Iterative Methods

Berlin Chen
Department of Computer Science & Information Engineering
National Taiwan Normal University

Reference:

1. *Applied Numerical Methods with MATLAB for Engineers*, Chapter 12 & Teaching material

Chapter Objectives

- Understanding the difference between the Gauss-Seidel and Jacobi methods
- Knowing how to assess diagonal dominance and knowing what it means
- Recognizing how relaxation can be used to improve convergence of iterative methods
- Understanding how to solve systems of nonlinear equations with successive substitution and Newton-Raphson

$$[A]\ \{x\} = \{b\}$$

Gauss-Seidel Method

- The ***Gauss-Seidel method*** is the most commonly used iterative method for solving linear algebraic equations $[A]\{x\}=\{b\}$
- The method solves each equation in a system for a particular variable, and then uses that value in later equations to solve later variables
- For a 3x3 system **with nonzero elements along the diagonal**, for example, the j^{th} iteration values are found from the j-1^{th} iteration using:

$$a_{11}x_1 + a_{12}x_2 + a_{13}x_3 = b_1$$
$$a_{21}x_1 + a_{22}x_2 + a_{23}x_3 = b_2$$
$$a_{31}x_1 + a_{32}x_2 + a_{33}x_3 = b_3$$

$$x_1^j = \frac{b_1 - a_{12}x_2^{j-1} - a_{13}x_3^{j-1}}{a_{11}}$$
$$x_2^j = \frac{b_2 - a_{21}x_1^{j} - a_{23}x_3^{j-1}}{a_{22}}$$
$$x_3^j = \frac{b_3 - a_{31}x_1^{j} - a_{32}x_2^{j}}{a_{33}}$$

Gauss-Seidel Method: Convergence

- The convergence of an iterative method can be calculated by determining the relative percent change of each element in {*x*}. For example, for the *i* th element in the *j* th iteration,

$$\varepsilon_{a,i} = \left| \frac{x_i^j - x_i^{j-1}}{x_i^j} \right| \times 100\%$$

- The method is ended when all elements have converged to a set tolerance

Gauss-Seidel Method: An Example (1/2)

Example 12.1

Problem Statement. Use the Gauss-Seidel method to obtain the solution for

$$\begin{aligned} 3x_1 - 0.1x_2 - 0.2x_3 &= 7.85 \\ 0.1x_1 + 7x_2 - 0.3x_3 &= -19.3 \\ 0.3x_1 - 0.2x_2 + 10x_3 &= 71.4 \end{aligned}$$

Note that the solution is $x_1 = 3$, $x_2 = -2.5$, and $x_3 = 7$.

Solution. First, solve each of the equations for its unknown on the diagonal:

$$x_1 = \frac{7.85 + 0.1x_2 + 0.2x_3}{3} \tag{E12.1.1}$$

$$x_2 = \frac{-19.3 - 0.1x_1 + 0.3x_3}{7} \tag{E12.1.2}$$

$$x_3 = \frac{71.4 - 0.3x_1 + 0.2x_2}{10} \tag{E12.1.3}$$

By assuming that x_2 and x_3 are zero, Eq. (E12.1.1) can be used to compute

$$x_1 = \frac{7.85 + 0.1(0) + 0.2(0)}{3} = 2.616667$$

This value, along with the assumed value of $x_3 = 0$, can be substituted into Eq. (E12.1.2) to calculate

$$x_2 = \frac{-19.3 - 0.1(2.616667) + 0.3(0)}{7} = -2.794524$$

The first iteration is completed by substituting the calculated values for x_1 and x_2 into Eq. (E12.1.3) to yield

$$x_3 = \frac{71.4 - 0.3(2.616667) + 0.2(-2.794524)}{10} = 7.005610$$

Gauss-Seidel Method: An Example (2/2)

Example 12.1

For the second iteration, the same process is repeated to compute

$$x_1 = \frac{7.85 + 0.1(-2.794524) + 0.2(7.005610)}{3} = 2.990557$$

$$x_2 = \frac{-19.3 - 0.1(2.990557) + 0.3(7.005610)}{7} = -2.499625$$

$$x_3 = \frac{71.4 - 0.3(2.990557) + 0.2(-2.499625)}{10} = 7.000291$$

The method is, therefore, converging on the true solution. Additional iterations could be applied to improve the answers. However, in an actual problem, we would not know the true answer *a priori*. Consequently, Eq. (12.2) provides a means to estimate the error. For example, for x_1:

$$\varepsilon_{a,1} = \left|\frac{2.990557 - 2.616667}{2.990557}\right| \times 100\% = 12.5\%$$

For x_2 and x_3, the error estimates are $\varepsilon_{a,2} = 11.8\%$ and $\varepsilon_{a,3} = 0.076\%$. Note that, as was the case when determining roots of a single equation, formulations such as Eq. (12.2) usually provide a conservative appraisal of convergence. Thus, when they are met, they ensure that the result is known to at least the tolerance specified by ε_s.

