

Engineering Analysis ENG 3420 Fall 2009

Dan C. Marinescu

Office: HEC 439 B

Office hours: Tu-Th 11:00-12:00

Lecture 19

- Last time:
 - Midterm: solutions and discussions Today:
- Today
 - The inverse of a matrix
 - Iterative methods for solving systems of linear equations
 - Gauss-Siedel
 - Jacobi
- Next Time
 - Relaxation
 - Non-linear systems

The inverse of a square

- If $[A]$ is a square matrix, there is another matrix $[A]^{-1}$, called the inverse of $[A]$, for which $[A][A]^{-1}=[A]^{-1}[A]=[I]$
- The inverse can be computed in a column by column fashion by generating solutions with unit vectors as the right-hand-side constants:

$$[A]\{x_1\} = \begin{Bmatrix} 1 \\ 0 \\ 0 \end{Bmatrix} \quad [A]\{x_2\} = \begin{Bmatrix} 0 \\ 1 \\ 0 \end{Bmatrix} \quad [A]\{x_3\} = \begin{Bmatrix} 0 \\ 0 \\ 1 \end{Bmatrix}$$

$$[A]^{-1} = [x_1 \quad x_2 \quad x_3]$$

Canonical base of an n-dimensional vector space

100.....000

010.....000

001.....000

.....

000.....100

000.....010

000.....001

The response of a linear system

- The response of a linear system to some stimuli can be found using the matrix inverse.

$$[\text{Interactions}]\{\text{response}\} = \{\text{stimuli}\}$$

$$Ar = s$$

$$A^{-1}Ar = A^{-1}s$$

$$A^{-1}A = I$$

$$r = A^{-1}s$$

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 $j-1^{\text{th}}$ iteration using:

$$x_1^j = \frac{b_1 - a_{12}x_2^{j-1} - a_{13}x_3^{j-1}}{a_{11}}$$

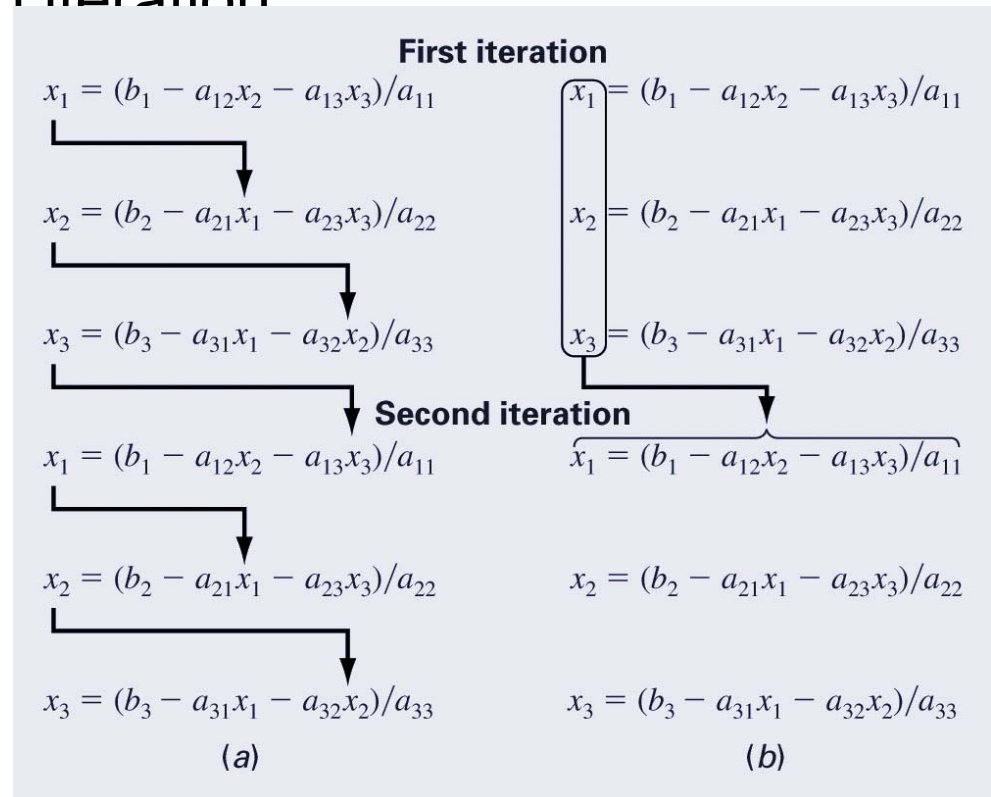
$$x_2^j = \frac{b_2 - a_{21}x_1^j - a_{23}x_3^{j-1}}{a_{22}}$$

