Calcul Scientifique

Ecole Hassania des Travaux Publics

Salem Nafiri

Méthodes Itératives pour la Résolution de Ax=b

Iterative Methods for System of Equations

Iterative Methods for Solving Matrix Equations

$$Ax = b \quad \Rightarrow$$
$$x = Cx + d\,, \quad C_{ii} = 0$$

- **Jacobi method**
- **Gauss-Seidel Method**
- **Successive Over Relaxation (SOR)**

Iterative Methods

- **Idea behind iterative methods:**
- **Convert Ax = b into x = Cx +d**

$$Ax = b \quad \Leftrightarrow \quad x = Cx + d$$

Equivalent system

- **Generate a sequence of approximations (iteration) x^1, x^2, …., with initial x^0**

$$x^j = Cx^{j-1} + d$$

- **Similar to fix-point iteration method**

Rearrange Matrix Equations

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + a_{14}x_4 = b_1 \\ a_{21}x_1 + a_{22}x_2 + a_{23}x_3 + a_{24}x_4 = b_2 \\ a_{31}x_1 + a_{32}x_2 + a_{33}x_3 + a_{34}x_4 = b_3 \\ a_{41}x_1 + a_{42}x_2 + a_{43}x_3 + a_{44}x_4 = b_4 \end{cases}$$

- **Rewrite the matrix equation in the same way**

$$\begin{cases} x_1 = & -\frac{a_{12}}{a_{11}}x_2 - \frac{a_{13}}{a_{11}}x_3 - \frac{a_{14}}{a_{11}}x_4 + \frac{b_1}{a_{11}} \\ x_2 = -\frac{a_{21}}{a_{22}}x_1 & -\frac{a_{23}}{a_{22}}x_3 - \frac{a_{24}}{a_{22}}x_4 + \frac{b_2}{a_{22}} \\ x_3 = -\frac{a_{31}}{a_{33}}x_1 - \frac{a_{32}}{a_{33}}x_2 & -\frac{a_{34}}{a_{33}}x_4 + \frac{b_3}{a_{33}} \\ x_4 = -\frac{a_{41}}{a_{44}}x_1 - \frac{a_{42}}{a_{44}}x_2 - \frac{a_{43}}{a_{44}}x_3 & + \frac{b_4}{a_{44}} \end{cases}$$

n equations

n variables

Iterative Methods

$$Ax = b \quad \Leftrightarrow \quad x^{j} = Cx^{j-1} + d \; ; \; C_{ii} = 0$$

- ***x*** **and** ***d*** **are column vectors, and** ***C*** **is a square matrix**

$$C = \begin{bmatrix} 0 & -\frac{a_{12}}{a_{11}} & -\frac{a_{13}}{a_{11}} & -\frac{a_{14}}{a_{11}} \\ -\frac{a_{21}}{a_{22}} & 0 & -\frac{a_{23}}{a_{22}} & -\frac{a_{24}}{a_{22}} \\ -\frac{a_{31}}{a_{33}} & -\frac{a_{32}}{a_{33}} & 0 & -\frac{a_{34}}{a_{33}} \\ -\frac{a_{41}}{a_{44}} & -\frac{a_{42}}{a_{44}} & -\frac{a_{43}}{a_{44}} & 0 \end{bmatrix} \; ; \quad d = \begin{bmatrix} \frac{b_1}{a_{11}} \\ \frac{b_2}{a_{22}} \\ \frac{b_3}{a_{33}} \\ \frac{b_4}{a_{44}} \end{bmatrix}$$

Convergence Criterion

$$\begin{cases} (1) \quad \varepsilon_{a,i} = \left| \dfrac{x_i^j - x_i^{j-1}}{x_i^j} \right| \times 100\% < \varepsilon_s \quad \text{for all } x_i \\ (2) \quad \text{Norm of the residual vector} \quad |Ax - b| < \varepsilon_s \end{cases}$$

- **For system of equations**

$$|\lambda_i| \le 1, \quad \lambda = eig(C)$$

Jacobi and Gauss-Seidel

Example:

$$\begin{cases} 5x_1 - 2x_2 + 2x_3 = 10 \\ 2x_1 - 4x_2 - x_3 = -7 \\ 3x_1 - x_2 + 6x_3 = 12 \end{cases}$$

Jacobi

$$\begin{cases} x_1^{new} = \frac{2}{5}x_2^{old} - \frac{2}{5}x_3^{old} + \frac{10}{5} \\ x_2^{new} = \frac{2}{4}x_1^{old} - \frac{1}{4}x_3^{old} + \frac{7}{4} \\ x_3^{new} = -\frac{3}{6}x_1^{old} + \frac{1}{6}x_2^{old} + \frac{12}{6} \end{cases}$$

Gauss-Seidel

$$\begin{cases} x_1^{new} = \frac{2}{5}x_2^{old} - \frac{2}{5}x_3^{old} + \frac{10}{5} \\ x_2^{new} = \frac{2}{4}x_1^{new} - \frac{1}{4}x_3^{old} + \frac{7}{4} \\ x_3^{new} = -\frac{3}{6}x_1^{new} + \frac{1}{6}x_2^{new} + \frac{12}{6} \end{cases}$$

Jacobi Code

- Matlab function for Jacobi method

```
function x = Jacobi_f(A, b, x0, tol, max)
%   Inputs :
%      A       coefficient matrix (n-by-n)
%      b       right-hand side (n-by-1)
%      x0      initial solution (n-by-1)
%      tol     stop if norm of change in x < tol
%      max     maximum number of iterations
%   Outputs :
%      x       solution vector (n-by-1)
[n m] = size(A);   xold = x0;   C = -A;
for i=1:n
   C(i,i) = 0;
end
for i=1:n
   C(i,:)=C(i,:)/A(i,i);
end
for i=1:n
   d(i,1) = b(i)/A(i,i);
end
```

```
i = 1;
while (i <= max)
   xnew = C * xold + d;
   if norm(xnew-xold) <= tol
      x = xnew;
      disp('Jacobi method converged');
      disp ([i   xnew']);
      return;
   else
      xold=xnew;
   end
   disp ([i   xnew']);
   i=i+1;
end
disp('Jacobi method did not converge');
disp('results after maximum number of iterations');
x = xnew;
```

Convergence and Diagonal Dominant

- **Sufficient condition -- A is diagonally dominant**
- **Diagonally dominant: the magnitude of the diagonal element is larger than the sum of absolute value of the other elements in the row**

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

- **Necessary and sufficient condition -- the magnitude of the largest eigenvalue of *C* (*not A*) is less than 1**
- **Fortunately, many engineering problems of practical importance satisfy this requirement**
- **Use partial pivoting to rearrange equations!**

Diagonally Dominant Matrix

$$A = \begin{bmatrix} 8 & 1 & -2 & -3 \\ -1 & -10 & 2 & 5 \\ 1 & -6 & 12 & -3 \\ -3 & 2 & 3 & -9 \end{bmatrix}$$

is strictly diagonally dominant because diagonal elements are greater than sum of absolute value of other elements in row

$$A = \begin{bmatrix} 8 & 1 & -2 & -3 \\ -1 & 6 & 2 & 5 \\ 1 & -6 & 12 & -3 \\ -3 & 2 & 3 & -9 \end{bmatrix}$$

is not diagonally dominant

Example

$$\begin{aligned} -5x_1 \qquad\quad &+ 12x_3 = 80 \\ 4x_1 - x_2 \quad &- x_3 = -2 \\ 6x_1 + 8x_2 \quad &= 45 \end{aligned}$$

$$\begin{bmatrix} -5 & 0 & 12 \\ 4 & -1 & -1 \\ 6 & 8 & 0 \end{bmatrix}$$

Not diagonally dominant !!

