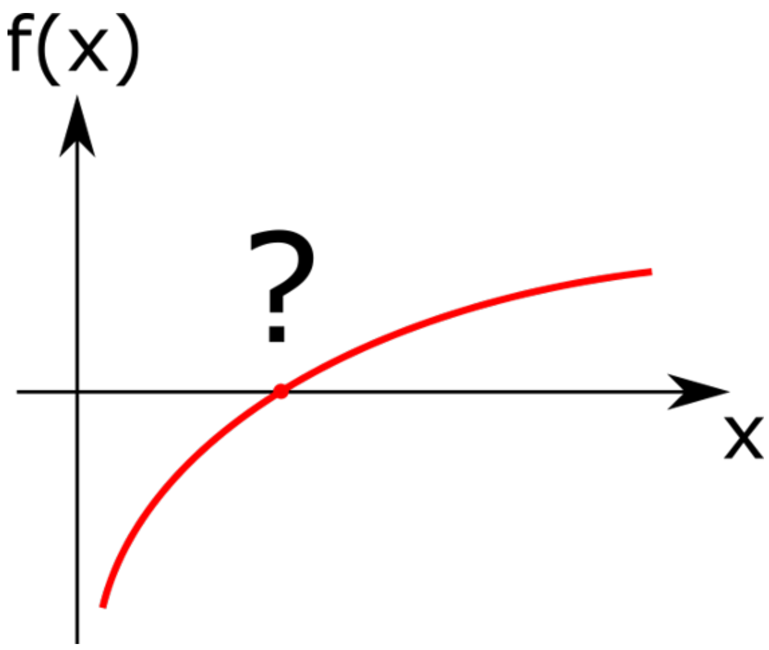
Computer Modelling Techniques

MMME3086 UNUK, 2023/24

Numerical Methods: Roots of equations

Author: Mirco Magnini
Office: Coates B100a
Email: mirco.magnini@nottingham.ac.uk



Contents

1 Introduction

At high school, you have probably learned how to find the roots of second-order polynomials such as $f(x) = ax^2 + bx + c$, using the formula:

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}. \tag{1}$$

In the engineering practice, you may encounter equations far more complicated than Eq. (1) above, be them higher-order polynomials or equations including trigonometric, exponential, logarithmic, and other, less familiar, functions. It is not always possible to derive an analytical solution for such complex equations, and therefore it is useful to utilise numerical methods to find the roots of these equations.

This lecture will present some powerful techniques to find roots of equations. In Section 2, we will start with presenting the Newton-Raphson method to find the roots of an equation with one unknown, $f(x) = 0$. Then, Section 3 will show you how to extend the method to a system of two equations with two unknowns; the two equations will be nonlinear, because in the case of a system of linear equations, the direct and iterative methods learned in Lecture 2 will be sufficient. Section 4 will extend the method to solve a system of n nonlinear equations with n unknowns. In the final Section 5, we will see how to implement the Newton-Raphson method in Matlab to solve the two examples proposed in these notes.

2 The Newton-Raphson method

We want to find the solution of the generic equation $f(x) = 0$. In general, equations in mathematics can be always recast in the form $f(x) = 0$, for example:

$$x^4 = 5 \Rightarrow f(x) = x^4 - 5 = 0, \tag{2}$$

or:

$$e^{-x} = x \Rightarrow f(x) = e^{-x} - x = 0. \tag{3}$$

Solving the equation means finding the root(s) of the equation. There exist many methods to find roots of an equation. These can be classified into *bracketing methods* and *open methods*:

- **Bracketing methods** exploit the fact that a function typically changes sign in the vicinity of a root. Two initial guesses for the root are required and these guesses must be located on each side of the root, so that the function takes opposite signs when evaluated at the two guesses. The root is then searched within this interval, which is iteratively narrowed down, but remaining always "bracketed" between two values where the function changes sign. These methods always converge to the correct root. **Further reading:** Chapra and Canale [1], Ch. 5.
- **Open methods** are based on formulas that require only a single starting value of x, or two starting values that do not necessarily bracket the root. As such, they sometimes diverge or move away from the true root as the computation progresses. However, when open methods converge,

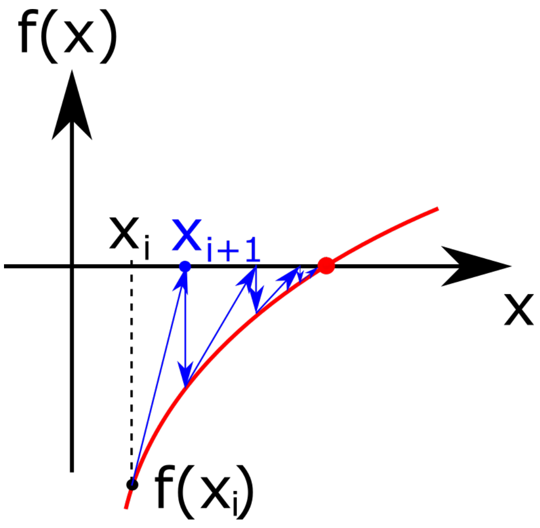


Figure 1: Schematic representation of the Newton-Raphson method, with the red curve representing $f(x)$ and the red dot representing a root of $f(x) = 0$. At a generic iteration i, where the guess value is x_i, the new guess value is obtained by extending the tangent line at x_i until it crosses zero, and then setting x_{i+1} to the abscissa of the zero crossing. As iterations continue, the procedure identified with the blue arrows should converge towards the correct solution.

they usually do so much more quickly than the bracketing methods. **Further reading:** Chapra and Canale [1], Ch. 6.

The **Newton-Raphson** method belongs to the family of open methods, and it is the most widely used numerical method to find the solution of nonlinear equations. The method works by using an initial guess or a trial solution, and then successively improving it by using iterations based on the slope (gradient) of the curve. To ensure convergence, the initial guess must be a reasonable one and must not be too far away from the exact solution. The curve is effectively approximated by a series of suitable tangents.

2.1 Outline of the method

Figure 1 shows a schematic representation of how the Newton-Raphson method can be used to arrive at the exact solution starting from an initial guess. At a generic iteration i, where the guess value is x_i, the new guess value is obtained by extending the tangent line at x_i until it crosses zero, and then setting x_{i+1} to the abscissa of the zero crossing. The iteration equation to find x_{i+1} is obtained by considering the first-order Taylor expansion of $f(x)$ near the previous guess value x_i:

$$f_T(x) = f(x_i) + (x - x_i)f'(x_i), \tag{4}$$

where the $f_T(x)$ ("T" for Taylor or tangent) above is a linear function of x and represents the equation of the tangent line passing by x_i, and therefore can be seen as one of the blue tangent segments in Fig. 1. This line crosses zero at some point where $f_T(x) = 0$, which identifies the abscissa x_{i+1}:

$$0 = f_T(x_{i+1}) = f(x_i) + (x_{i+1} - x_i)f'(x_i). \tag{5}$$

and, rearranging:

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}, \tag{6}$$

which is the iteration equation of the Newton-Raphson method.
Example. Consider the equation:

$$f(x) = e^{-x} - x = 0, \tag{7}$$

which is plotted in Fig. 4. Note that the function values span 4 orders of magnitude, from $f(x) \approx 10^4$ when $x = -10$, to $f(x) = -10$ when $x = 10$. Let's apply the Newton-Raphson method to find the root of the function. The derivative of $f(x)$ is:

$$f'(x) = -e^{-x} - 1, \tag{8}$$

Therefore, according to Eq. (6), the equation to use to find a new guess is:

$$x_{i+1} = x_i - \frac{e^{-x_i} - x_i}{-e^{-x_i} - 1}. \tag{9}$$

If we take the initial guess $x_0 = 0$, the next guess will be:

$$x_1 = 0 - \frac{e^0 - 0}{-e^0 - 1} = 0 - \frac{1 - 0}{-1 - 1} = 0.5. \tag{10}$$