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

Jacobi Iteration

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

- Gauss-Seidel
- Jacobi



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,

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

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

Diagonal Dominance

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

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

```

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

```

```

function xnew=Jacobi(A,b,maxerr,maxiter,guess)
    if(nargin <2)
        error('at least two arguments required')
        return
    end
    if(nargin < 3)
        maxerr=0.00001;
    end
    if(nargin < 4)
        maxiter=100;
    end
    [n,m] = size(A);
    if(n~=m)
        error('A is not a square matrix')
        return;
    end
    [k,k1]=size(b)
    if(k~=n)
        error('the dimensions of A and b do not match')
        return;
    end
    if(nargin < 5)
        for i=1:n
            guess(i)=0.0
        end;
    end
    for i=1:n
        sum=0
        for j=1:n
            if (j~=i)
                sum = sum + A(i,j)
            end
        end
        if (A(i,j) < sum )
            error('convergence criteria is not satisfied');
            return
        end
    end
    xnew=zeros(n);
    D=zeros(n,n)
    for i=1:n
        D(i,i)=A(i,i)
    end
    R=zeros(n,n)
    R=A-D
    D1 = inv(D)
    C=D1*b
    E=D1*R
    xoldt = guess'
    for i=1:maxiter
        F=E*xoldt
        xnewt = C-F
        xnew=xnewt'
        xold=xoldt'
        for k=1:n
            relerr=zeros(n);
            relerr(k) = abs((xnew(1,k)-xold(1,k))/xnew(1,k));
        end
        if (max(relerr) < maxerr)
            return;
        end
        xoldt = xnewt
        iteration=i
    end
end

```

Error using ==> Jacobi at 3

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



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*.

Newton-Raphson

- 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:

$f_{1,i+1} = f_{1,i} + (x_{1,i+1} - x_{1,i}) \frac{\mathcal{F}_{1,i}}{\partial x_1} + (x_{2,i+1} - x_{2,i}) \frac{\mathcal{F}_{1,i}}{\partial x_2}$	$x_{1,i+1} = x_{1,i} - \frac{f_{1,i} \frac{\mathcal{F}_{2,i}}{\partial x_2} - f_{2,i} \frac{\mathcal{F}_{1,i}}{\partial x_2}}{\frac{\mathcal{F}_{1,i}}{\partial x_1} \frac{\mathcal{F}_{2,i}}{\partial x_2} - \frac{\mathcal{F}_{1,i}}{\partial x_2} \frac{\mathcal{F}_{2,i}}{\partial x_1}}$
$f_{2,i+1} = f_{2,i} + (x_{1,i+1} - x_{1,i}) \frac{\mathcal{F}_{2,i}}{\partial x_1} + (x_{2,i+1} - x_{2,i}) \frac{\mathcal{F}_{2,i}}{\partial x_2}$	$x_{2,i+1} = x_{2,i} - \frac{f_{2,i} \frac{\mathcal{F}_{1,i}}{\partial x_1} - f_{1,i} \frac{\mathcal{F}_{2,i}}{\partial x_1}}{\frac{\mathcal{F}_{1,i}}{\partial x_1} \frac{\mathcal{F}_{2,i}}{\partial x_2} - \frac{\mathcal{F}_{1,i}}{\partial x_2} \frac{\mathcal{F}_{2,i}}{\partial x_1}}$

```

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

```