Order of the equation can be important

Rearrange the equations to ensure convergence

$$\begin{aligned} 4x_1 - x_2 \quad &- x_3 = -2 \\ 6x_1 + 8x_2 \quad &= 45 \\ -5x_1 \qquad\quad &+ 12x_3 = 80 \end{aligned}$$

$$\begin{bmatrix} 4 & -1 & -1 \\ 6 & 8 & 0 \\ -5 & 0 & 12 \end{bmatrix}$$

Gauss-Seidel Iteration

Rearrange

$$\begin{cases} x_1 = (x_2 + x_3 - 2)/4 \\ x_2 = (45 - 6x_1)/8 \\ x_3 = (80 + 5x_1)/12 \end{cases}$$

Assume $x_1 = x_2 = x_3 = 0$

First iteration

$$\begin{cases} x_1 = (0 + 0 - 2)/4 = -0.5 \\ x_2 = [45 - 6 \times (-0.5)]/8 = 6.0 \\ x_3 = [80 + 5 \times (-0.5)]/12 = 6.4583 \end{cases}$$

Gauss-Seidel Method

Second iteration

$$\begin{cases} x_1 = (-2+6+6.4583)/4 = 2.6146 \\ x_2 = (45-6(2.6146))/8 = 3.6641 \\ x_3 = (80+5(2.6146))/12 = 7.7561 \end{cases}$$

Third iteration

$$\begin{cases} x_1 = (-2+3.6641+7.7561)/4 = 2.3550 \\ x_2 = (45-6(2.3550))/8 = 3.8587 \\ x_3 = (80+5(2.3550))/12 = 7.6479 \end{cases}$$

Fourth iteration

$$\begin{cases} x_1 = (-2+3.8587+7.6479)/4 = 2.3767 \\ x_2 = (45-6(2.3767))/8 = 3.8425 \\ x_3 = (80+5(2.3767))/12 = 7.6569 \end{cases}$$

$$5th: \quad x_1 = 2.3749, \quad x_2 = 3.8439, \quad x_3 = 7.6562$$

$$6th: \quad x_1 = 2.3750, \quad x_2 = 3.8437, \quad x_3 = 7.6563$$

$$7th: \quad x_1 = 2.3750, \quad x_2 = 3.8438, \quad x_3 = 7.6562$$

MATLAB M-File for Gauss-Seidel method

```
function x = GaussSeidel(A, b, es, maxit)
% GaussSeidel(A,b): Gauss-Seidel method.
% input:
%   A = Coefficient Matrix
%   b = right hand side vector
%   es = (optional) stop criterion (%) (default = 0.00001)
%   maxit = (optional) maximum iterations (default = 50)
% Output
%   x = solution vector
if nargin < 4, maxit = 50; end
if nargin < 3, 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
```

Continued on next page

MATLAB M-File for Gauss-Seidel method

Continued from previous page

```
disp('       i          x1          x2          x3          x4   ...');
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;
    disp([iter  x'])
    if max(ea) <= es | iter >= maxit, break, end
end
if iter >= maxit
disp('Gauss Seidel method did not converge');
disp('results after maximum number of iterations');
else
    disp('Gauss Seidel method has converged');
end
x;
```

Gauss-Seidel Iteration

```
» A = [4 -1 -1; 6 8 0; -5 0 12];
» b = [-2 45 80];
» x=Seidel(A,b,x0,tol,100);
      i          x1          x2          x3      x4 ....
    1.0000   -0.5000     6.0000     6.4583
    2.0000    2.6146     3.6641     7.7561
    3.0000    2.3550     3.8587     7.6479
    4.0000    2.3767     3.8425     7.6569
    5.0000    2.3749     3.8439     7.6562
    6.0000    2.3750     3.8437     7.6563
    7.0000    2.3750     3.8438     7.6562
    8.0000    2.3750     3.8437     7.6563
Gauss-Seidel method converged
```

Converges faster than the Jacobi method shown in next page

Jacobi Iteration

```
» A = [4 -1 -1; 6 8 0; -5 0 12];
» b = [-2 45 80];
» x=Jacobi(A,b,0.0001,100);
        i          x1          x2          x3          x4 ....
   1.0000    -0.5000     5.6250     6.6667
   2.0000     2.5729     6.0000     6.4583
   3.0000     2.6146     3.6953     7.7387
   4.0000     2.3585     3.6641     7.7561
   5.0000     2.3550     3.8561     7.6494
   6.0000     2.3764     3.8587     7.6479
   7.0000     2.3767     3.8427     7.6568
   8.0000     2.3749     3.8425     7.6569
   9.0000     2.3749     3.8438     7.6562
  10.0000     2.3750     3.8439     7.6562
  11.0000     2.3750     3.8437     7.6563
  12.0000     2.3750     3.8437     7.6563
  13.0000     2.3750     3.8438     7.6562
  14.0000     2.3750     3.8438     7.6562

Jacobi method converged
```

Relaxation Method

- **Relaxation (weighting) factor λ**
- **Gauss-Seidel method: $\lambda = 1$**
- **Overrelaxation** $1 < \lambda < 2$ ***(faster cv)***
- **Underrelaxation** $0 < \lambda < 1$ ***(slower cv)***

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

- ***Successive Over-relaxation (SOR)***

Successive Over Relaxation (SOR)

- **Relaxation method**

$$\text{G-S method } x_2^{new} = (b_2 - a_{21}x_1^{new} - a_{23}x_3^{old} - a_{24}x_4^{old})/a_{22}$$

$$\text{SOR method } x_2^{new} = (1-\lambda)x_2^{old} + \lambda x_2^{new}$$

$$= (1-\lambda)x_2^{old} + \lambda(b_2 - a_{21}x_1^{new} - a_{23}x_3^{old} - a_{24}x_4^{old})/a_{22}$$

$$\begin{cases} x_1^{new} = (1-\lambda)x_1^{old} + \lambda(b_1 - a_{12}x_2^{old} - a_{13}x_3^{old} - a_{14}x_4^{old})/a_{11} \\ x_2^{new} = (1-\lambda)x_2^{old} + \lambda(b_2 - a_{21}x_1^{new} - a_{23}x_3^{old} - a_{24}x_4^{old})/a_{22} \\ x_3^{new} = (1-\lambda)x_3^{old} + \lambda(b_3 - a_{31}x_1^{new} - a_{32}x_2^{new} - a_{34}x_4^{old})/a_{33} \\ x_4^{new} = (1-\lambda)x_4^{old} + \lambda(b_4 - a_{41}x_1^{new} - a_{42}x_2^{new} - a_{43}x_3^{new})/a_{44} \end{cases}$$

SOR Iterations

Rearrange

$$\begin{cases} x_1 = (x_2 + x_3 - 2)/4 \\ x_2 = (45 - 6x_1)/8 \\ x_3 = (80 + 5x_1)/12 \end{cases}$$

Assume $x_1 = x_2 = x_3 = 0$, and $\lambda = 1.2$

$$x_i = \lambda\, x_i^{G-S} + (1-\lambda)\, x_i^{old};\ G\text{-}S : Gauss\text{-}Seidel$$

First iteration

$$\begin{cases} x_1 = (-0.2)\times 0 + 1.2\times(0+0-2)/4 = -0.6 \\ x_2 = (-0.2)\times 0 + 1.2\times[45 - 6\times(-0.6)]/8 = 7.29 \\ x_3 = (-0.2)\times 0 + 1.2\times[80 + 5\times(-0.6)]/12 = 7.7 \end{cases}$$