Tactic of Gauss-Seidel Method

- As each new *x* value is computed for the Gauss-Seidel method, it is immediately used in the next equation to determine another *x* value
- Thus, if the solution is converging, the best available estimates will be employed for the Gauss-Seidel method

Jacobi Iteration

- The ***Jacobi iteration*** is similar to the Gauss-Seidel method, except the *j*-1th information is used to update all variables in the *j* th iteration:
  - Gauss-Seidel
  - Jacobi

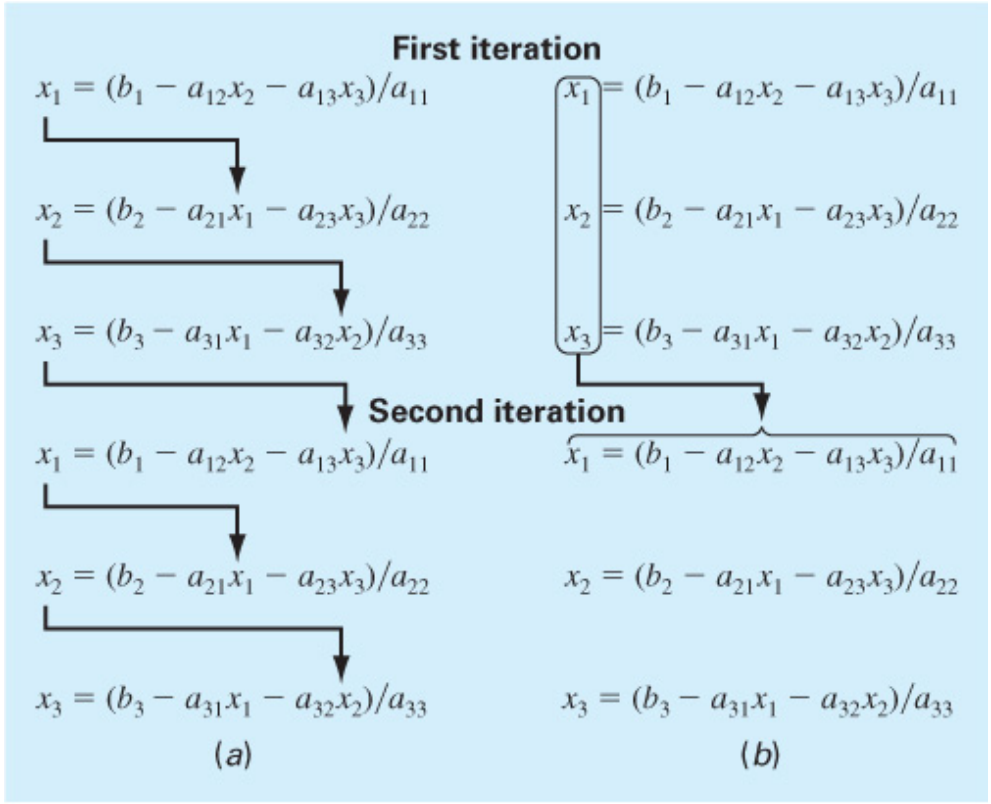


FIGURE 12.1
Graphical depiction of the difference between (*a*) the Gauss-Seidel and (*b*) the Jacobi iterative methods for solving simultaneous linear algebraic equations.