If we continue, we will eventually converge to the solution, which is $x = 0.56714$. This can be done implementing a simple matlab code. Figure 3 shows the sequential values of the guesses as the iterations proceed, for three different values of the initial guess, $x_0 = 0$, $x_0 = 10$ and $x_0 = -10$. As a criterion to arrest the iterative procedure, we look at error, defined as $|f(x_i)|$; intuitively, it makes sense that the error is the difference between the value of the function calculated at x_i, and zero; as

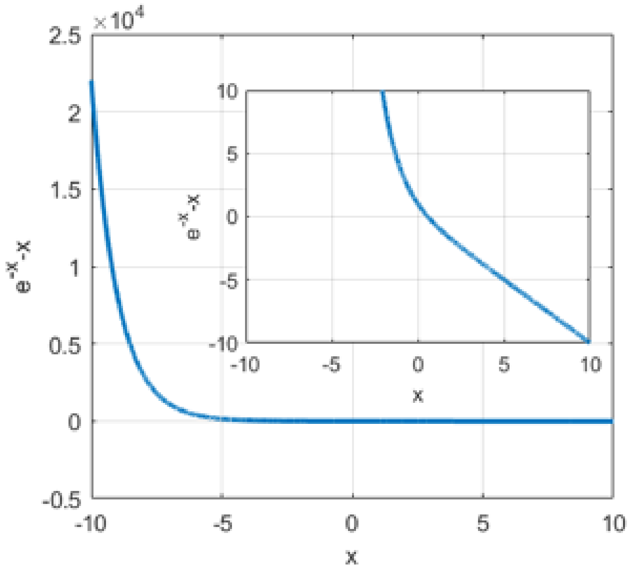
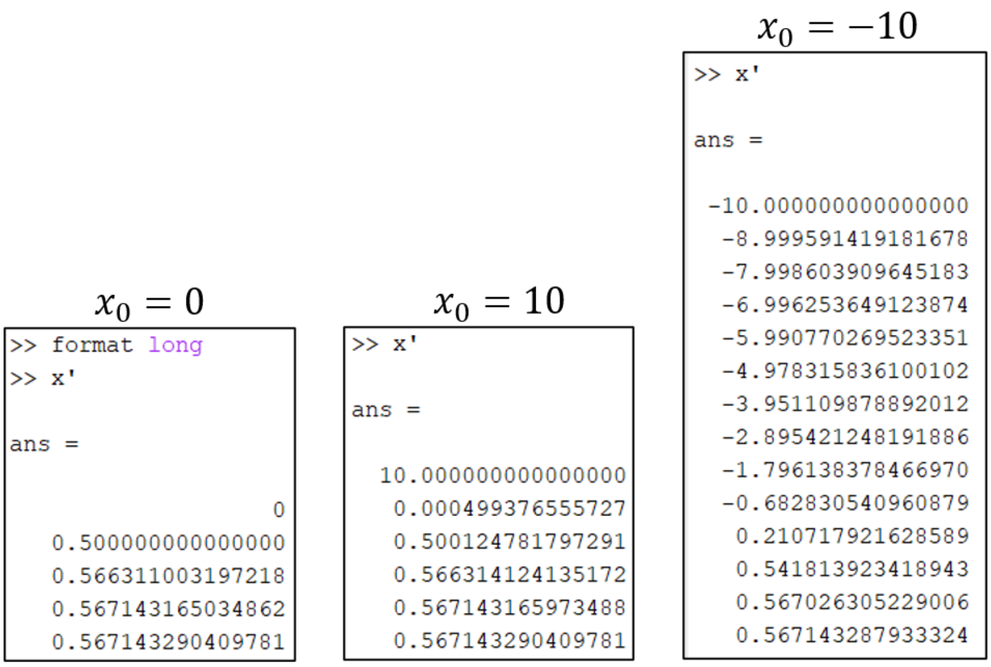


Figure 2: Plot of the function $f(x) = e^{-x} - x$; the inset shows a close-up view of the function near the root $x = 0.56714$.



$x_0 = 0$

```
>> format long
>> x'

ans =

                   0
   0.500000000000000
   0.566311003197218
   0.567143165034862
   0.567143290409781
```

$x_0 = 10$

```
>> x'

ans =

  10.000000000000000
   0.000499376555727
   0.500124781797291
   0.566314124135172
   0.567143165973488
   0.567143290409781
```

$x_0 = -10$

```
>> x'

ans =

 -10.000000000000000
  -8.999591419181678
  -7.998603909645183
  -6.996253649123874
  -5.990770269523351
  -4.978315836100102
  -3.951109878892012
  -2.895421248191886
  -1.796138378466970
  -0.682830540960879
   0.210717921628589
   0.541813923418943
   0.567026305229006
   0.567143287933324
```

Figure 3: Convergence of the solution for different values of the initial guess x_0.

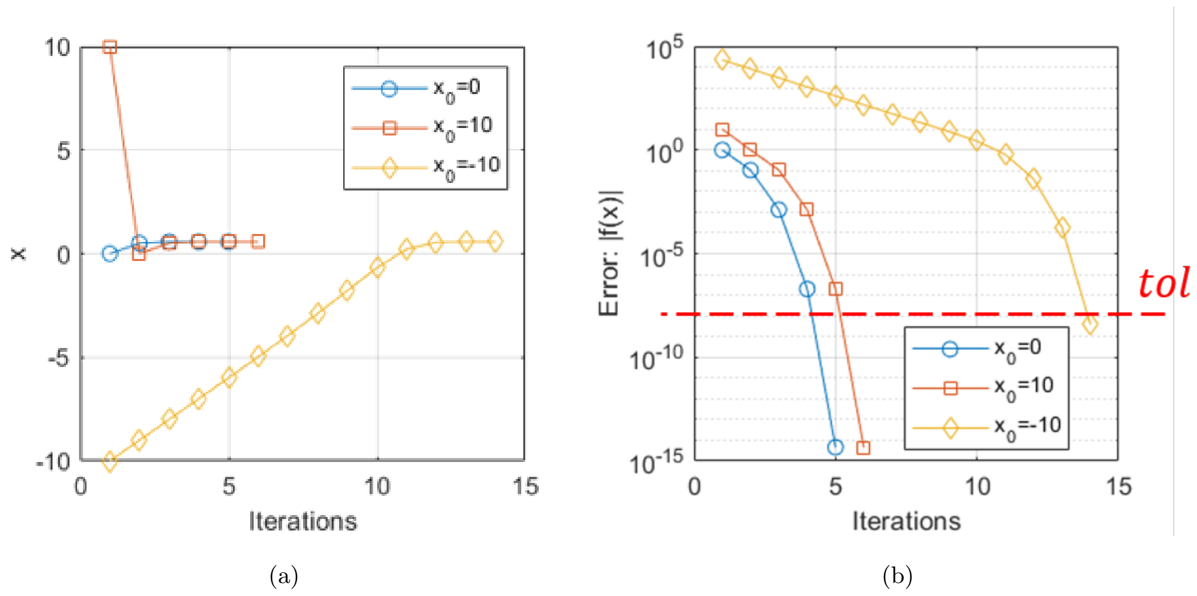


Figure 4: (a) Convergence of the solution for different values of the initial guess x_0, and (b) convergence of the error, calculated as $|f(x_i)| = |e^{-x_i} - x_i|$. The iterative procedure is arrested when the error falls below the tolerance, set to $tol = 10^{-8}$.