SOR Iterations

Second iteration

$$\begin{cases} x_1 = (-0.2)\times(-0.6)+1.2\times(7.29+7.7-2)/4 = 4.017 \\ x_2 = (-0.2)\times(7.29)+1.2\times(45-6(4.017))/8 = 1.6767 \\ x_3 = (-0.2)\times(7.7)+1.2\times(80+5(4.017))/12 = 8.4685 \end{cases}$$

Third iteration

$$\begin{cases} x_1 = (-0.2)\times(4.017)+1.2\times(1.6767+8.4685-2)/4 = 1.6402 \\ x_2 = (-0.2)\times(1.6767)+1.2\times(45-6(1.6402))/8 = 4.9385 \\ x_3 = (-0.2)\times(8.4685)+1.2\times(80+5(1.6402))/12 = 7.1264 \end{cases}$$

- **Converges slower !! (see MATLAB solutions)**
- **There is an optimal relaxation parameter**

Successive Over Relaxation

```
function x = SOR(A, b, x0, w, tol, max)
% Solution of the system of linear equations Ax=b
% using SOR iterative algorithm
% Outputs : x --- solution vector (n-by-1)

[n,m] = size(A); x=x0; C=-A;
for i = 1 : n
    C(i,i) = 0;
end
for i = 1 : n
    C(i,1:n) = C(i,1:n)/A(i,i);
end
for i = 1 : n
    r(i,1) = b(i)/A(i,i);
end
i = 1;
disp('   i       x1       x2       x3     ....')
while (i <= max)
    xold = x;      % save solution from previous step
   for j = 1 : n
     x(j) = (1-w)*xold(j) + w*(C(j,:) * x + r(j));
   end
   if norm(xold-x) <= tol
      disp('SOR method converged'); return;
   end
   disp([ i    x'])
   i = i + 1;
end
disp('SOR method did not converge');
```

Relaxation factor

w (= λ)

SOR with $\lambda = 1.2$

```
» [A,b]=Example

A =
 4     -1     -1
 6      8      0
-5      0     12

b =
    -2
    45
    80

» x0=[0 0 0]'
x0 =
     0
     0
     0
» tol=1.e-6
tol =
  1.0000e-006
```

Converge Slower !!

```
» w=1.2; x = SOR(A, b, x0, w, tol, 100);
        i          x1          x2          x3      ....
   1.0000    -0.6000     7.2900     7.7000
   2.0000     4.0170     1.6767     8.4685
   3.0000     1.6402     4.9385     7.1264
   4.0000     2.6914     3.3400     7.9204
   5.0000     2.2398     4.0661     7.5358
   6.0000     2.4326     3.7474     7.7091
   7.0000     2.3504     3.8851     7.6334
   8.0000     2.3855     3.8261     7.6661
   9.0000     2.3705     3.8513     7.6521
  10.0000     2.3769     3.8405     7.6580
  11.0000     2.3742     3.8451     7.6555
  12.0000     2.3753     3.8432     7.6566
  13.0000     2.3749     3.8440     7.6561
  14.0000     2.3751     3.8436     7.6563
  15.0000     2.3750     3.8438     7.6562
  16.0000     2.3750     3.8437     7.6563
  17.0000     2.3750     3.8438     7.6562
  18.0000     2.3750     3.8437     7.6563
  19.0000     2.3750     3.8438     7.6562
  20.0000     2.3750     3.8437     7.6563
  21.0000     2.3750     3.8438     7.6562

SOR method converged
```

Nonlinear Systems

- **Simultaneous nonlinear equations**

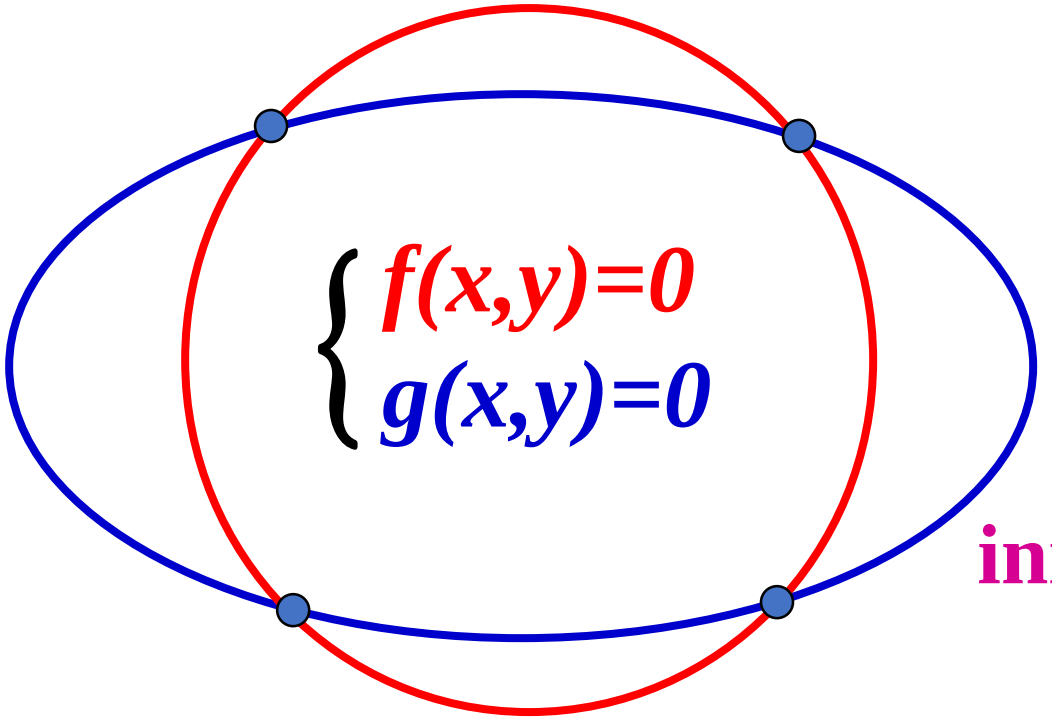
$$f_1(x_1, x_2, x_3, \cdots, x_n) = 0$$
$$f_2(x_1, x_2, x_3, \cdots, x_n) = 0$$
$$f_3(x_1, x_2, x_3, \cdots, x_n) = 0$$
$$\vdots$$
$$f_n(x_1, x_2, x_3, \cdots, x_n) = 0$$

- **Example**

$$x_1^2 + 2x_2^2 + 4x_3 = 7$$
$$2x_1x_2 + 5x_2 - 7x_2x_3^2 = 8$$
$$5x_1 - 2x_2 + x_3^2 = 4$$