Diagonal Dominance

- ***The Gauss-Seidel method*** may diverge, but if the system is **diagonally dominant,** it will definitely converge

- Diagonal dominance means:

$$|a_{ii}| > \sum_{\substack{j=1 \\ j \neq i}}^{n} |a_{ij}|$$

MATLAB Program

```
function x = GaussSeidel(A,b,es,maxit)
% GaussSeidel: Gauss Seidel method
%   x = GaussSeidel(A,b): Gauss Seidel without relaxation
% input:
%   A = coefficient matrix
%   b = right hand side vector
%   es = stop criterion (default = 0.00001%)
%   maxit = max iterations (default = 50)
% output:
%   x = solution vector

if nargin<2,error('at least 2 input arguments required'),end
if nargin<4|isempty(maxit),maxit=50;end
if nargin<3|isempty(es),es=0.00001;end
[m,n] = size(A);
if m~=n, error('Matrix A must be square'); end
C = A;
for i = 1:n
  C(i,i) = 0;
  x(i) = 0;
end
x = x';
for i = 1:n
  C(i,1:n) = C(i,1:n)/A(i,i);
end
for i = 1:n
  d(i) = b(i)/A(i,i);
end
iter = 0;
while (1)
  xold = x;
  for i = 1:n
    x(i) = d(i)-C(i,:)*x;
    if x(i) ~= 0
      ea(i) = abs((x(i) - xold(i))/x(i)) * 100;
    end
  end
  iter = iter+1;
  if max(ea)<=es | iter >= maxit, break, end
end
```

FIGURE 12.2
MATLAB M-file to implement Gauss-Seidel.

$$x_1^{new} = \frac{b_1}{a_{11}} - \frac{a_{12}}{a_{11}}x_2^{old} - \frac{a_{13}}{a_{11}}x_3^{old}$$

$$x_2^{new} = \frac{b_2}{a_{22}} - \frac{a_{21}}{a_{22}}x_1^{new} - \frac{a_{23}}{a_{22}}x_3^{old}$$

$$x_3^{new} = \frac{b_3}{a_{33}} - \frac{a_{31}}{a_{33}}x_1^{new} - \frac{a_{32}}{a_{33}}x_2^{new}$$

Notice that the solution can be expressed concisely in matrix form as

$$\{x\} = \{d\} - [C]\{x\}$$

where

$$\{d\} = \begin{Bmatrix} b_1/a_{11} \\ b_2/a_{22} \\ b_3/a_{33} \end{Bmatrix}$$

and

$$[C] = \begin{bmatrix} 0 & a_{12}/a_{11} & a_{13}/a_{11} \\ a_{21}/a_{22} & 0 & a_{23}/a_{22} \\ a_{31}/a_{33} & a_{32}/a_{33} & 0 \end{bmatrix}$$

Relaxation

- To enhance convergence, an iterative program can introduce ***relaxation*** where the value at a particular iteration is made up of a combination of the old value and the newly calculated value:

$$x_i^{\text{new}} = \lambda x_i^{\text{new}} + (1-\lambda) x_i^{\text{old}}$$

  - where λ is a weighting factor that is assigned a value between 0 and 2
    - $0<\lambda<1$: underrelaxation
    - $\lambda=1$: no relaxation
    - $1<\lambda\leq 2$: overrelaxation
  - The choice of a proper value for λ is highly problem-specific and is often determined empirically

Gauss-Seidel with Relaxation: An Example (1/3)

Problem Statement. Solve the following system with Gauss-Seidel using overrelaxation ($\lambda = 1.2$) and a stopping criterion of $\varepsilon_s = 10\%$:

$$-3x_1 + 12x_2 = 9$$
$$10x_1 - 2x_2 = 8$$

Solution. First rearrange the equations so that they are diagonally dominant and solve the first equation for x_1 and the second for x_2:

$$x_1 = \frac{8 + 2x_2}{10} = 0.8 + 0.2x_2$$
$$x_2 = \frac{9 + 3x_1}{12} = 0.75 + 0.25x_1$$

Example 12.2

First iteration: Using initial guesses of $x_1 = x_2 = 0$, we can solve for x_1:

$$x_1 = 0.8 + 0.2(0) = 0.8$$

Before solving for x_2, we first apply relaxation to our result for x_1:

$$x_{1,r} = 1.2(0.8) - 0.2(0) = 0.96$$

We use the subscript r to indicate that this is the "relaxed" value. This result is then used to compute x_2:

$$x_2 = 0.75 + 0.25(0.96) = 0.99$$

We then apply relaxation to this result to give

$$x_{2,r} = 1.2(0.99) - 0.2(0) = 1.188$$

At this point, we could compute estimated errors with Eq. (12.2). However, since we started with assumed values of zero, the errors for both variables will be 100%.