a tolerance value, we choose $tol = 10^{-8}$. The sequential values of the guesses and the corresponding errors are reported in Fig. 4. It can be seen that $x_0 = 0$ provides the fastest convergence, with 4 iterations being sufficient. The solution is achieved quickly also when starting with $x_0 = 10$ (5 iterations), whereas it takes 13 iterations to converge when $x_0 = -10$. The reason can be readily understood by looking at Fig. 3. When $x_0 = -10$, $f(x_0)$ is very far from zero, and thus it takes many steps to get there. When $x_0 = 10$, $f(x_0)$ is much closer to zero and, furthermore, the behavior of

$f(x)$ is almost linear between $x_0 = 10$ and the correct solution $x = 0.56714$, therefore after only one iteration (see Fig. 3) the guess is already very close to the final solution. The error convergence history reported in Fig. 4(b) emphasise another specific feature of the Newton-Raphson method. Once that the guess is sufficiently close to the actual solution, convergence accelerates and the error decreases at a faster rate; you see this in particular for $x_0 = -10$: when the error becomes below 1, the error reduces at a faster rate as iterations continue. This happens because it can be demonstrated that, with the Newton-Raphson method, when x_i is near a root the error at the next iteration e_{i+1} is proportional to e_i^2; this means that if at iteration i the error is $e_i = 0.1$, at the next iteration the error will be $e_{i+1} = 0.01$, and then $e_{i+2} = 0.0001$ and so on, i.e. the error reduces at an increasing rate. **Further reading** about this: Press et al., Sec. 9.4.

The first tutorial exercise at the end of this document will show how to solve this example using Matlab.

2.2 Pitfalls of the method

The pitfalls of the Newton-Raphson are:

1. The initial guess has to be "sufficiently" close to the root.
2. Convergence depends on the nature of the function, and in particular its derivative, see the four cases displayed in Fig. 5. Nearby an inflection point (Fig. 5(a)), where $f'' = 0$, the method actually gets further from the solution, and eventually diverges. Nearby a maximum or a minimum, iterations may oscillate (Fig. 5(b)) and eventually shoot out if the guess hits the point of zero derivative (Fig. 5(d)). If multiple solutions are present in the correspondence of a maximum or a minimum, the guess jumps to another root (Fig. 5(c)).
3. No bracketing is done, and hence divergence may occur.
4. Convergence is not guaranteed.
5. Needs knowledge of the first derivative.

The remedies to these issues are:

- Always set a max number of iterations to avoid entering an infinite loop.
- Check that the solution is converging during the iterative procedure, plotting $|f(x_i)|$ during runtime.
- Code your algorithm to output a warning message if the guess shoots out.
- If the derivative f' does not have an analytical expression, calculate the derivative using two successive values of the function. This turns the Newton-Raphson method into the **secant method**. **Further reading:** Chapra and Canale [1], Sec. 6.3.
- To avoid that the guess shoots out, first run a bracketing method to narrow down the search interval, and then use an open method once you are close to the solution, as open methods are much

faster once the solution is approached. This turns the Newton-Raphson method into the **Brent method**. **Further reading:** Chapra and Canale [1], Sec. 6.4. This is the method use by Matlab. Try typing in your command window: `fzero(@(x) exp(-x)-x,0,optimset('DISP','ITER'))`, and look at the output. Use `doc fzero` to see how the function works.

Further reading about the Newton-Raphson method: Chapra and Canale [1], Sec. 6.2.

3 The Newton-Raphson method for two nonlinear equations

Consider now that you have two equations with two unknowns to be solved simultaneously:

$$u(x,y) = 0, \quad v(x,y) = 0. \tag{11}$$

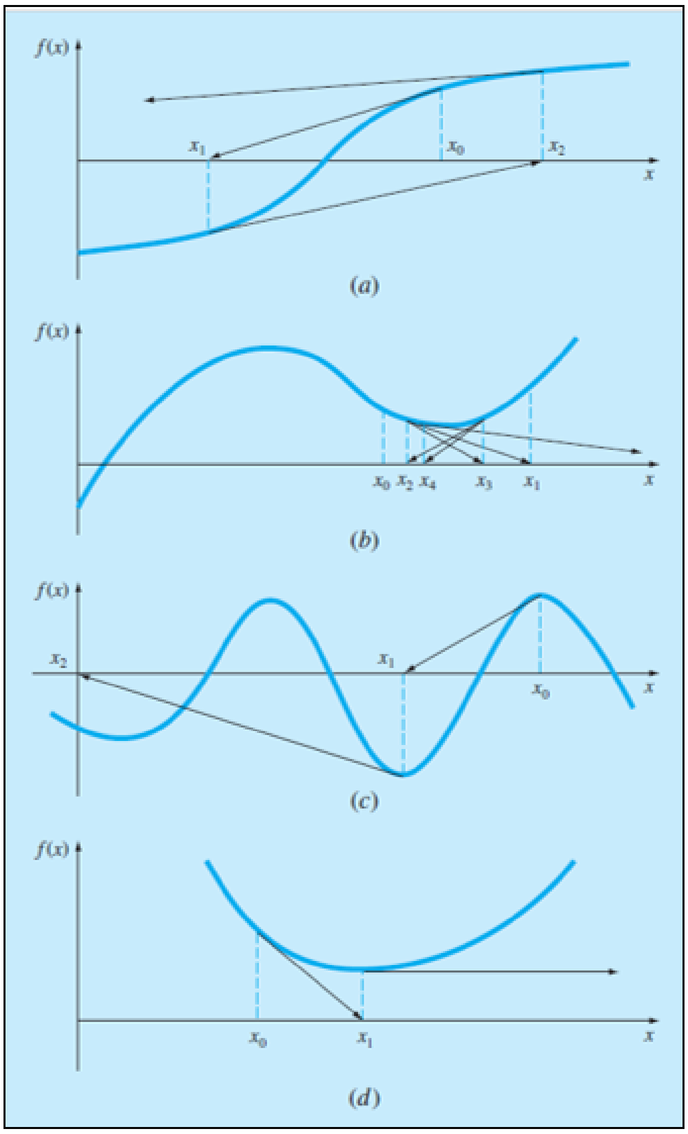


Figure 5: Four cases where the Newton-Raphson method exhibits poor convergence: (a) an inflection point, (b) a maximum/minimum, (c) multiple solutions, (d) a zero derivative. Source: Chapra and Canale [1].

If the two equations are linear, they form a linear system of two equations with two variables, that can be solved with the methods described in Lecture 2. Therefore, in this lecture we consider the case that the two equations are nonlinear, in which case the methods of Lecture 2 do not apply.