Two Nonlinear Functions

Circle $x^2 + y^2 = r^2$

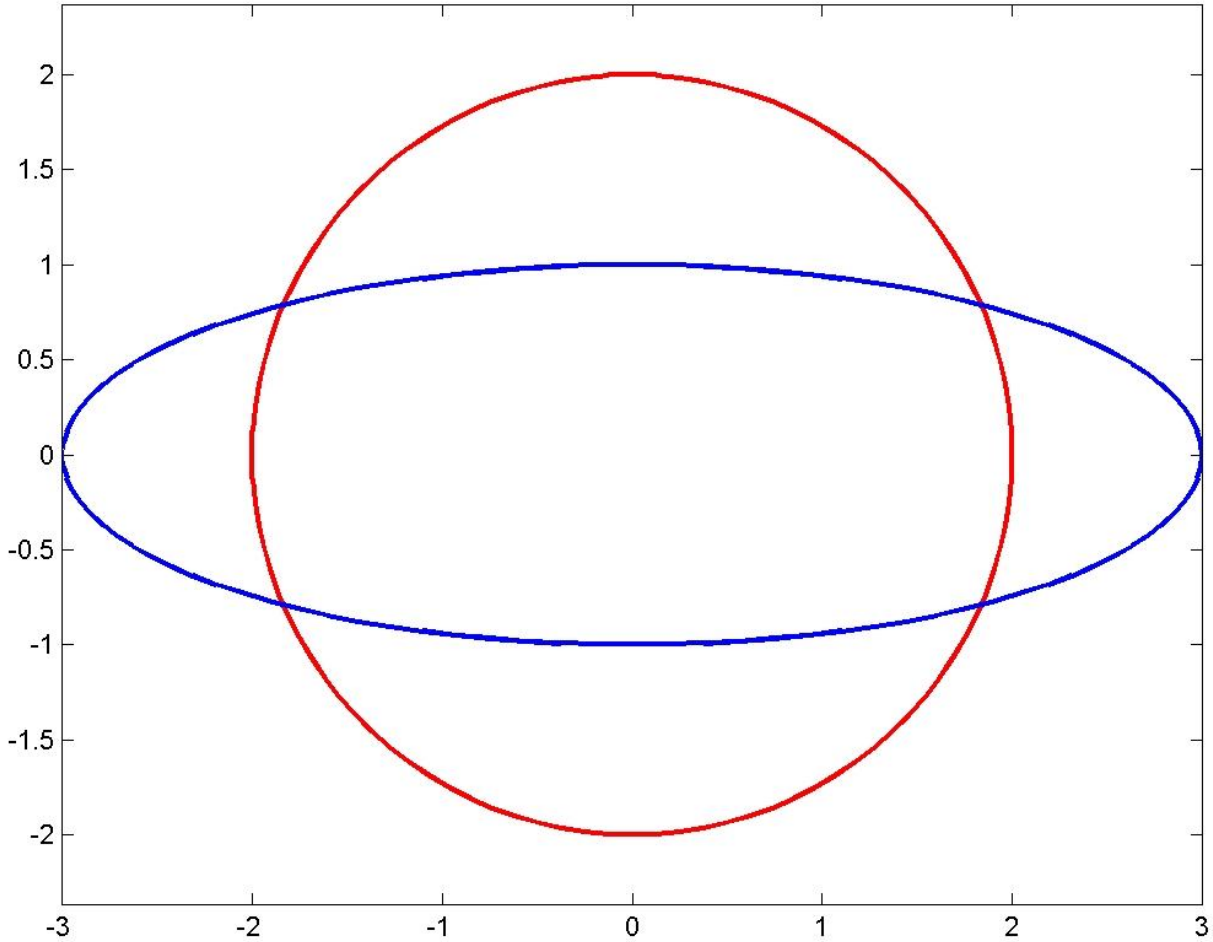


Solution depends on initial guesses

Ellipse $(x/a)^2 + (y/b)^2 = 1$

```
function [x1,y1,x2,y2]=curves(r,a,b,theta)
% Circle with radius A
x1=r*cos(theta); y1=r*sin(theta);
% Ellipse with major and minor axes (a,b)
x2=a*cos(theta); y2=b*sin(theta);
```

```
» r=2; a=3; b=1; theta=0:0.02*pi:2*pi;
» [x1,y1,x2,y2]=curves(2,3,1,theta);
» H=plot(x1,y1,'r',x2,y2,'b');
» set(H,'LineWidth',3.0); axis equal;
```



Newton-Raphson Method

- **One nonlinear equation**

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

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

- **Two nonlinear equations (Taylor-series)**

$$\begin{cases} f_{1,i+1} = f_{1,i} + (x_{1,i+1} - x_{1,i}) \dfrac{\partial f_{1,i}}{\partial x_1} + (x_{2,i+1} - x_{2,i}) \dfrac{\partial f_{1,i}}{\partial x_2} \\ f_{2,i+1} = f_{2,i} + (x_{1,i+1} - x_{1,i}) \dfrac{\partial f_{2,i}}{\partial x_1} + (x_{2,i+1} - x_{2,i}) \dfrac{\partial f_{2,i}}{\partial x_2} \end{cases}$$

Intersection of Two Curves

- Two roots: $f_1(x_1,x_2) = 0$, $f_2(x_1,x_2) = 0$

$$\begin{cases} f_{1,i+1} = 0 \\ f_{2,i+1} = 0 \end{cases} \Rightarrow \begin{cases} \dfrac{\partial f_{1,i}}{\partial x_1} x_{1,i+1} + \dfrac{\partial f_{1,i}}{\partial x_2} x_{2,i+1} = -f_{1,i} + x_{1,i} \dfrac{\partial f_{1,i}}{\partial x_1} + x_{2,i} \dfrac{\partial f_{1,i}}{\partial x_2} \\ \dfrac{\partial f_{2,i}}{\partial x_1} x_{1,i+1} + \dfrac{\partial f_{2,i}}{\partial x_2} x_{2,i+1} = -f_{2,i} + x_{1,i} \dfrac{\partial f_{2,i}}{\partial x_1} + x_{2,i} \dfrac{\partial f_{2,i}}{\partial x_2} \end{cases}$$

- Alternatively

$$\begin{bmatrix} \dfrac{\partial f_{1,i}}{\partial x_1} & \dfrac{\partial f_{1,i}}{\partial x_2} \\ \dfrac{\partial f_{2,i}}{\partial x_1} & \dfrac{\partial f_{2,i}}{\partial x_2} \end{bmatrix} \begin{Bmatrix} x_{1,i+1} - x_{1,i} \\ x_{2,i+1} - x_{2,i} \end{Bmatrix} = -\begin{Bmatrix} f_{1,i} \\ f_{2.i} \end{Bmatrix}$$