Gauss-Seidel with Relaxation: An Example (2/3)

Second iteration: Using the same procedure as for the first iteration, the second iteration yields

$$x_1 = 0.8 + 0.2(1.188) = 1.0376$$
$$x_{1,r} = 1.2(1.0376) - 0.2(0.96) = 1.05312$$
$$\varepsilon_{a,1} = \left|\frac{1.05312 - 0.96}{1.05312}\right| \times 100\% = 8.84\%$$

Example 12.2

$$x_2 = 0.75 + 0.25(1.05312) = 1.01328$$
$$x_{2,r} = 1.2(1.01328) - 0.2(1.188) = 0.978336$$
$$\varepsilon_{a,2} = \left|\frac{0.978336 - 1.188}{0.978336}\right| \times 100\% = 21.43\%$$

Because we have now have nonzero values from the first iteration, we can compute approximate error estimates as each new value is computed. At this point, although the error estimate for the first unknown has fallen below the 10% stopping criterion, the second has not. Hence, we must implement another iteration.

Gauss-Seidel with Relaxation: An Example (3/3)

Third iteration:

$$x_1 = 0.8 + 0.2(0.978336) = 0.995667$$
$$x_{1,r} = 1.2(0.995667) - 0.2(1.05312) = 0.984177$$
$$\varepsilon_{a,1} = \left|\frac{0.984177 - 1.05312}{0.984177}\right| \times 100\% = 7.01\%$$

$$x_2 = 0.75 + 0.25(0.984177) = 0.996044$$
$$x_{2,r} = 1.2(0.996044) - 0.2(0.978336) = 0.999586$$
$$\varepsilon_{a,2} = \left|\frac{0.999586 - 0.978336}{0.999586}\right| \times 100\% = 2.13\%$$

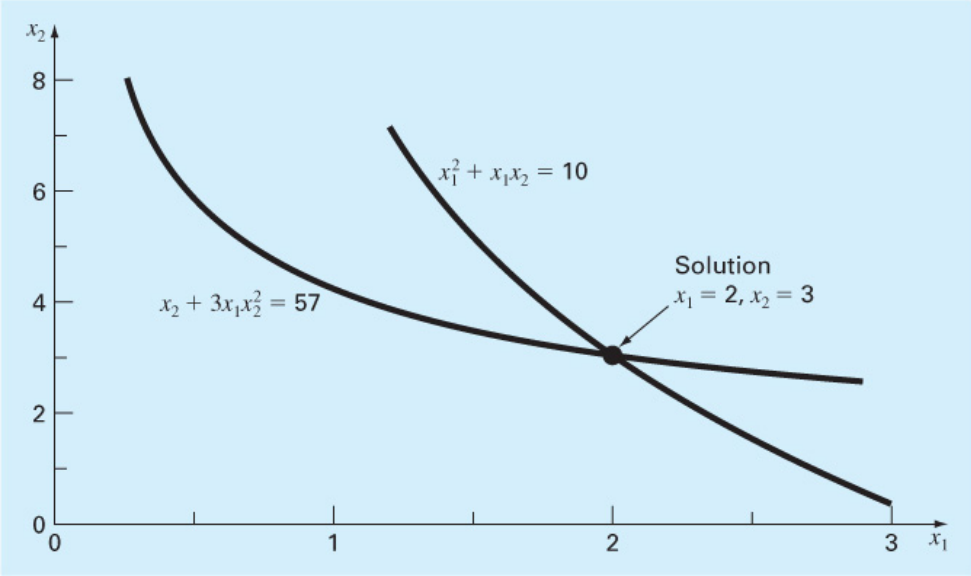
Example 12.2

At this point, we can terminate the computation because both error estimates have fallen below the 10% stopping criterion. The results at this juncture, $x_1 = 0.984177$ and $x_2 = 0.999586$, are converging on the exact solution of $x_1 = x_2 = 1$.

Nonlinear Systems

- Nonlinear systems can also be solved using the same strategy as the ***Gauss-Seidel method***
  - Solve each system for one of the unknowns and update each unknown using information from the previous iteration
- This is called ***successive substitution***

FIGURE 12.3
Graphical depiction of the solution of two simultaneous nonlinear equations.



$$x_1^2 + x_1 x_2 = 10$$
$$x_2 + 3x_1 x_2^2 = 57$$

Successive Substitution: An Example (1/2)

Problem Statement. Use successive substitution to determine the roots of Eq. (12.6). Note that a correct pair of roots is $x_1 = 2$ and $x_2 = 3$. Initiate the computation with guesses of $x_1 = 1.5$ and $x_2 = 3.5$.

