University of Nottingham
UK | CHINA | MALAYSIA

Computer Modelling Techniques

Numerical Methods
Lecture 3: Roots of equations

Mirco Magnini

Recap

- A differential equation such as: $$\frac{d}{dx}\left(\lambda\frac{dT}{dx}\right)+S(x)=0$$ is linear in the temperature if $\lambda \neq \lambda(T)$. In this case, the linear ODE can be discretised with any method and this leads to a linear algebraic equation, $a_W T_W + a_P T_P + a_E T_E = b$, where the coefficients a are independent of the temperature (L1). Assembling the *n* eqs for *n* control volumes yields a system of linear eqs, that can be solved with any direct/iterative method (L2).
- However, the same equation may become nonlinear if $\lambda = \lambda(T)$; in this case, the discretisation equation is nonlinear because the coefficients a depend on the temperature, and assembling the eqs leads to a system of nonlinear equations. This can still be solved (i) with iterations or (ii) with the methods that we see today.

Today's menu

- Newton-Raphson method to find roots of one equation
- Newton-Raphson method for a system of two nonlinear equations
- Newton-Raphson method for a system of *n* nonlinear equations
- Worked examples – Newton-Raphson method in Matlab

Expected outcome: know the principles of the N-R method, advantages and pitfalls; be able to implement it using matlab.

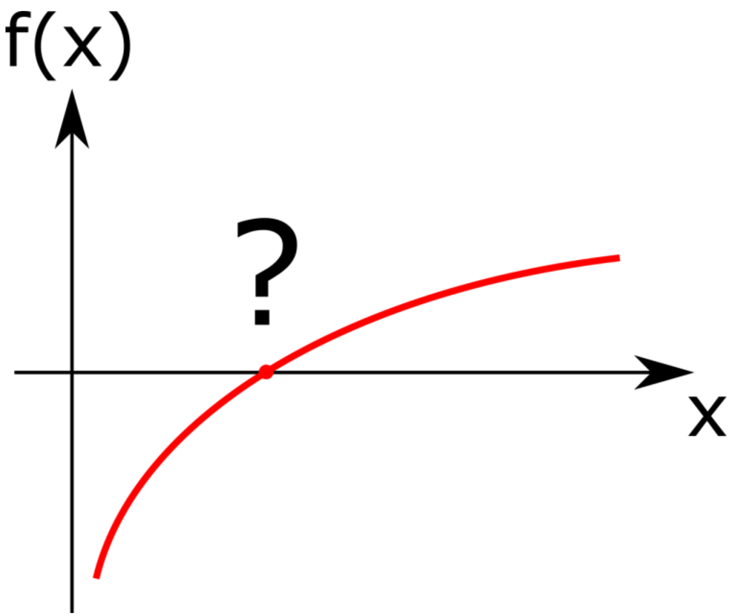
Solution of nonlinear equations

In general, equations in mathematics can be recast in the form $f(x) = 0$.

Example: $x^4 = 5 \implies x^4 - 5 = 0$,

or: $e^{-x} = x \implies e^{-x} - x = 0$

Therefore, solving the equation means finding the root(s) of the equation.



Several methods are available:

- Bisection (bracketing method)
- Newton-Raphson (open method)
- Secant (open method)
- Brent method (bracketing+open)
- …and so on

To be clear:

- **Bracketing methods**: the search interval is “bracketed” between two values where the function changes sign, and this interval is narrowed down iteratively.
- **Open methods**: use only one starting guess, no bracketing is done.

Newton-Raphson method

The N-R method works by using an initial guess, and then successively improves it by using iterations based on the slope (gradient) of the curve.

From the previous guess x_i, we want to find the next guess x_{i+1}. First, we find the tangent to $f(x)$ in x_i. Then we extend the tangent line till it crosses the x-axis, and set x_{i+1} as the abscissa of the zero crossing. How do we "translate" this into an iteration equation?