To solve a nonlinear system of two equations, we can use the Newton-Raphson method extended to two dimensions. As done for the one-dimensional case in the previous section, we start from the guess point (x_i, y_i) and we want to find the next guess point (x_{i+1}, y_{i+1}). We write the first-order Taylor expansion of both $u(x, y)$ and $v(x, y)$ near (x_i, y_i):

$$u_T(x,y) = u(x_i,y_i) + (x - x_i)\frac{\partial u}{\partial x}(x_i,y_i) + (y - y_i)\frac{\partial u}{\partial y}(x_i,y_i), \tag{12a}$$

$$v_T(x,y) = v(x_i,y_i) + (x - x_i)\frac{\partial v}{\partial x}(x_i,y_i) + (y - y_i)\frac{\partial v}{\partial y}(x_i,y_i), \tag{12b}$$

and we use these to express $u_T(x_{i+1}, y_{i+1})$ and $v_T(x_{i+1}, y_{i+1})$:

$$u_T(x_{i+1},y_{i+1}) = u(x_i,y_i) + (x_{i+1} - x_i)\frac{\partial u}{\partial x}(x_i,y_i) + (y_{i+1} - y_i)\frac{\partial u}{\partial y}(x_i,y_i), \tag{13a}$$

$$v_T(x_{i+1},y_{i+1}) = v(x_i,y_i) + (x_{i+1} - x_i)\frac{\partial v}{\partial x}(x_i,y_i) + (y_{i+1} - y_i)\frac{\partial v}{\partial y}(x_i,y_i), \tag{13b}$$

here rewritten using a more compact notation:

$$u_{T,i+1} = u_i + (x_{i+1} - x_i)\left.\frac{\partial u}{\partial x}\right|_i + (y_{i+1} - y_i)\left.\frac{\partial u}{\partial y}\right|_i, \tag{14a}$$

$$v_{T,i+1} = v_i + (x_{i+1} - x_i)\left.\frac{\partial v}{\partial x}\right|_i + (y_{i+1} - y_i)\left.\frac{\partial v}{\partial y}\right|_i, \tag{14b}$$

where the vertical bar on the right of the derivatives means "evaluated at", and i stands for (x_i, y_i). We know that, as it is specific to the Newton-Raphson method, we want (x_{i+1}, y_{i+1}) to be a point where $u_{T,i+1} = 0$ and $v_{T,i+1} = 0$. Rearranging Eq. (14), with $u_{T,i+1} = 0$ and $v_{T,i+1} = 0$, we obtain:

$$x_{i+1}\left.\frac{\partial u}{\partial x}\right|_i + y_{i+1}\left.\frac{\partial u}{\partial y}\right|_i = -u_i + x_i\left.\frac{\partial u}{\partial x}\right|_i + y_i\left.\frac{\partial u}{\partial y}\right|_i, \tag{15a}$$

$$x_{i+1}\left.\frac{\partial v}{\partial x}\right|_i + y_{i+1}\left.\frac{\partial v}{\partial y}\right|_i = -v_i + x_i\left.\frac{\partial v}{\partial x}\right|_i + y_i\left.\frac{\partial v}{\partial y}\right|_i, \tag{15b}$$

which represents a linear system with two equations and two unknowns, x_{i+1} and y_{i+1}, whereas all the other terms are known. This can be rewritten in matricial form as:

$$\begin{bmatrix} \left.\frac{\partial u}{\partial x}\right|_i & \left.\frac{\partial u}{\partial y}\right|_i \\ \left.\frac{\partial v}{\partial x}\right|_i & \left.\frac{\partial v}{\partial y}\right|_i \end{bmatrix} \cdot \begin{bmatrix} x_{i+1} \\ y_{i+1} \end{bmatrix} = - \begin{bmatrix} u_i \\ v_i \end{bmatrix} + \begin{bmatrix} \left.\frac{\partial u}{\partial x}\right|_i & \left.\frac{\partial u}{\partial y}\right|_i \\ \left.\frac{\partial v}{\partial x}\right|_i & \left.\frac{\partial v}{\partial y}\right|_i \end{bmatrix}, \cdot \begin{bmatrix} x_i \\ y_i \end{bmatrix},$$

where:

$$\mathbf{J_i} = \begin{bmatrix} \left.\frac{\partial u}{\partial x}\right|_i & \left.\frac{\partial u}{\partial y}\right|_i \\ \left.\frac{\partial v}{\partial x}\right|_i & \left.\frac{\partial v}{\partial y}\right|_i \end{bmatrix},$$

is called *Jacobian matrix*, that is, a matrix containing all the first-order partial derivatives of a vector-valued function in several variables, and:

$$\mathbf{F_i} = \begin{bmatrix} u_i \\ v_i \end{bmatrix},$$

is the vector containing the known function evaluations at (x_i, y_i). In matricial notation, the linear system of two equations Eq. (15) can be written as:

$$\mathbf{J_i} \cdot \mathbf{x_{i+1}} = -\mathbf{F_i} + \mathbf{J_i} \cdot \mathbf{x_i}, \tag{16}$$

where $\mathbf{x_{i+1}}$ is a vector including the new guess values, which are unknown, and $\mathbf{x_i}$ is a vector including the previous guess values. Equation (16) is the iteration equation of the Newton-Raphson method in more than one dimension, which can be used to find the new guess values $\mathbf{x_{i+1}}$. Note that Eq. (16) represents a linear system of equations, just as those that we were used to see in the previous lectures as $\mathbf{A} \cdot \mathbf{x} = \mathbf{B}$; now, the matrix of the coefficients is replaced by the Jacobian matrix $\mathbf{J}$, and the vector of known terms $\mathbf{B}$ is replaced by the expression $-\mathbf{F_i} + \mathbf{J_i} \cdot \mathbf{x_i}$, which is known. Therefore, Eq. (16) can be solved with any of the direct or iterative methods outlined in Lecture 2. Note that, unlike Lecture 2, the solution of the linear system Eq. (16) does not provide the final, converged solution for $\mathbf{x}$ satisfying the initial system of nonlinear equations, but it provides only the new guess vector $\mathbf{x_{i+1}}$ to carry on the iterative procedure towards a converged solution. Therefore, the linear system Eq. (16) must be solved many times during the iterative procedure for the solution, and each time $\mathbf{J_i}$, $\mathbf{F_i}$ and $\mathbf{x_i}$ must be updated according to the new guess values. This will become more clear after the second tutorial exercise.

In the present case of two equations and two variables, we can express the iteration equation via the Cramer's rule or by employing algebraic manipulations of Eq. (15):

$$x_{i+1} = x_i - \frac{u_i \left.\frac{\partial v}{\partial y}\right|_i - v_i \left.\frac{\partial u}{\partial y}\right|_i}{det(\mathbf{J_i})}, \qquad y_{i+1} = y_i - \frac{v_i \left.\frac{\partial u}{\partial x}\right|_i - u_i \left.\frac{\partial v}{\partial x}\right|_i}{det(\mathbf{J_i})}, \tag{17}$$

with:

$$det(\mathbf{J_i}) = \left.\frac{\partial u}{\partial x}\right|_i \cdot \left.\frac{\partial v}{\partial y}\right|_i - \left.\frac{\partial u}{\partial y}\right|_i \cdot \left.\frac{\partial v}{\partial x}\right|_i. \tag{18}$$

Example. Consider the system of two nonlinear equations:

$$u(x, y) = x^2 + xy - 10 = 0, \qquad v(x, y) = y + 3xy^2 - 57 = 0, \tag{19}$$

which has exact solution $x = 2$ and $y = 3$. From the start guesses $x_0 = 1.5$ and $y_0 = 3.5$, use the Newton-Raphson method to calculate the next guesses.