$$x_1^2 + x_1 x_2 = 10$$
$$x_2 + 3x_1 x_2^2 = 57$$

Solution. Equation (12.6*a*) can be solved for

$$x_1 = \frac{10 - x_1^2}{x_2} \qquad \text{(E12.3.1)}$$

and Eq. (12.6*b*) can be solved for

$$x_2 = 57 - 3x_1 x_2^2 \qquad \text{(E12.3.2)}$$

On the basis of the initial guesses, Eq. (E12.3.1) can be used to determine a new value of x_1:

$$x_1 = \frac{10 - (1.5)^2}{3.5} = 2.21429$$

This result and the initial value of $x_2 = 3.5$ can be substituted into Eq. (E12.3.2) to determine a new value of x_2:

$$x_2 = 57 - 3(2.21429)(3.5)^2 = -24.37516$$

Thus, the approach seems to be diverging. This behavior is even more pronounced on the second iteration:

$$x_1 = \frac{10 - (2.21429)^2}{-24.37516} = -0.20910$$

$$x_2 = 57 - 3(-0.20910)(-24.37516)^2 = 429.709$$

Obviously, the approach is deteriorating.

Example 12.3

Successive Substitution: An Example (2/2)

Now we will repeat the computation but with the original equations set up in a different format. For example, an alternative solution of Eq. (12.6*a*) is

$$x_1 = \sqrt{10 - x_1 x_2}$$

and of Eq. (12.6*b*) is

$$x_2 = \sqrt{\frac{57 - x_2}{3x_1}}$$

Now the results are more satisfactory:

$$x_1 = \sqrt{10 - 1.5(3.5)} = 2.17945$$

$$x_2 = \sqrt{\frac{57 - 3.5}{3(2.17945)}} = 2.86051$$

$$x_1 = \sqrt{10 - 2.17945(2.86051)} = 1.94053$$

$$x_2 = \sqrt{\frac{57 - 2.86051}{3(1.94053)}} = 3.04955$$

Thus, the approach is converging on the true values of $x_1 = 2$ and $x_2 = 3$.

1. For successive substitution, convergence often depends on the manner in which the equations are formulated
2. Divergence also can occur if the initial guesses are insufficiently close to the true solution

Example 12.3

Newton-Raphson (1/3)

- Nonlinear systems may also be solved using the ***Newton-Raphson method*** for multiple variables
- For ***a two-variable system***, the Taylor series approximation and resulting Newton-Raphson equations are:

$$f_{1,i+1} = f_{1,i} + \left(x_{1,i+1} - x_{1,i}\right)\frac{\partial f_{1,i}}{\partial x_1} + \left(x_{2,i+1} - x_{2,i}\right)\frac{\partial f_{1,i}}{\partial x_2}$$

$$f_{2,i+1} = f_{2,i} + \left(x_{1,i+1} - x_{1,i}\right)\frac{\partial f_{2,i}}{\partial x_1} + \left(x_{2,i+1} - x_{2,i}\right)\frac{\partial f_{2,i}}{\partial x_2}$$

$$f_{1,i+1} = 0$$

$$\Rightarrow$$

$$f_{2,i+1} = 0$$

$$\frac{\partial f_{1,i}}{\partial x_1}x_{1,i+1} + \frac{\partial f_{1,i}}{\partial x_2}x_{2,i+1} = -f_{1,i} + \frac{\partial f_{1,i}}{\partial x_1}x_{1,i} + \frac{\partial f_{1,i}}{\partial x_2}x_{2,i}$$

$$\frac{\partial f_{1,i}}{\partial x_1}x_{1,i+1} + \frac{\partial f_{1,i}}{\partial x_2}x_{2,i+1} = -f_{1,i} + \frac{\partial f_{1,i}}{\partial x_1}x_{1,i} + \frac{\partial f_{2,i}}{\partial x_2}x_{2,i}$$

Recall in Chapter 6, Newton-Raphon for the root of a single equation:

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

(first-order Taylor series expansion of $f(\cdot)$)

$\Rightarrow$

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

Newton-Raphson (2/3)

– After algebraic manipulation

$$x_{1,i+1} = x_{1,i} - \frac{f_{1,i}\frac{\partial f_{2,i}}{\partial x_2} - f_{2,i}\frac{\partial f_{1,i}}{\partial x_2}}{\frac{\partial f_{1,i}}{\partial x_1}\frac{\partial f_{2,i}}{\partial x_2} - \frac{\partial f_{1,i}}{\partial x_2}\frac{\partial f_{2,i}}{\partial x_1}}$$

$$x_{2,i+1} = x_{2,i} - \frac{f_{2,i}\frac{\partial f_{1,i}}{\partial x_1} - f_{1,i}\frac{\partial f_{2,i}}{\partial x_1}}{\frac{\partial f_{1,i}}{\partial x_1}\frac{\partial f_{2,i}}{\partial x_2} - \frac{\partial f_{1,i}}{\partial x_2}\frac{\partial f_{2,i}}{\partial x_1}}$$