First-order Taylor series expansion nearby x_i:

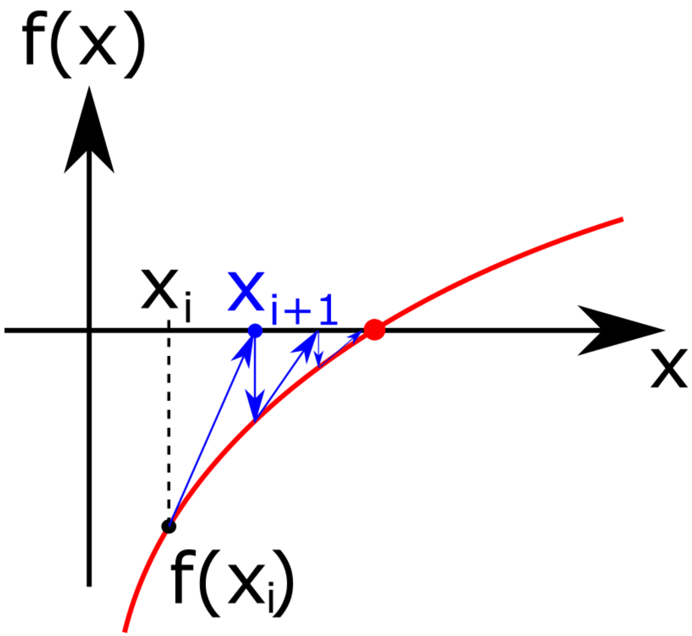
$$f_T(x) = f(x_i) + f'(x_i)(x - x_i)$$

This crosses the x-axis at some point where $f_T(x) = 0$, which identifies x_{i+1}:

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

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

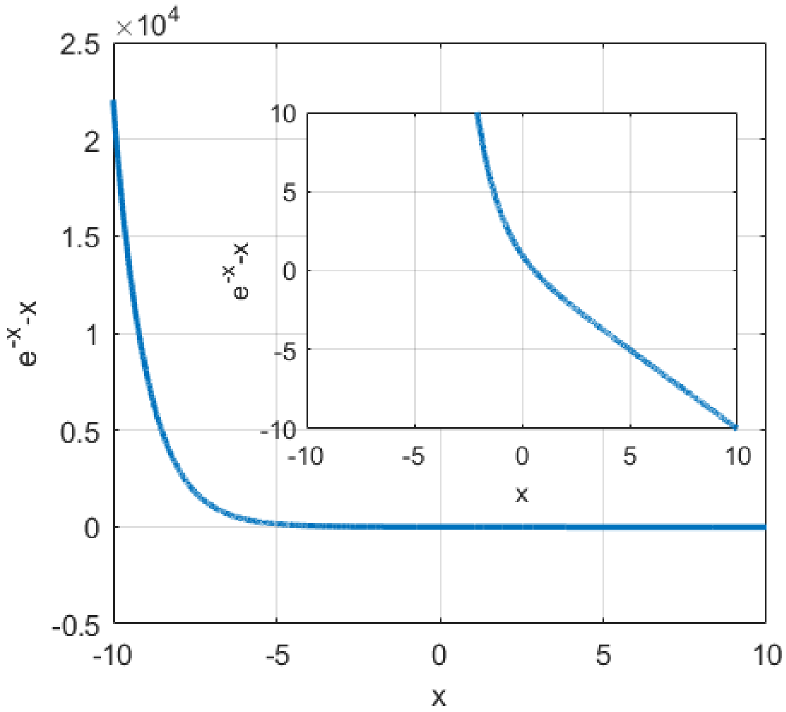
Iterative procedure for new guess



Newton-Raphson method

Example: consider the equation

$$f(x) = e^{-x} - x = 0$$



The derivative of $f(x)$ is:

$$f'(x) = -e^{-x} - 1$$

Therefore the equation to use to find a new guess is:

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)} = x_i - \frac{e^{-x_i} - x_i}{-e^{-x_i} - 1}$$

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

$$x_{i+1} = 0 - \frac{1-0}{-1-1} = 0.5$$

If we continue, we will eventually converge to the solution, which is $x \approx 0.56714$

Newton-Raphson method

$f(x) = e^{-x} - x = 0$

Route to convergence and impact of a different initial guess:

$tol\text{: } |e^{-x_i} - x_i| < 1e-08$

$x_0 = 0$

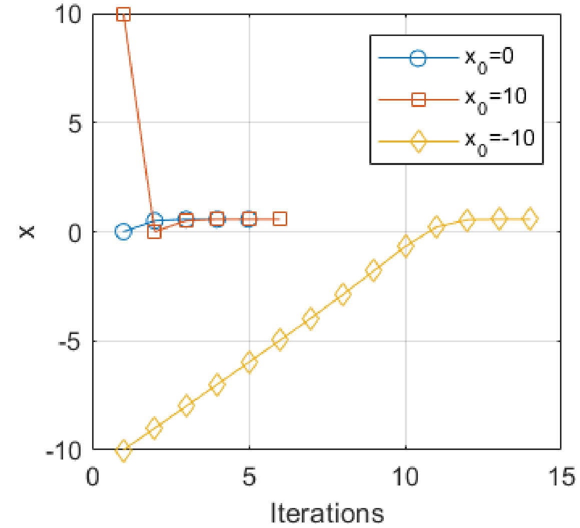
```
>> format long
>> x'

ans =

                 0
   0.500000000000000
   0.566311003197218
   0.567143165034862
   0.567143290409781
```

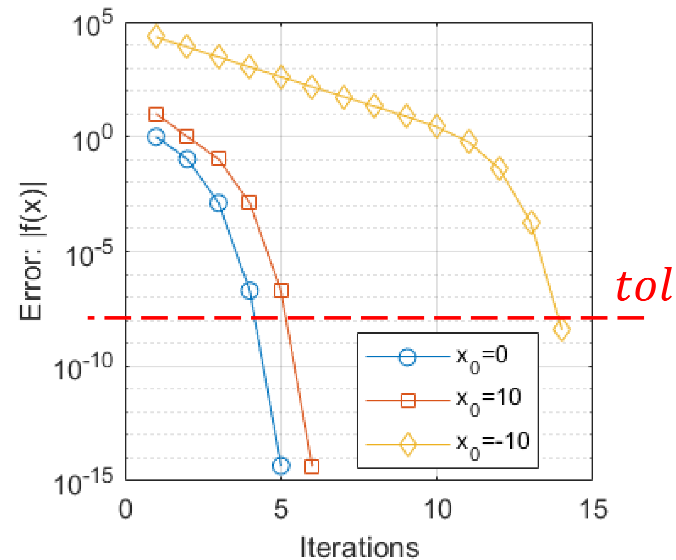
$x_0 = 10$

```
>> 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
```