We start off with evaluating the elements of the Jacobian:

$$\left.\frac{\partial u}{\partial x}\right|_0 = 2x_0 + y_0 = 6.5, \tag{20a}$$

$$\left.\frac{\partial u}{\partial y}\right|_0 = x_0 = 1.5, \tag{20b}$$

$$\left.\frac{\partial v}{\partial x}\right|_0 = 3y_0^2 = 36.75, \tag{20c}$$

$$\left.\frac{\partial v}{\partial y}\right|_0 = 1 + 6x_0y_0 = 32.5, \tag{20d}$$

and its determinant:

$$det(\mathbf{J_0}) = \left.\frac{\partial u}{\partial x}\right|_0 \cdot \left.\frac{\partial v}{\partial y}\right|_0 - \left.\frac{\partial u}{\partial y}\right|_0 \cdot \left.\frac{\partial v}{\partial x}\right|_0 = 156.125. \tag{21}$$

Next, we calculate the values of the functions at the initial guesses:

$$u(x_0, y_0) = x_0^2 + x_0y_0 - 10 = -2.5, \qquad v(x_0, y_0) = y_0 + 3x_0y_0^2 - 57 = 1.625. \tag{22}$$

We have now all the ingredients to apply the iteration equation Eq. (17):

$$x_1 = x_0 - \frac{u_0 \left.\frac{\partial v}{\partial y}\right|_0 - v_0 \left.\frac{\partial u}{\partial y}\right|_0}{det(\mathbf{J_0})} = 2.03603, \qquad y_1 = y_0 - \frac{v_0 \left.\frac{\partial u}{\partial x}\right|_0 - u_0 \left.\frac{\partial v}{\partial x}\right|_0}{det(\mathbf{J_0})} = 2.84388, \tag{23}$$

which is our first step towards a converged solution. With two equations, the error can be calculated as $e_i = |u_i| + |v_i|$.

The second tutorial at the end of this document shows how to implement in Matlab the code to solve this exercise till convergence.

Further reading: Chapra and Canale [1], Sec. 6.6.2.

4 The Newton-Raphson method for a system of nonlinear equations

We are now ready to extend the Newton-Raphson method to the case of a system of n nonlinear equations:

$$\begin{aligned} f_1(x_1, x_2, \ldots, x_n) &= 0, \\ f_2(x_1, x_2, \ldots, x_n) &= 0, \\ &\vdots \\ f_n(x_1, x_2, \ldots, x_n) &= 0, \end{aligned}$$

The generalisation of the method to n dimensions has already been seen in the previous section, Eq. (16). The new guess vector $\mathbf{x_{i+1}}$ is found by solving the following linear system:

$$\mathbf{J_i} \cdot \mathbf{x_{i+1}} = -\mathbf{F_i} + \mathbf{J_i} \cdot \mathbf{x_i}, \tag{24}$$

written here in extended notation:

$$\begin{bmatrix} \left.\frac{\partial f_1}{\partial x_1}\right|_i & \left.\frac{\partial f_1}{\partial x_2}\right|_i & \cdots & \left.\frac{\partial f_1}{\partial x_n}\right|_i \\ \left.\frac{\partial f_2}{\partial x_1}\right|_i & \left.\frac{\partial f_2}{\partial x_2}\right|_i & \cdots & \left.\frac{\partial f_2}{\partial x_n}\right|_i \\ \vdots & \vdots & \vdots & \vdots \\ \left.\frac{\partial f_n}{\partial x_1}\right|_i & \left.\frac{\partial f_n}{\partial x_2}\right|_i & \cdots & \left.\frac{\partial f_n}{\partial x_n}\right|_i \end{bmatrix} \cdot \begin{bmatrix} x_{1,i+1} \\ x_{2,i+1} \\ \vdots \\ x_{n,i+1} \end{bmatrix} = - \begin{bmatrix} f_{1,i} \\ f_{2,i} \\ \vdots \\ f_{n,i} \end{bmatrix} + \begin{bmatrix} \left.\frac{\partial f_1}{\partial x_1}\right|_i & \left.\frac{\partial f_1}{\partial x_2}\right|_i & \cdots & \left.\frac{\partial f_1}{\partial x_n}\right|_i \\ \left.\frac{\partial f_2}{\partial x_1}\right|_i & \left.\frac{\partial f_2}{\partial x_2}\right|_i & \cdots & \left.\frac{\partial f_2}{\partial x_n}\right|_i \\ \vdots & \vdots & \vdots & \vdots \\ \left.\frac{\partial f_n}{\partial x_1}\right|_i & \left.\frac{\partial f_n}{\partial x_2}\right|_i & \cdots & \left.\frac{\partial f_n}{\partial x_n}\right|_i \end{bmatrix} \cdot \begin{bmatrix} x_{1,i} \\ x_{2,i} \\ \vdots \\ x_{n,i} \end{bmatrix} .$$

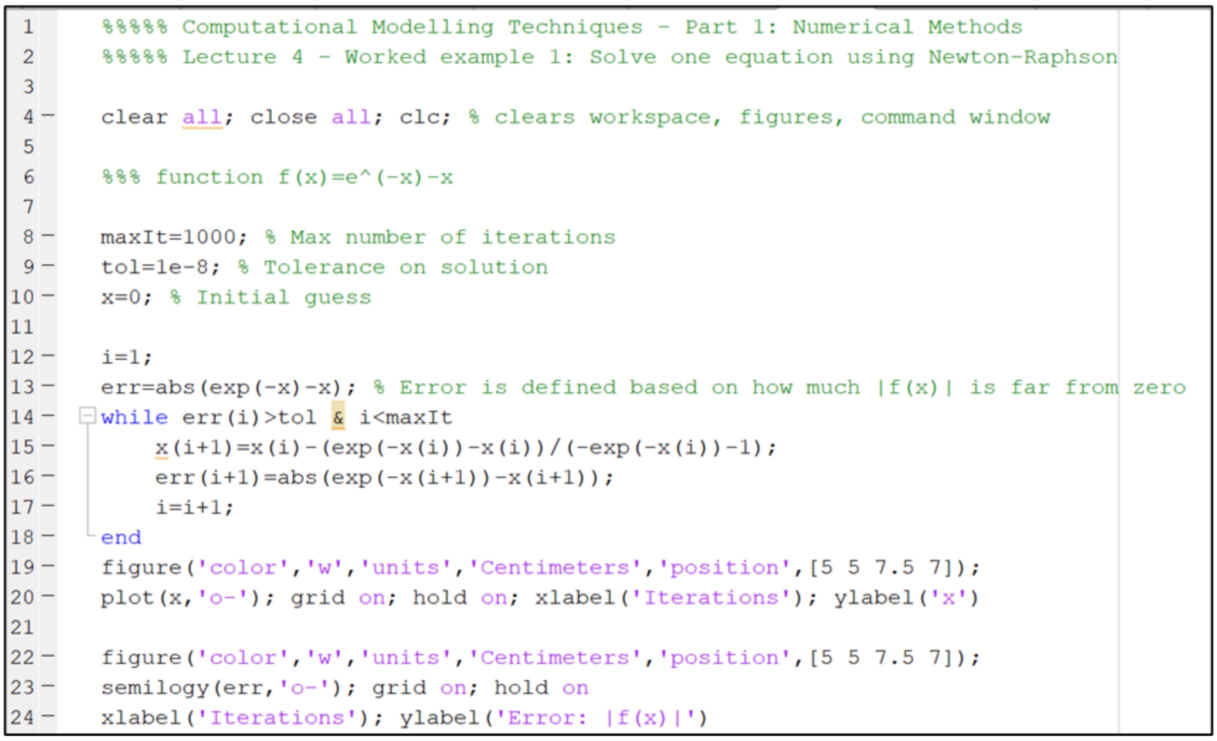
The linear system above must be solved at each $i-$th iteration, in order to obtain the new guess values at $\mathbf{x_{i+1}}$. In summary, the solution of a nonlinear system of equations using the Newton-Raphson method, is turned into an iterative procedure where, at each iteration, we must solve a linear system of equations to update the solution guesses, till convergence is achieved. Convergence is achieved when all the $f_1, f_2, \ldots, f_n$ functions are zero, or below a set tolerance, which can be expressed as the condition that the error $e_i = |\mathbf{F_i}| < tol$. **Further reading:** Chapra and Canale [1], Sec. 6.6.