Referred to as the determinant of the Jacobian of the system

$$[J] = \begin{bmatrix} \frac{\partial f_{1,i}}{\partial x_1} & \frac{\partial f_{1,i}}{\partial x_2} \\ \frac{\partial f_{2,i}}{\partial x_1} & \frac{\partial f_{2,i}}{\partial x_2} \end{bmatrix}$$

Referred to as the Jacobian of the system

Newton-Raphson (3/3)

- Matrix notion of Newton-Raphson
  - Also can be generalized to *n* simultaneous equations

$$[J]\{x_{i+1}\} = -\{f\} + [J]\{x_i\}$$

$$\Rightarrow$$

$$\{x_{i+1}\} = \{x_i\} - [J]^{-1}\{f\}$$

Newton-Raphson: An Example

Newton-Raphson for a Nonlinear System

Problem Statement. Use the multiple-equation Newton-Raphson method to determine roots of Eq. (12.6). Initiate the computation with guesses of $x_1 = 1.5$ and $x_2 = 3.5$.

Solution. First compute the partial derivatives and evaluate them at the initial guesses of x and y:

$$\frac{\partial f_{1,0}}{\partial x_1} = 2x_1 + x_2 = 2(1.5) + 3.5 = 6.5 \qquad \frac{\partial f_{1,0}}{\partial x_2} = x_1 = 1.5$$

$$\frac{\partial f_{2,0}}{\partial x_1} = 3x_2^2 = 3(3.5)^2 = 36.75 \qquad \frac{\partial f_{2,0}}{\partial x_2} = 1 + 6x_1x_2 = 1 + 6(1.5)(3.5) = 32.5$$

Thus, the determinant of the Jacobian for the first iteration is

$$6.5(32.5) - 1.5(36.75) = 156.125$$

The values of the functions can be evaluated at the initial guesses as

$$f_{1,0} = (1.5)^2 + 1.5(3.5) - 10 = -2.5$$
$$f_{2,0} = 3.5 + 3(1.5)(3.5)^2 - 57 = 1.625$$

These values can be substituted into Eq. (12.12) to give

$$x_1 = 1.5 - \frac{-2.5(32.5) - 1.625(1.5)}{156.125} = 2.03603$$

$$x_2 = 3.5 - \frac{1.625(6.5) - (-2.5)(36.75)}{156.125} = 2.84388$$

Thus, the results are converging to the true values of $x_1 = 2$ and $x_2 = 3$. The computation can be repeated until an acceptable accuracy is obtained.

$$x_1^2 + x_1x_2 = 10$$
$$x_2 + 3x_1x_2^2x_1^2 = 57$$
$$\Rightarrow$$
$$f_1(x_1, x_2) = x_1^2 + x_1x_2 - 10 = 0$$
$$f_2(x_1, x_2) = x_2 + 3x_1x_2^2x_1^2 - 57 = 0$$

Example 12.4

MATLAB Program

```
function [x,f,ea,iter]=newtmult(func,x0,es,maxit,varargin)
% newtmult: Newton-Raphson root zeroes nonlinear systems
%  [x,f,ea,iter]=newtmult(func,x0,es,maxit,p1,p2,...):
%    uses the Newton-Raphson method to find the roots of
%    a system of nonlinear equations
% input:
%   func = function handle to function that returns f and J
%   x0 = initial guess
%   es = desired percent relative error (default = 0.0001%)
%   maxit = maximum allowable iterations (default = 50)
%   p1,p2,... = additional parameters used by function
% output:
%   x = vector of roots
%   f = vector of functions evaluated at roots
%   ea = approximate percent relative error (%)
%   iter = number of iterations

if nargin<2,error('at least 2 input arguments required'),end
if nargin<3||isempty(es),es=0.0001;end
if nargin<4||isempty(maxit),maxit=50;end
iter = 0;
x=x0;
while (1)
  [J,f]=func(x,varargin{:});
  dx=J\f;
  x=x-dx;
  iter = iter + 1;
  ea=100*max(abs(dx./x));
  if iter>=maxit||ea<=es, break, end
end
```