithm is rather small.

Next, we consider a different initial solution. Table A.3 provides the results. In this case, the algorithm converges in 11 iterations for a tolerance of $1 \cdot 10^{-4}$.

We conclude that the initial solution does not have an important impact on the number of iterations required for convergence, provided that convergence is attained. However, convergence is not necessarily guaranteed, and the Jacobian may be singular at any iteration. Further details on convergence guarantee and on convergence speed are available in [1].

□

Table A.3 Illustrative Example A.2: results considering a different initial solution

Iteration	x	y
0	2.1000	0.9000
1	2.0500	0.9500
2	2.0250	0.9745
3	2.0125	0.9875
4	2.0062	0.9938
5	2.0031	0.9969
6	2.0016	0.9984
7	2.0008	0.9992
8	2.0004	0.9996
9	2.0002	0.9998
10	2.0001	0.9999
11	2.0000	1.0000

A.2 Direct Solution

Generally, the Newton-Raphson method does not need to be implemented. An off-the-self routine (in GNU Octave [2] or MATLAB [3]) embodying the Newton-Raphson algorithm can be used to solve systems of nonlinear equations.

Illustrative Examples A.1 and A.2 are solved below using GNU Octave routines.

A.2.1 One Unknown

The GNU Octave [2] routines below solve Illustrative Example A.1:

```
clc
fun = @NR1;
x0 = [0]; x = fsolve(fun,x0)
```

```
function F = NR1(x)
%
F(1)=x(1)*x(1)-3*x(1)+2;
```

The solution provided by GNU Octave is:

```
x =  1.00000
```

A.2.2 Many Unknowns

The GNU Octave routines below solve Illustrative Example A.2:

```
clc
fun = @NR2;
x0 = [1.98,1.02]; x = fsolve(fun,x0)
```

```
function F = NR2(x)
%
F(1)=x(1)+x(1)*x(2)-4;
F(2)=x(1)+x(2)-3;
```

The solution provided by GNU Octave is:

```
x =
   1.9994   1.0006
```

A.3 Summary and Further Reading

This appendix describes the Newton-Raphson method, which is the most common method for solving systems of nonlinear equations, as those considered in Chap. 4 of this book. The Newton-Raphson method is based on an iterative procedure that updates the value of the unknowns involved until the changes in their values in two consecutive iterations are small enough.

Different illustrative examples are used to show the working of the Newton-Raphson method. Additionally, this appendix explains also how to directly solve a system of nonlinear equations using appropriate software, such as GNU Octave [2].

Additional details can be found in the monograph by Chapra and Canale on numerical methods in engineering [1].

References

1. Chapra, S.C., Canale, R.P.: Numerical Methods for Engineers, 6th edn. McGraw-Hill, New York (2010)
2. GNU Octave (2016): Available at www.gnu.org/software/octave
3. MATLAB (2016): Available at www.mathworks.com/products/matlab