Newton-Raphson method - Pitfalls

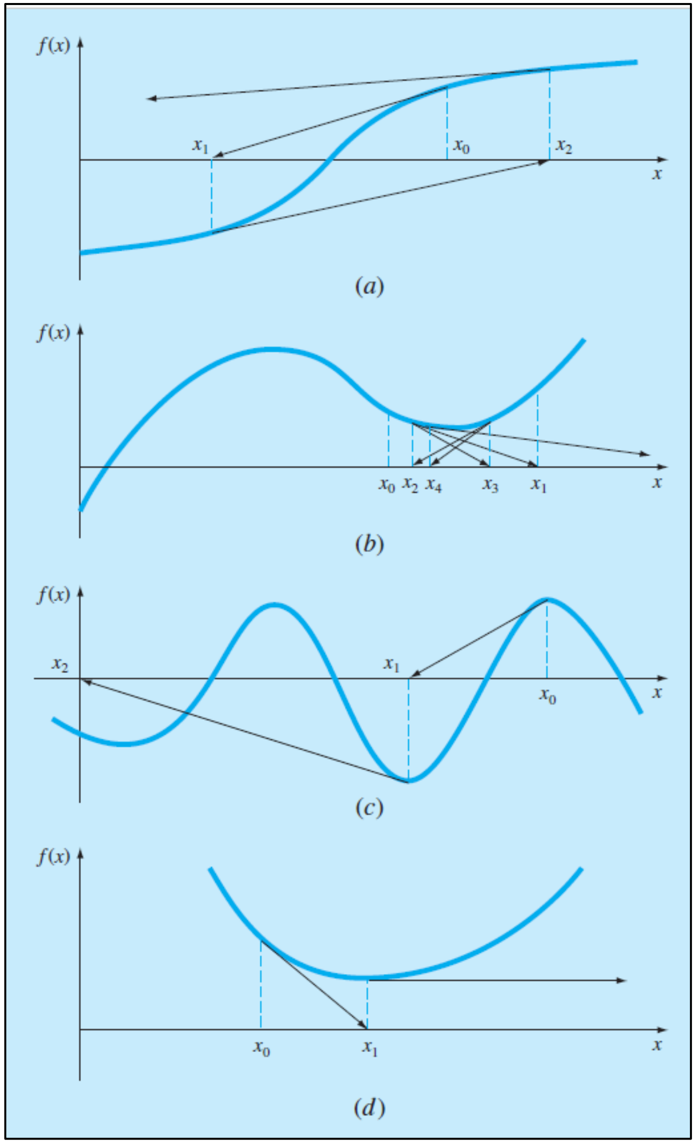


Figure from: Numerical Methods for Engineers, S. C. Chapra, R. P. Canale, McGraw-Hill 2015

- The initial guess has to be ”sufficiently” close to the root
- Convergence depends on the nature of the function, in particular its derivative (see figure):
  - a) nearby an inflection point $(f'' = 0)$, iterations diverge
  - b) nearby a max/min, iterations oscillate or diverge (d)
  - c) nearby max/min, the guess jumps to another root
- No bracketing is done, divergence may occur
- Convergence is not guaranteed
- Needs knowledge of the first derivative

Remedies:

- Always set a max number of iterations
- Check that the solution is converging, $|f(x)| \longrightarrow 0$
- Alert if the guess shoots out
- Secant method - derivative calculated with two guesses
- Brent method - first bisection, then open methods; try:

```
fzero(@(x) exp(-x)-x,0,optimset('DISP','ITER'))
```

Newton-Raphson method for two nonlinear eqs.

Suppose we have to solve the system of equations: $u(x,y) = 0, \quad v(x,y) = 0.$

Newton-Raphson in 2 dimensions: first-order Taylor series expansion,

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

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

$$u_{T,i+1} = 0 \implies \left.\frac{\partial u}{\partial x}\right|_i x_{i+1} + \left.\frac{\partial u}{\partial y}\right|_i y_{i+1} = -u_i + x_i \left.\frac{\partial u}{\partial x}\right|_i + y_i \left.\frac{\partial u}{\partial y}\right|_i$$

$$v_{T,i+1} = 0 \implies \left.\frac{\partial v}{\partial x}\right|_i x_{i+1} + \left.\frac{\partial v}{\partial y}\right|_i y_{i+1} = -v_i + x_i \left.\frac{\partial v}{\partial x}\right|_i + y_i \left.\frac{\partial v}{\partial y}\right|_i$$

Jacobian

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

$$\boldsymbol{F_i} = \begin{bmatrix} u_i \\ v_i \end{bmatrix}$$

$$\boldsymbol{J_i} \cdot \boldsymbol{x_{i+1}} = -\boldsymbol{F_i} + \boldsymbol{J_i} \cdot \boldsymbol{x_i}$$

Iterative procedure for new guess

unknown

In 2 dimensions: $x_{i+1} = x_i - \dfrac{u_i \dfrac{\partial v_i}{\partial y} - v_i \dfrac{\partial u_i}{\partial y}}{det(\boldsymbol{J_i})}$ $\qquad y_{i+1} = y_i - \dfrac{v_i \dfrac{\partial u_i}{\partial x} - u_i \dfrac{\partial v_i}{\partial x}}{det(\boldsymbol{J_i})}$

Newton-Raphson method for two nonlinear eqs.