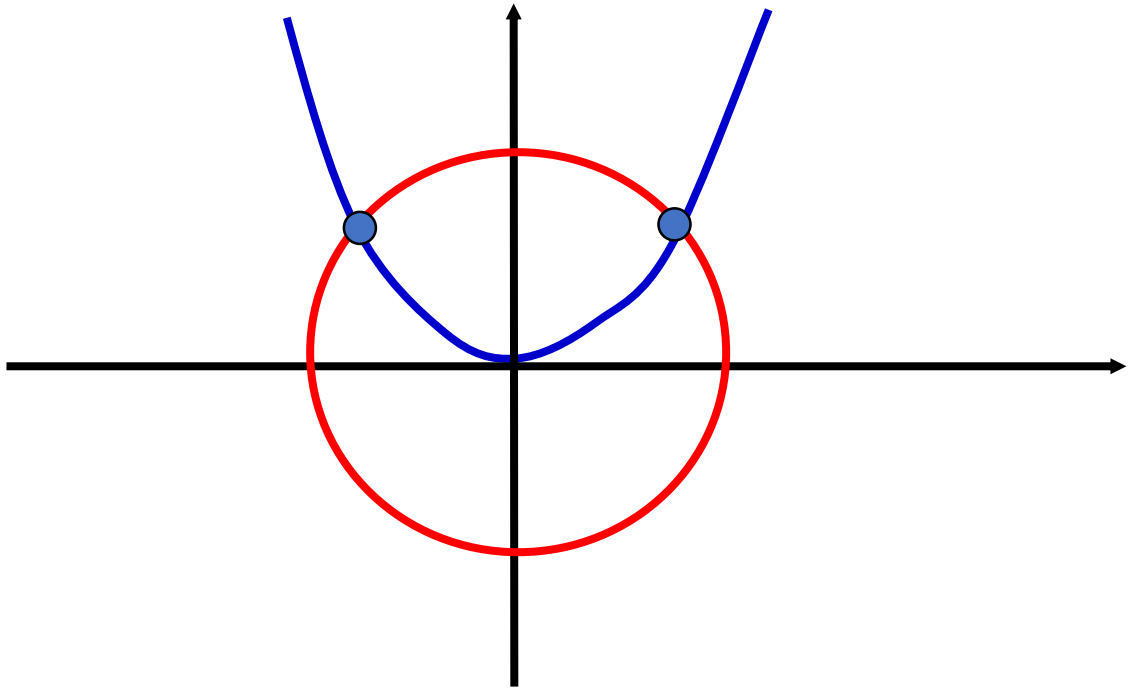
$$[J]\{\Delta x\} = -\{ f \}$$

Jacobian matrix

Intersection of Two Curves

- **Intersection of a circle and a parabola**

$$\begin{cases} f(x,y) = x^2 + y^2 - 1 = 0 \\ g(x,y) = x^2 - y = 0 \end{cases} \quad or \quad \begin{cases} f_1(x_1,x_2) = x_1^2 + x_2^2 - 1 = 0 \\ f_2(x_1,x_2) = x_1^2 - x_2 = 0 \end{cases}$$



Intersection of Two Curves

$$\begin{cases} f_1(x_1, x_2) = x_1^2 + x_2^2 - 1 \\ f_2(x_1, x_2) = x_1^2 - x_2 \end{cases} \qquad \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} \end{bmatrix} = \begin{bmatrix} 2x_1 & 2x_2 \\ 2x_1 & -1 \end{bmatrix}$$

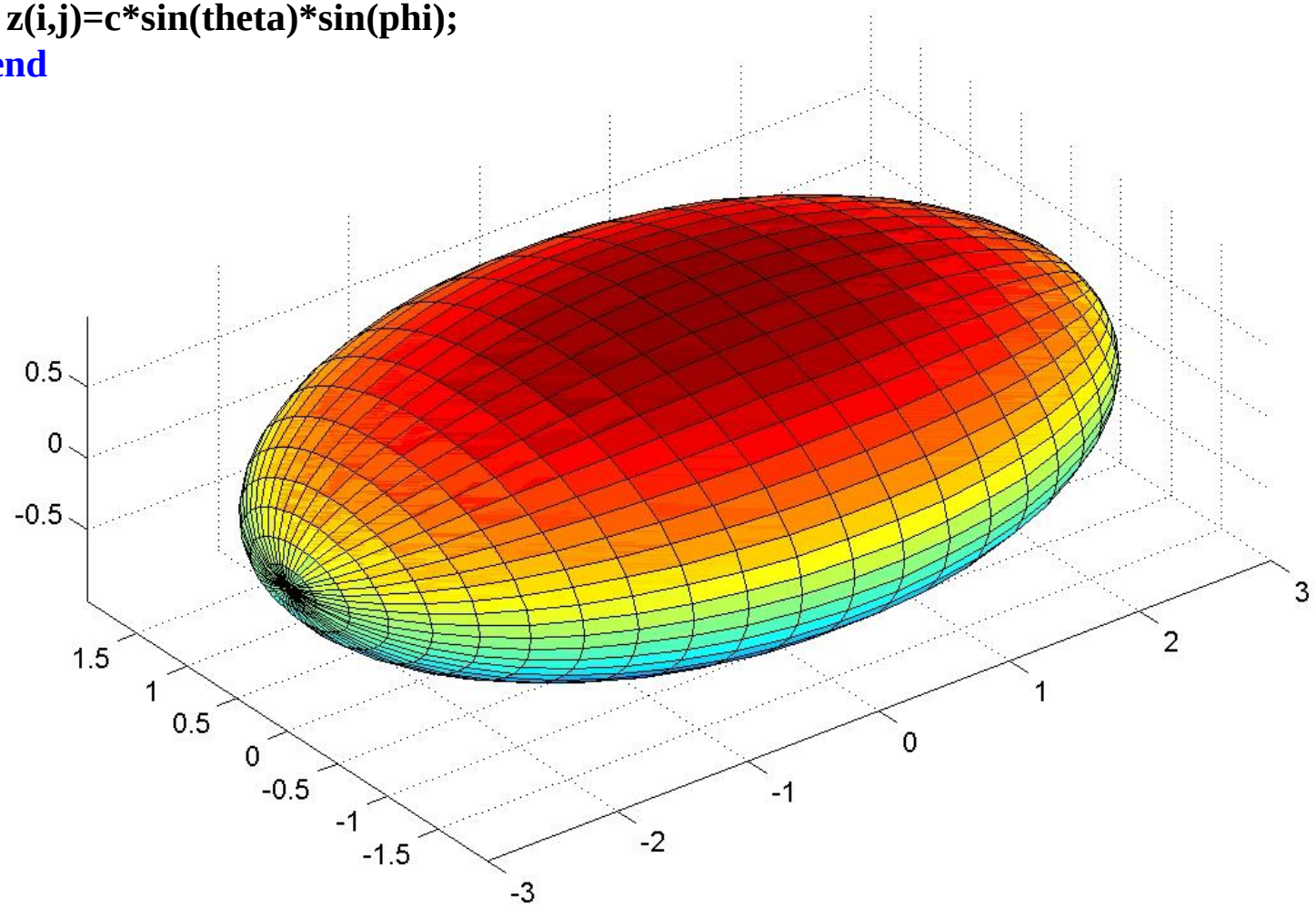
$$[J]\{\Delta x\} = \begin{bmatrix} 2x_{1,i} & 2x_{2,i} \\ 2x_{1,i} & -1 \end{bmatrix} \begin{Bmatrix} \Delta x_{1,i} \\ \Delta x_{2,i} \end{Bmatrix} = \begin{Bmatrix} -f_{1,i} \\ -f_{2,i} \end{Bmatrix}$$

Solve for x^{new}

$$\{\Delta x\} = \begin{Bmatrix} \Delta x_{1,i} \\ \Delta x_{2,i} \end{Bmatrix} = x^{new} - x^{old} = \begin{Bmatrix} x_{1,i+1} - x_{1,i} \\ y_{2,i+1} - x_{2,i} \end{Bmatrix}$$

```
function [x,y,z]=ellipse(a,b,c,imax,jmax)
for i=1:imax+1
   theta=2*pi*(i-1)/imax;
    for j=1:jmax+1
      phi=2*pi*(j-1)/jmax;
      x(i,j)=a*cos(theta);
      y(i,j)=b*sin(theta)*cos(phi);
      z(i,j)=c*sin(theta)*sin(phi);
    end
end
```

```
» a=3; b=2; c=1; imax=50; jmax=50;
» [x,y,z]=ellipse(a,b,c,imax,jmax);
» surf(x,y,z); axis equal;
» print -djpeg100 ellipse.jpg
```

