a System of n nonlinear equations with n variables.

Coding with MATLAB – Part III

$\begin{cases} f_1(x_1, \dots, x_n) = 0 \\ f_2(x_1, \dots, x_n) = 0 \\ \vdots \\ f_n(x_1, \dots, x_n) = 0 \end{cases}$

$\begin{cases} f_1 = x^2 + y^2 = 5 \checkmark \\ f_2 = 2xy - y^3 = 1 \end{cases}$ $F = \begin{pmatrix} f_1 \\ f_2 \end{pmatrix} = 0$

Newton's method for nonlinear univariate equations can be extended to a system of nonlinear equations. Consider the following system:

$J = \begin{pmatrix} 2x & 2y \\ 2y & 2x - 3y^2 \end{pmatrix}$

$$F(X) = F(x_1, x_2, \dots, x_n) = 0 \iff \begin{cases} f_1(x_1, x_2, \dots, x_n) = 0 \\ f_2(x_1, x_2, \dots, x_n) = 0 \\ \vdots \\ f_n(x_1, x_2, \dots, x_n) = 0 \end{cases}$$

$f(x) = 0$

$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$

$= x_n - (f'(x_n))^{-1} f(x_n)$

where $f_1, f_2, \dots, f_n$ are differentiable functions and their Jacobian

$J = \nabla F$

$X_{n+1} = X_n - J^{-1}(X_n) F(X_n)$

$$J = \left(\frac{\partial f_i}{\partial x_j}\right)_{n \times n}$$

is invertible in a neighborhood around the solution $X^* = (x_1^*, x_2^*, \dots, x_n^*)^T$ for the system. Then, the Newton's iterative method for solving nonlinear systems of equations starting from $X^0 = (x_1^0, x_2^0, \dots, x_n^0)^T$ would be

$$X^{k+1} = X^k - J^{-1}(X^k).F(X^k)$$

X^0

The following M-file implements Newton's method for the following nonlinear systems of equations:

∇f_1 ∇f_2

$$\begin{cases} x^2 + xy = 10 \\ y + 3xy^2 = 57 \end{cases}$$

$\checkmark \begin{cases} f_1 = x^2 + xy - 10 = 0 \checkmark \\ f_2 = y + 3xy^2 - 57 = 0 \checkmark \end{cases}$

$\checkmark$ $J = \begin{bmatrix} 2x + y & 3y^2 \\ x & 1 + 6xy \end{bmatrix}$

```
x0=[1.5,3.5]';
maxit=[]; es=[];
iter = 0;
x = x0;

while (1)
    [J,f] = func(x);
    dx = J\f;
    x = x - dx;
    iter = iter + 1;
    ea=100*max(abs(dx./x));
    if iter>= maxit || ea<= es,
        break,
    end
end
function [J, f] =func(x)
% This function defines f(x) and returns the function value
and its Jacobian at x

f=[x(1,1)^2+x(1,1)*x(2,1)-10; x(2,1)+3*x(1,1)*x(2,1)^2-57];
J=[2*x(1,1)+x(2,1) x(1,1);3*x(2,1)^2 1+6*x(1,1)*x(2,1)];
end
```

$$x = x - \underbrace{J(x)^{-1}F(x)}_{dx}$$

$$J(x)\,dx$$

Matlab `fsolve` function

The `fsolve` function solves systems of nonlinear equations with several variables. A general representation of its syntax is

```
>>[x, fx] = fsolve(function, x0, options)
```

where `[x, fx]` = a vector containing the roots `x` and a vector containing the values of the functions evaluated at the roots, `function` = the name of the function containing a vector holding the equations being solved, `x0` is a vector holding the initial guesses for the unknowns, and options is a data structure created by the `optimset` function. Note that if you desire to pass function parameters but not use the options, pass an empty vector `[]` in its place.

The `optimset` function has the syntax

```
>> options = optimset('par1',val1,'par2',val2,...)
```

where the parameter `pari` has the value `vali`. A complete listing of all the possible parameters can be obtained by merely entering `optimset` at the command prompt. The parameters commonly used with the `fsolve` function are

- `display`: When set to `'iter'` displays a detailed record of all the iterations.
- `tolx`: A positive scalar that sets a termination tolerance on `x`.
- `tolfun`: A positive scalar that sets a termination tolerance on `fx`.

For example, consider the following system of nonlinear equations:

$$\begin{cases} 2x_1 + x_1 x_2 - 10 = 0 \\ x_2 + 3x_1 x_2^2 - 57 = 0 \end{cases}$$

Try the following:

```
clc,
format compact

[x,fx] = fsolve(@fun,[1.5;3.5])

function f = fun(x)
f = [x(1)^2+x(1)*x(2)-10;x(2)+3*x(1)*x(2)^2-57];
end
```