Example:

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

Initial guesses: $x_0 = 1.5, \;\; y_0 = 3.5$

We start off with evaluating the elements of the Jacobian:

$$\left.\frac{\partial u}{\partial x}\right|_0 = 2x_0 + y_0 = 6.5, \quad \left.\frac{\partial u}{\partial y}\right|_0 = x_0 = 1.5$$

$$\left.\frac{\partial v}{\partial x}\right|_0 = 3y_0^2 = 36.75, \quad \left.\frac{\partial v}{\partial y}\right|_0 = 1 + 6x_0y_0 = 32.5$$

$$\Rightarrow \quad det(\boldsymbol{J_0}) = \left.\frac{\partial u}{\partial x}\right|_0 \cdot \left.\frac{\partial v}{\partial y}\right|_0 - \left.\frac{\partial u}{\partial y}\right|_0 \cdot \left.\frac{\partial v}{\partial x}\right|_0 = = 156.125$$

Values of the functions at the initial guesses:

$$u_0 = x_0^2 + x_0y_0 - 10 = -2.5, \qquad v_0 = y_0 + 3x_0y_0^2 - 57 = 1.625$$

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

The computation is converging towards the exact solution $x = 2, y = 3$.

Newton-Raphson for systems of nonlinear eqs.

Suppose we have to solve a system with *n* unknows and *n* nonlinear equations:

$$f_1(x_1, x_2, \dots, x_n) = 0$$

$$f_2(x_1, x_2, \dots, x_n) = 0$$

$$\vdots$$

$$f_n(x_1, x_2, \dots, x_n) = 0$$

Generalisation of the **Newton-Raphson method to *n* dimensions**:

$$\boldsymbol{J_i} \cdot \boldsymbol{x_{i+1}} = -\boldsymbol{F_i} + \boldsymbol{J_i} \cdot \boldsymbol{x_i}$$

At each $i-th$ iteration, we have to solve a new linear system to obtain $\boldsymbol{x_{i+1}}$

$$\begin{pmatrix} \frac{\partial f_{1,i}}{\partial x_1} & \frac{\partial f_{1,i}}{\partial x_2} & \cdots & \frac{\partial f_{1,i}}{\partial x_n} \\ \frac{\partial f_{2,i}}{\partial x_1} & \frac{\partial f_{2,i}}{\partial x_2} & \cdots & \frac{\partial f_{2,i}}{\partial x_n} \\ \vdots & \vdots & \vdots & \ddots \\ \frac{\partial f_{n,i}}{\partial x_1} & \frac{\partial f_{n,i}}{\partial x_2} & \cdots & \frac{\partial f_{n,i}}{\partial x_n} \end{pmatrix} \times \begin{pmatrix} x_{1,i+1} \\ x_{2,i+1} \\ \vdots \\ x_{n,i+1} \end{pmatrix} = - \begin{pmatrix} f_{1,i} \\ f_{2,i} \\ \vdots \\ f_{n,i} \end{pmatrix} + \begin{pmatrix} \frac{\partial f_{1,i}}{\partial x_1} & \frac{\partial f_{1,i}}{\partial x_2} & \cdots & \frac{\partial f_{1,i}}{\partial x_n} \\ \frac{\partial f_{2,i}}{\partial x_1} & \frac{\partial f_{2,i}}{\partial x_2} & \cdots & \frac{\partial f_{2,i}}{\partial x_n} \\ \vdots & \vdots & \vdots & \ddots \\ \frac{\partial f_{n,i}}{\partial x_1} & \frac{\partial f_{n,i}}{\partial x_2} & \cdots & \frac{\partial f_{n,i}}{\partial x_n} \end{pmatrix} \times \begin{pmatrix} x_{1,i} \\ x_{2,i} \\ \vdots \\ x_{n,i} \end{pmatrix}$$

What to take home from today's lecture

- Advantages/pitfalls of the Newton-Raphson method
- How to set the iterative procedure for the solution of one, two, or a system of nonlinear equations
- How to use Matlab to implement the solution procedure

Worked example 1

Implement the Newton-Raphson method in Matlab to solve the equation:

$$f(x) = e^{-x} - x = 0$$

starting with initial guess $x_0 = 0$, till convergence. For convergence, consider the error evaluated at each iteration as $err_i = |f(x_i)|$ and use $tol = 10^{-8}$.

```
%%%%% 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)|')
```

Worked example 2

Implement the Newton-Raphson method in Matlab to solve the system of equations:

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

starting with initial guess $x_0 = 1.5, y_0 = 3.5$, till convergence. For convergence, consider the error evaluated at each iteration as $err_i = |u_i| + |v_i|$ and use $tol = 10^{-8}$.

```
%%%%% 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)|')
```