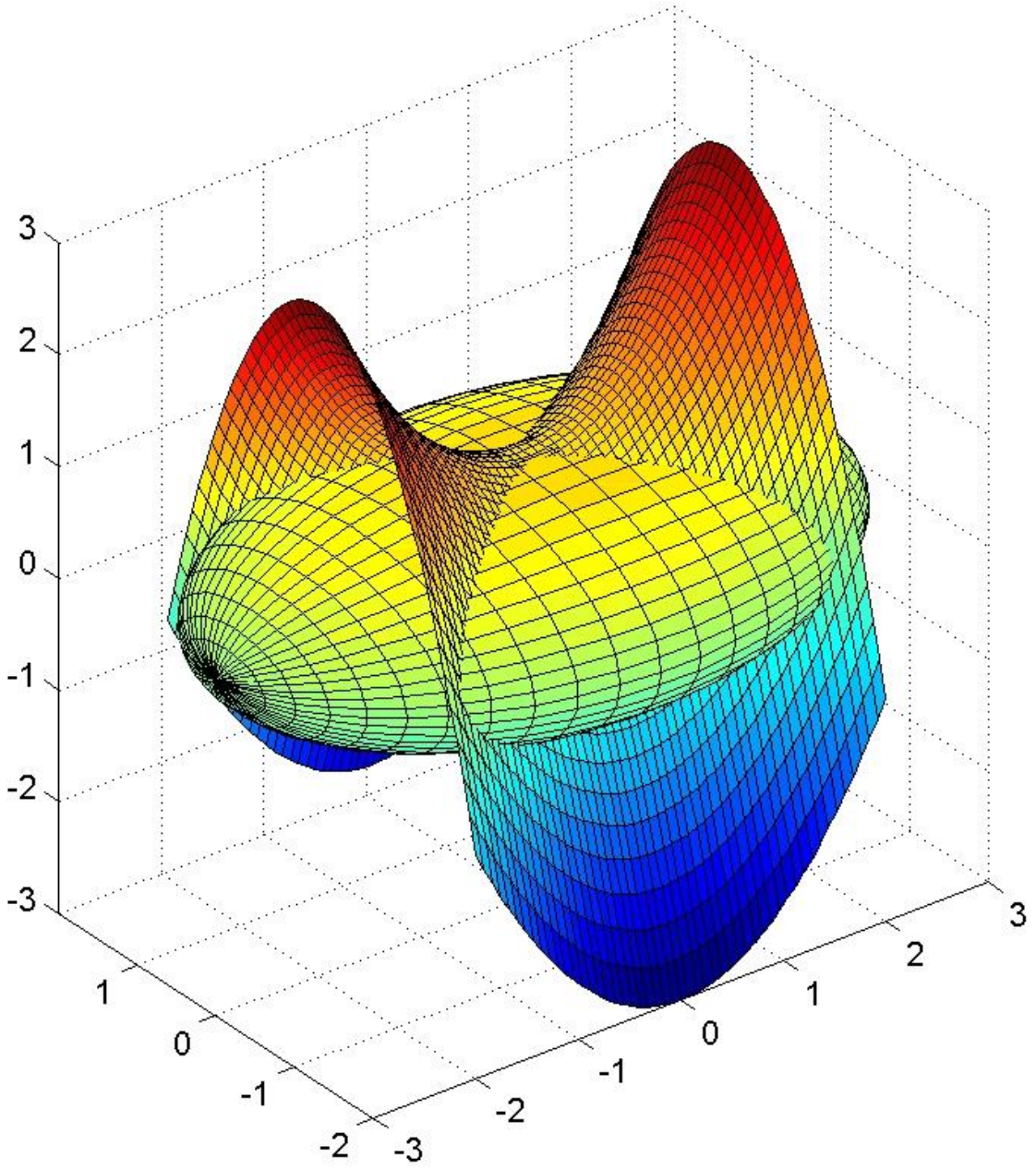


```
function
[x,y,z]=parabola(a,b,c,imax,jmax)
% z=a*x^2+b*y^2+c
for i=1:imax+1
    xi=-2+4*(i-1)/imax;
     for j=1:jmax
       eta=-2+4*(j-1)/jmax;
       x(i,j)=xi;
       y(i,j)=eta;

z(i,j)=a*x(i,j)^2+b*y(i,j)^2+c;
     end
end
```

Parabolic or Hyperbolic Surfaces

```
» a=0.5; b=-1; c=1; imax=50; jmax=50;

» [x2,y2,z2]=parabola(a,b,c,imax,jmax);

» surf(x2,y2,z2); axis equal;

» a=3; b=2; c=1; imax=50; jmax=50;

» [x1,y1,z1]=ellipse(a,b,c,imax,jmax);

» surf(x1,y1,z1); hold on; surf(x2,y2,z2); axis equal;

» print -djpeg100 parabola.jpg
```

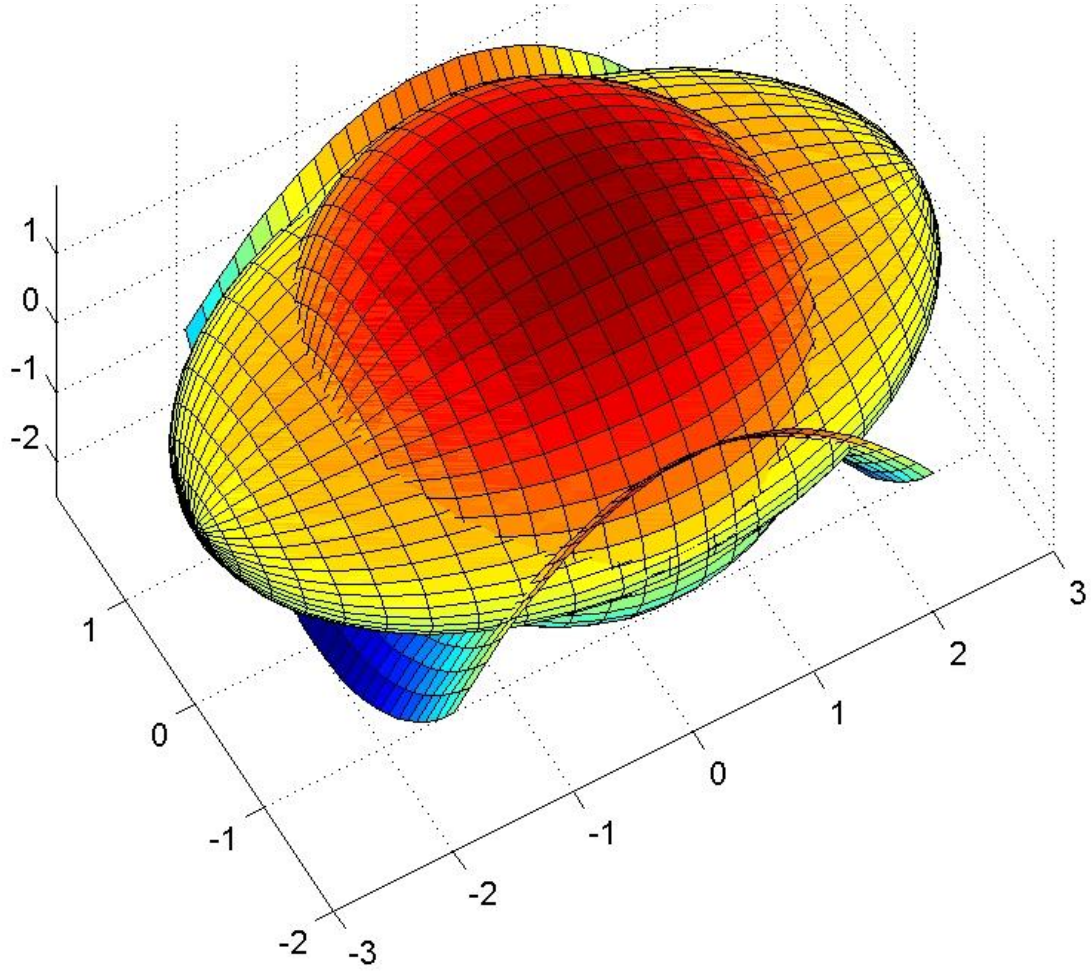
Intersection of two nonlinear surfaces



Infinite many solutions

Intersection of three nonlinear surfaces

```
» [x1,y1,z1]=ellipse(2,2,2,50,50);
» [x2,y2,z2]=ellipse(3,2,1,50,50);
» [x3,y3,z3]=parabola(-0.5,0.5,-0.5,30,30);
» surf(x1,y1,z1); hold on; surf(x2,y2,z2); surf(x3,y3,z3);
» axis equal; view (-30,60);
» print -djpeg100 surf3.jpg
```