5 Matlab tutorials: Newton-Raphson method

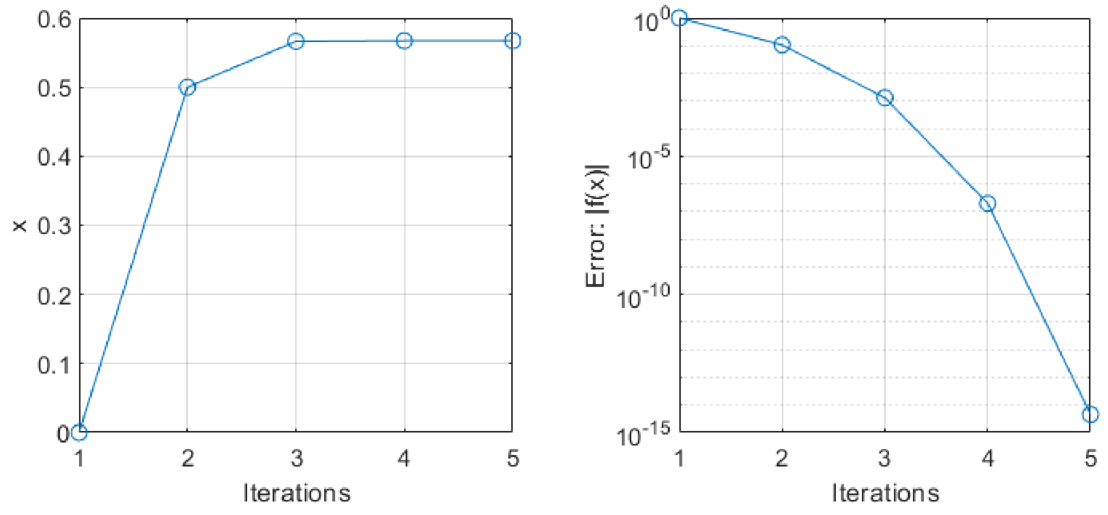
5.1 Worked example 1: Newton-Raphson method to solve one equation

Implement the Newton-Raphson method in Matlab to solve the example in Sec. 2.1, starting with the initial guess $x_0 = 0$, till convergence. For convergence, consider the error evaluated at each iteration as $e_i = |f(x_i)|$, and take 10^{-8} as the tolerance.

The code is fairly simple, it is reported below without any further comments other than those in the matlab code. The output of the code is displayed in Fig. 6.



```
%%%%% Computational Modelling Techniques - Part 1: Numerical Methods
%%%%% Lecture 4 - Worked example 1: Solve one equation using Newton-Raphson

clear all; close all; clc; % clears workspace, figures, command window

%%% function f(x)=e^(-x)-x

maxIt=1000; % Max number of iterations
tol=1e-8; % Tolerance on solution
x=0; % Initial guess

i=1;
err=abs(exp(-x)-x); % Error is defined based on how much |f(x)| is far from zero
while err(i)>tol & i<maxIt
    x(i+1)=x(i)-(exp(-x(i))-x(i))/(-exp(-x(i))-1);
    err(i+1)=abs(exp(-x(i+1))-x(i+1));
    i=i+1;
end
figure('color','w','units','Centimeters','position',[5 5 7.5 7]);
plot(x,'o-'); grid on; hold on; xlabel('Iterations'); ylabel('x')

figure('color','w','units','Centimeters','position',[5 5 7.5 7]);
semilogy(err,'o-'); grid on; hold on
xlabel('Iterations'); ylabel('Error: |f(x)|')
```

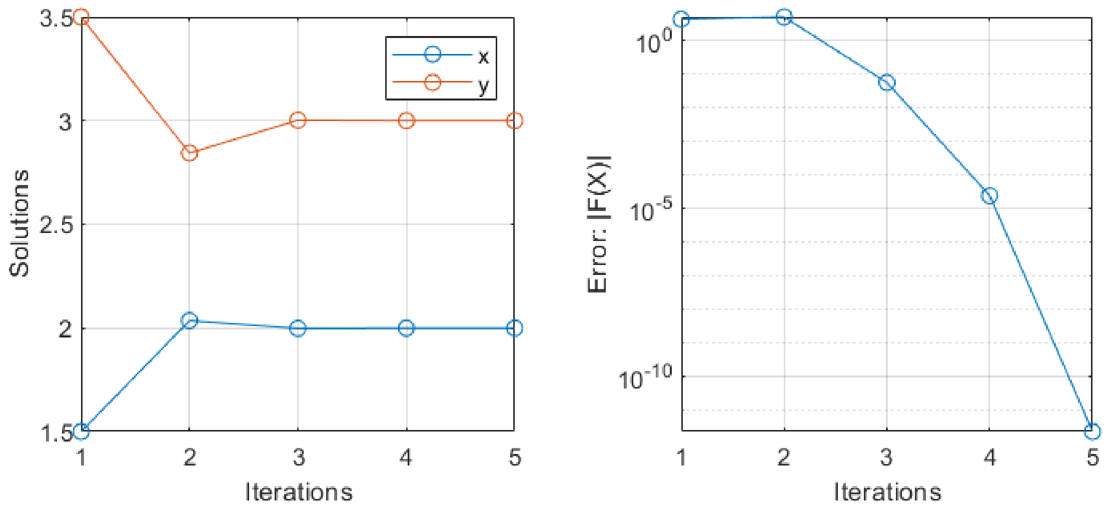


(a) (b)

Figure 6: (a) Convergence of the solution for initial guess $x_0 = 0$, and (b) convergence of the error, calculated as $|f(x_i)| = |e^{-x_i} - x_i|$. The iterative procedure is arrested when the error falls below the tolerance, set to $tol = 10^{-8}$.

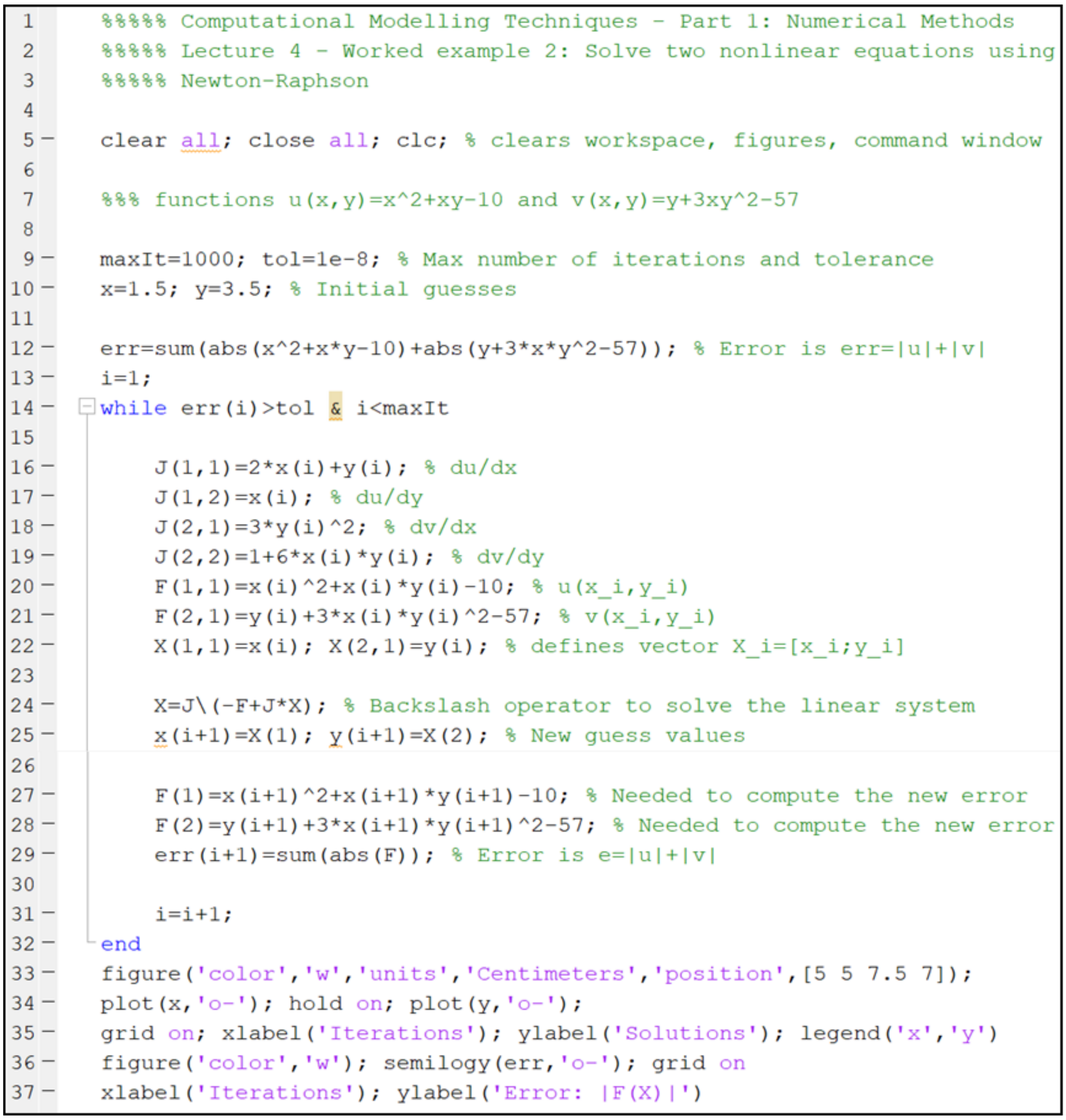
5.2 Worked example 2: Newton-Raphson method to solve a system of two non-linear equations