Newton-Raphson Method

- *n nonlinear equations in n unknowns*

$$\begin{cases} f_1(x_1, x_2, x_3, \ldots, x_n) = 0 \\ f_2(x_1, x_2, x_3, \ldots, x_n) = 0 \\ f_3(x_1, x_2, x_3, \ldots, x_n) = 0 \\ \vdots \\ f_n(x_1, x_2, x_3, \ldots, x_n) = 0 \end{cases} \qquad \vec{F}(\vec{x}) = \begin{Bmatrix} f_1(\vec{x}) \\ f_2(\vec{x}) \\ f_3(\vec{x}) \\ \vdots \\ f_n(\vec{x}) \end{Bmatrix}$$

$$\vec{F}(\vec{x}) = \vec{0}\,, \quad \vec{x} = \begin{bmatrix} x_1 & x_2 & x_3 & \cdots & x_n \end{bmatrix}$$

Newton-Raphson Method

- **Jacobian (matrix of partial derivatives)**

$$J(\vec{x}) = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \frac{\partial f_1}{\partial x_3} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \frac{\partial f_2}{\partial x_3} & \cdots & \frac{\partial f_2}{\partial x_n} \\ \frac{\partial f_3}{\partial x_1} & \frac{\partial f_3}{\partial x_2} & \frac{\partial f_3}{\partial x_3} & \cdots & \frac{\partial f_3}{\partial x_n} \\ \vdots & \vdots & \vdots & \ddots & \\ \frac{\partial f_n}{\partial x_1} & \frac{\partial f_n}{\partial x_2} & \frac{\partial f_n}{\partial x_3} & \cdots & \frac{\partial f_n}{\partial x_n} \end{bmatrix}$$

- **Newton's iteration (without inversion)**

$$J(\vec{x}_{old})\vec{y} = -\vec{F}(\vec{x}_{old}); \quad \vec{y} = \Delta\vec{x} = \vec{x}_{new} - \vec{x}_{old}$$

Newton-Raphson Method

- For a single equation with one variable

$$J(x) = \frac{df}{dx} = f'(x)$$

- Newton's iteration

$$x_{new} = x_{old} - J^{-1}(\vec{x}_{old})\, f = x_{old} - \frac{f(x_{old})}{f'(x_{old})}$$

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

```
function x = Newton_sys(F, JF, x0, tol, maxit)

% Solve the nonlinear system F(x) = 0 using Newton's method
% Vectors x and x0 are row vectors (for display purposes)
% function F returns a column vector, [f1(x), ..fn(x)]'
% stop if norm of change in solution vector is less than tol
% solve JF(x) y = - F(x) using Matlab's "backslash operator"
% y = -feval(JF, xold) \ feval(F, xold);
% the next approximate solution is x_new = xold + y';

xold = x0;
disp([0 xold ]);
iter = 1;
while (iter <= maxit)
     y= - feval(JF, xold) \ feval(F, xold);
     xnew = xold + y';
     dif = norm(xnew - xold);
     disp([iter    xnew    dif]);
     if dif <= tol
        x = xnew;
        disp('Newton method has converged')
        return;
     else
        xold = xnew;
     end
     iter = iter + 1;
end
disp('Newton method did not converge')
x=xnew;
```

Intersection of Circle and Parabola

```
function f = ex11_1(x)

f = [(x(1)^2 + x(2)^2 -1)
     (x(1)^2 - x(2)) ];
```

```
function df = ex11_1_j(x)

df = [2*x(1)   2*x(2)
      2*x(1)    -1    ]  ;
```

Use x0 = [0.5 0.5] as initial estimates

```
» x0=[0.5 0.5]; tol=1.e-6; maxit=100;
» x=Newton_sys('ex11_1','ex11_1_j',x0,tol,maxit);

         0    0.5000    0.5000
    1.0000    0.8750    0.6250    0.3953
    2.0000    0.7907    0.6181    0.0846
    3.0000    0.7862    0.6180    0.0045
    4.0000    0.7862    0.6180    0.0000
    5.0000    0.7862    0.6180    0.0000
Newton method has converged
```

Intersection of Three Surfaces

- **Solve the nonlinear system**

$$\begin{cases} f(x,y,z) = x^2 + y^2 + z^2 - 14 = 0 \\ g(x,y,z) = x^2 + 2y^2 - 9 = 0 \\ h(x,y,z) = x - 3y^2 + z^2 = 0 \end{cases} \quad or \quad \begin{cases} f_1(x_1,x_2,x_3) = x_1^2 + x_2^2 + x_3^2 - 14 = 0 \\ f_2(x_1,x_2,x_3) = x_1^2 + 2x_2^2 - 9 = 0 \\ f_3(x_1,x_2,x_3) = x_1 - 3x_2^2 + x_3^2 = 0 \end{cases}$$

- **Jacobian**

$$[J] = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \frac{\partial f_1}{\partial x_3} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \frac{\partial f_2}{\partial x_3} \\ \frac{\partial f_3}{\partial x_1} & \frac{\partial f_3}{\partial x_2} & \frac{\partial f_3}{\partial x_3} \end{bmatrix} = \begin{bmatrix} 2x_1 & 2x_2 & 2x_3 \\ 2x_1 & 4x_2 & 0 \\ 1 & -6x_2 & 2x_3 \end{bmatrix}$$

Newton-Raphson Method

- Solve the nonlinear system

$$\begin{bmatrix} 2x_1 & 2x_2 & 2x_3 \\ 2x_1 & 4x_2 & 0 \\ 1 & -6x_2 & 2x_3 \end{bmatrix} \begin{Bmatrix} \Delta x_1 \\ \Delta x_2 \\ \Delta x_3 \end{Bmatrix} = -\begin{Bmatrix} f_1 \\ f_2 \\ f_3 \end{Bmatrix}$$

$$\begin{Bmatrix} x_{new} \\ y_{new} \\ z_{new} \end{Bmatrix} = \begin{Bmatrix} x_{old} \\ y_{old} \\ z_{old} \end{Bmatrix} + \begin{Bmatrix} \Delta x_1 \\ \Delta x_2 \\ \Delta x_3 \end{Bmatrix}$$

- MATLAB function ($y = J^{-1}F$)

```
y = -feval(JF,xold) \ feval(F, xold)
```

Intersection of Three Surfaces

```
function f = ex11_2(x)
% Intersection of three surfaces

f = [ (x(1)^2 + x(2)^2 + x(3)^2 - 14)
      (x(1)^2 + 2*x(2)^2 - 9)
      (x(1) - 3*x(2)^2 + x(3)^2) ];
```

```
function df = ex11_2_j(x)
% Jacobian matrix
df = [ 2*x(1)   2*x(2)   2*x(3)
       2*x(1)   4*x(2)     0
         1     -6*x(2)   2*x(3) ] ;
```

```
» x0=[1 1 1]; es=1.e-6; maxit=100;
» x=Newton_sys('ex11_2','ex11_2_j',x0,es,maxit);
     0      1      1      1
    1.0000    1.6667    2.1667    4.6667    3.9051
    2.0000    1.5641    1.8407    3.2207    1.4858
    3.0000    1.5616    1.8115    2.8959    0.3261
    4.0000    1.5616    1.8113    2.8777    0.0182
    5.0000    1.5616    1.8113    2.8776    0.0001
    6.0000    1.5616    1.8113    2.8776    0.0000
Newton method has converged
```