Implement the Newton-Raphson method in Matlab to solve the example in Sec. 3, starting with initial guesses $x_0 = 1.5$ and $y_0 = 3.5$, till convergence. For convergence, consider the error evaluated at each iteration as $e_i = |\mathbf{F_i}|$, and take 10^{-8} as the tolerance. The code is reported in the next page, without any further comments other than those in the matlab code. The output of the code is in Fig. 7.



(a) (b)

Figure 7: (a) Convergence of the solution for initial guesses $x_0 = 1.5$ and $y_0 = 3.5$, and (b) convergence of the error, calculated as $e_i = |\mathbf{F_i}| = |u_i| + |v_i|$. The iterative procedure is arrested when the error falls below the tolerance, set to $tol = 10^{-8}$.



```
%%%%% Computational Modelling Techniques - Part 1: Numerical Methods
%%%%% Lecture 4 - Worked example 2: Solve two nonlinear equations using
%%%%% Newton-Raphson

clear all; close all; clc; % clears workspace, figures, command window

%%% functions u(x,y)=x^2+xy-10 and v(x,y)=y+3xy^2-57

maxIt=1000; tol=1e-8; % Max number of iterations and tolerance
x=1.5; y=3.5; % Initial guesses

err=sum(abs(x^2+x*y-10)+abs(y+3*x*y^2-57)); % Error is err=|u|+|v|
i=1;
while err(i)>tol & i<maxIt

    J(1,1)=2*x(i)+y(i); % du/dx
    J(1,2)=x(i); % du/dy
    J(2,1)=3*y(i)^2; % dv/dx
    J(2,2)=1+6*x(i)*y(i); % dv/dy
    F(1,1)=x(i)^2+x(i)*y(i)-10; % u(x_i,y_i)
    F(2,1)=y(i)+3*x(i)*y(i)^2-57; % v(x_i,y_i)
    X(1,1)=x(i); X(2,1)=y(i); % defines vector X_i=[x_i;y_i]

    X=J\(-F+J*X); % Backslash operator to solve the linear system
    x(i+1)=X(1); y(i+1)=X(2); % New guess values

    F(1)=x(i+1)^2+x(i+1)*y(i+1)-10; % Needed to compute the new error
    F(2)=y(i+1)+3*x(i+1)*y(i+1)^2-57; % Needed to compute the new error
    err(i+1)=sum(abs(F)); % Error is e=|u|+|v|

    i=i+1;
end
figure('color','w','units','Centimeters','position',[5 5 7.5 7]);
plot(x,'o-'); hold on; plot(y,'o-');
grid on; xlabel('Iterations'); ylabel('Solutions'); legend('x','y')
figure('color','w'); semilogy(err,'o-'); grid on
xlabel('Iterations'); ylabel('Error: |F(X)|')
```

5.3 Suggested exercises

1. Let's extend the previous worked example to solve a system of three nonlinear equations:

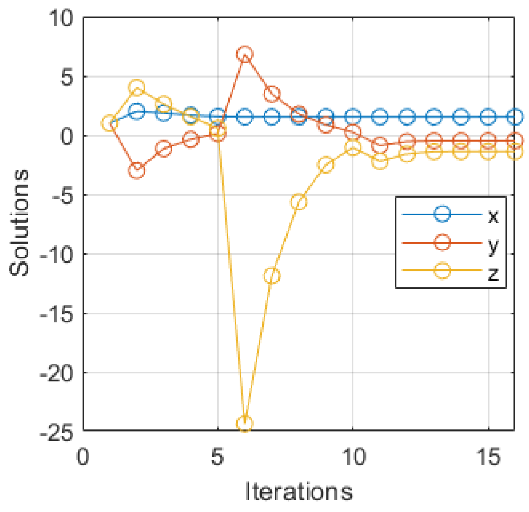
$$x^2 + yz = 3, \tag{25a}$$

$$2xy + y + z^2 = 0, \tag{25b}$$

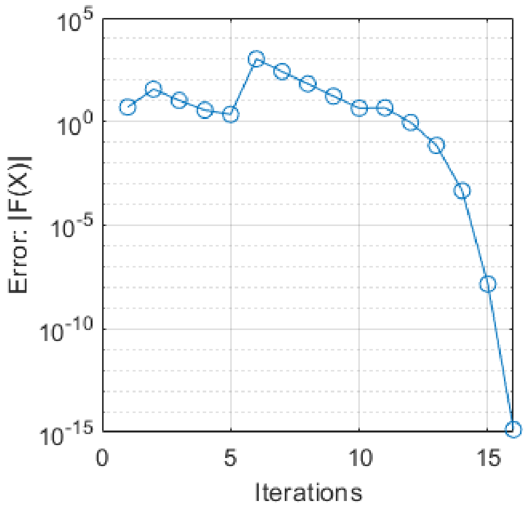
$$xyz = 1. \tag{25c}$$

   Select appropriate starting guess and other parameters for the solution procedure. For the procedure, refer to Sec. 4. An example of solution is shown in Fig. 8. After 16 iterations, I obtain: $x = 1.5321$, $y = -0.4715$ and $z = -1.3843$.

2. Let's now show that the Newton-Raphson method can be used also to solve more applied problems. Let's consider a one-dimensional steady-state heat conduction problem such as those



(a)



(b)

Figure 8: Suggested exercise 1. (a) Convergence of the solution when setting all initial guess to 1, and (b) convergence of the error. The iterative procedure is arrested when the error falls below the tolerance, set to $tol = 10^{-8}$.

encountered in Lecture 1, but this time we consider a thermal conductivity that varies with temperature in linear way: $\lambda(T) = \lambda_0(1 + \gamma T)$, with T measured in Kelvin degrees. All metals have a significant variation of λ with T. For example, copper has $\lambda_0 = 500\,\mathrm{W/(m\,K)}$ with $\gamma = -0.00085\,1/\mathrm{K}$, which means that every 10 K the thermal conductivity decreases by about 0.85%. Neglecting the source term, the ODE governing this problem is the usual one:

$$\frac{d}{dx}\left(\lambda(T)\frac{dT}{dx}\right) = 0. \tag{26}$$

This equation is now non-linear, because the derivative of T multiplies a function of T. The finite-volume integration can still be applied, but will not eventually lead to a system of linear equations. However, the resulting system of nonlinear equations can be eventually solved using the Newton-Raphson method.

We begin with integrating the equation above using the finite-volume method for an internal control volume, and the same procedure as Lecture 1 leads to:

$$-\frac{\lambda_w}{\delta x_w}T_W + \left(\frac{\lambda_w}{\delta x_w} + \frac{\lambda_e}{\delta x_e}\right)T_P - \frac{\lambda_e}{\delta x_e}T_E = 0. \tag{27}$$

Now we need to be careful, because λ_e and λ_w are both a function of the temperature:

$$\lambda_e = \lambda_0(1 + \gamma T_e) = \lambda_0\left(1 + \gamma\frac{T_E + T_P}{2}\right), \tag{28}$$

and:

$$\lambda_w = \lambda_0(1 + \gamma T_w) = \lambda_0\left(1 + \gamma\frac{T_P + T_W}{2}\right), \tag{29}$$

where the temperature at the control volume faces is evaluated as average of the temperatures at the neighbour centres; if faces are mid-way between the centres, as we always assumed in Lecture

1, the operation is correct. Substituting the expressions above in Eq. (27) and rearranging, we obtain a nonlinear algebraic equation:

$$-(a+bT_W)T_W + 2(a+bT_P)T_P - (a+bT_E)T_E = 0, \qquad a = \frac{\lambda_0}{\delta x}, \quad b = \frac{\gamma\lambda_0}{2\delta x}, \tag{30}$$

where now we have assumed that the nodes of the grid are all equidistant, with distance δx. The equation above is now a nonlinear algebraic equation, which is valid for every internal node.

Let's now consider that we want to solve the problem with Dirichlet boundary conditions $T(x = 0) = T_a$ and $T(x = L) = T_b$, and 5 control volumes, 3 of which internal. Our unknowns are the temperatures at the control volume centres, T_1, T_2, T_3, T_4, T_5. Using the Dirichlet boundary conditions for T_1 and T_5, and the discretisation Eq. (30) for T_2, T_3, T_4, we have the system of 5 nonlinear equations to solve:

$$T_1 = T_a \rightarrow T_1 - T_a = 0, \tag{31a}$$

$$-(a+bT_1)T_1 + 2(a+bT_2)T_2 - (a+bT_3)T_3 = 0, \tag{31b}$$

$$-(a+bT_2)T_2 + 2(a+bT_3)T_3 - (a+bT_4)T_4 = 0, \tag{31c}$$

$$-(a+bT_3)T_3 + 2(a+bT_4)T_4 - (a+bT_5)T_5 = 0, \tag{31d}$$

$$T_5 = T_b \rightarrow T_5 - T_b = 0. \tag{31e}$$

This system is equivalent to the system of 3 equations seen in suggested exercise 1, and can be solved iteratively using the Newton-Raphson method, following the procedure outlined in Sec. 4. The iteration equation will be: $\mathbf{J_i} \cdot \mathbf{x_{i+1}} = -\mathbf{F_i} + \mathbf{J_i} \cdot \mathbf{x_i}$, where $\mathbf{x_i} = (T_1, T_2, T_3, T_4, T_5)_i$, i.e. the vector of the 5 unknowns at iteration i. At each iteration, the Jacobian of the system is:

$$\mathbf{J_i} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & \\ -a-2bT_1 & 2a+4bT_2 & -a-2bT_3 & 0 & 0 & \\ 0 & -a-2bT_2 & 2a+4bT_3 & -a-2bT_4 & 0 & \\ 0 & 0 & -a-2bT_3 & 2a+4bT_4 & -a-2bT_5 & 0 \\ & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

where all the values of the temperature refer to iteration i and thus must be recalculated at each iteration, as done in Worked example 2. The vector of known function evaluations at iteration i will be given by the system of equations above:

$$\mathbf{F_i} = \begin{pmatrix} T_1 - T_a \\ -(a+bT_1)T_1 + 2(a+bT_2)T_2 - (a+bT_3)T_3 \\ -(a+bT_2)T_2 + 2(a+bT_3)T_3 - (a+bT_4)T_4 \\ -(a+bT_3)T_3 + 2(a+bT_4)T_4 - (a+bT_5)T_5 \\ T_5 - T_b \end{pmatrix}$$

where all the values of the temperature refer to iteration i and thus must be recalculated at each iteration. The problem can then be solve iteratively and, once converged, we will obtain the values of temperature at the nodes. We have thus solved a system of nonlinear equations

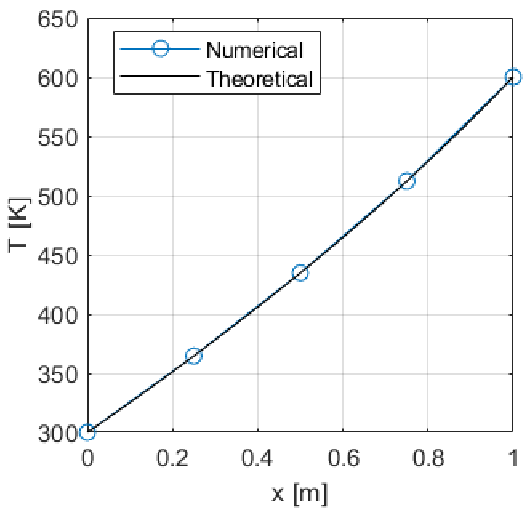


Figure 9: Suggested exercise 4. Numerical and theoretical temperature field.

representing a heat conduction problem with temperature-varying thermal conductivity.

You can try solving this exercise with the following parameters: $T_a = 300\,\mathrm{K}$, $T_b = 600\,\mathrm{K}$, $L = 1\,\mathrm{m}$, $\lambda_0 = 500\,\mathrm{W/(m\,K)}$, $\gamma = -0.00085\,\mathrm{1/K}$, and $n = 5$ nodes. The algorithm will be the same as that for Worked example 2 or suggested exercise 1, being careful in redefining all the vectors and matrices. Figure 9 shows a plot of the expected solution, obtained using an initial guess vector $\mathbf{x_0} = (T_1, T_2, T_3, T_4, T_5)_0 = (T_a, T_a, T_a, T_a, T_a)$ and a tolerance of 10^{-12}; the solution converges after 21 iterations. This problem has the following analytical solution:

$$T(x) = -\frac{1}{\gamma} - \sqrt{\frac{1}{\gamma^2} - \frac{2\lambda_m}{\gamma\lambda_0}\frac{x}{L}(T_a - T_b) + T_a^2 + \frac{2}{\gamma}T_a}, \qquad \lambda_m = \lambda_0\left(1 + \gamma\frac{T_a + T_b}{2}\right), \tag{32}$$

which is the black curve in Fig. 9. Note that, if $\gamma > 0$, the minus sign before the square root in the analytical solution becomes a plus sign. You can see that the temperature profile in Fig. 9 is no longer linear, owing to the thermal conductivity not being a constant.

References

[1] S. C. Chapra and R. P. Canale. *Numerical Methods for Engineers, 7th edition.* McGraw-Hill Education, New York, USA, 2015. NUsearch; Download(may not work).