Fixed-Point Iteration

- No need to compute partial derivatives

$$\begin{cases} f_1(x_1, x_2, x_3, \ldots, x_n) = 0 \\ f_2(x_1, x_2, x_3, \ldots, x_n) = 0 \\ f_3(x_1, x_2, x_3, \ldots, x_n) = 0 \\ \vdots \\ f_n(x_1, x_2, x_3, \ldots, x_n) = 0 \end{cases} \Rightarrow \begin{cases} x_1 = g_1(x_1, x_2, x_3, \ldots, x_n) \\ x_2 = g_2(x_1, x_2, x_3, \ldots, x_n) \\ x_3 = g_3(x_1, x_2, x_3, \ldots, x_n) \\ \vdots \\ x_n = g_n(x_1, x_2, x_3, \ldots, x_n) \end{cases}$$

Convergence Theorem

$$\left|\frac{\partial g_i}{\partial x_j}\right| \leq \frac{K}{n}; \quad K < 1$$

Example1: Fixed-Point

- **Solve the nonlinear system**

$$\begin{cases} f_1(x_1, x_2, x_3) = x_1^2 + 50x_1 + x_2^2 + x_3^2 - 200 = 0 \\ f_2(x_1, x_2, x_3) = x_1^2 + 20x_2 + x_3^2 - 50 = 0 \\ f_3(x_1, x_2, x_3) = -x_1^2 - x_2^2 + 40x_3 + 75 = 0 \end{cases}$$

- **Rearrange (initial guess: *x = y = z = 2*)**

$$\begin{cases} x_1 = g_1(x_1, x_2, x_3) = (200 - x_1^2 - x_2^2 - x_3^2)/50 \\ x_2 = g_2(x_1, x_2, x_3) = (50 - x_1^2 - x_3^2)/20 \\ x_3 = g_3(x_1, x_2, x_3) = (x_1^2 + x_2^2 - 75)/40 \end{cases}$$

```
function x = fixed_pt_sys(G, x0, es, maxit)
% Solve the nonlinear system x = G(x) using fixed-point method
% Vectors x and x0 are row vectors (for display purposes)
% function G returns a column vector, [g1(x), ..gn(x)]'
% stop if norm of change in solution vector is less than es
% y = feval(G,xold); the next approximate solution is xnew = y';

disp([0    x0 ]);       %display initial estimate
xold = x0;
iter = 1;
while (iter <= maxit)
     y = feval(G, xold);
     xnew = y';
     dif = norm(xnew - xold);
     disp([iter    xnew    dif]);
     if dif <= es
        x = xnew;
        disp('Fixed-point iteration has converged')
        return;
     else
        xold = xnew;
     end
     iter = iter + 1;
end
disp('Fixed-point iteration did not converge')
x=xnew;
```

```
function G = example1(x)
G = [ (-0.02*x(1)^2 - 0.02*x(2)^2 - 0.02*x(3)^2 + 4)
      (-0.05*x(1)^2 - 0.05*x(3)^2 + 2.5)
      (0.025*x(1)^2 + 0.025*x(2)^2 -1.875) ];
```

```
» x0=[0 0 0]; es=1.e-6; maxit=100;
» x=fixed_pt_sys('example1', x0, es, maxit);

     0      0      0      0
    1.0000    4.0000    2.5000   -1.8750    5.0760
    2.0000    3.4847    1.5242   -1.3188    1.2358
    3.0000    3.6759    1.8059   -1.5133    0.3921
    4.0000    3.6187    1.7099   -1.4557    0.1257
    5.0000    3.6372    1.7393   -1.4745    0.0395
    6.0000    3.6314    1.7298   -1.4686    0.0126
    7.0000    3.6333    1.7328   -1.4705    0.0040
    8.0000    3.6327    1.7318   -1.4699    0.0013
    9.0000    3.6329    1.7321   -1.4701    0.0004
   10.0000    3.6328    1.7321   -1.4700    0.0001
   11.0000    3.6328    1.7321   -1.4701    0.0000
   12.0000    3.6328    1.7321   -1.4701    0.0000
   13.0000    3.6328    1.7321   -1.4701    0.0000
   14.0000    3.6328    1.7321   -1.4701    0.0000
   15.0000    3.6328    1.7321   -1.4701    0.0000

Fixed-point iteration has converged
```

Example 2: Fixed-Point

- **Solve the nonlinear system**

$$\begin{cases} f_1(x_1, x_2, x_3) = x_1^2 + 4x_2^2 + 9x_3^2 - 36 = 0 \\ f_2(x_1, x_2, x_3) = x_1^2 + 9x_2^2 - 47 = 0 \\ f_3(x_1, x_2, x_3) = x_1^2 x_3 - 11 = 0 \end{cases}$$

- **Rearrange (initial guess: *x = 0, y = 0, z > 0*)**

$$\begin{cases} x_1 = g_1(x_1, x_2, x_3) = \sqrt{11 / x_3} \\ x_2 = g_2(x_1, x_2, x_3) = \sqrt{47 - x_1^2} \,/\, 3 \\ x_3 = g_3(x_1, x_2, x_3) = \sqrt{36 - x_1^2 - 4x_2^2} \,/\, 3 \end{cases}$$

```
function G = example2(x)
G = [ (sqrt(11/x(3)))
      (sqrt(47 - x(1)^2) / 3 )
      (sqrt(36 - x(1)^2 - 4*x(2)^2) / 3 ) ];
```

```
» x0=[0 0 10]; es=0.0001; maxit=500;
» x=fixed_pt_sys('example2',x0,es,maxit);

     0     0     0    10
    1.0000    1.0488    2.2852    2.0000    8.3858
    2.0000    2.3452    2.2583    1.2477    1.4991
    3.0000    2.9692    2.1473    1.0593    0.6612
    4.0000    3.2224    2.0598    0.9854    0.2779
    5.0000    3.3411    2.0170    0.9801    0.1263
    6.0000    3.3501    1.9955    0.9754    0.0238
    7.0000    3.3581    1.9938    0.9916    0.0181
    8.0000    3.3307    1.9923    0.9901    0.0275
    9.0000    3.3332    1.9974    1.0017    0.0129
   10.0000    3.3139    1.9969    0.9962    0.0201
   11.0000    3.3230    2.0005    1.0037    0.0124
   12.0000    3.3105    1.9988    0.9972    0.0142
   13.0000    3.3213    2.0011    1.0033    0.0126
   14.0000    3.3112    1.9991    0.9973    0.0120
   15.0000    3.3212    2.0010    1.0028    0.0116
   16.0000    3.3120    1.9992    0.9974    0.0107
   17.0000    3.3209    2.0008    1.0024    0.0103
   18.0000    3.3126    1.9992    0.9977    0.0097
   19.0000    3.3205    2.0007    1.0022    0.0092
   20.0000    3.3130    1.9993    0.9979    0.0087
    ...
   50.0000    3.3159    1.9999    0.9996    0.0017
    ...
  102.0000    3.3166    2.0000    1.0000    0.0001
  103.0000    3.3167    2.0000    1.0000    0.0001

Fixed-point iteration has converged
```