Introduction to Applied Scientific Computing using MATLAB

Mohsen Jenadeleh

In this lecture, slides from MIT, Rutgers and Waterloo University are used to form the lecture slides

Matrix Algebra

- dot product
- matrix-vector multiplication
- matrix-matrix multiplication
- matrix inverse
- solving linear systems
- least-squares solutions
- determinant, rank, condition number
- vector & matrix norms
- iterative solutions of linear systems
- examples
- electric circuits
- temperature distributions

Iterative solutions of linear systems **Ax=b**

the only practical way to solve very large linear systems is iteratively

Methods:

1. Jacobi method
2. Gauss-Seidel method
3. Relaxation methods
4. Conjugate Gradient method
5. Others

G. H. Golub and C. F. Van Loan, *Matrix Computations*, 3/e, JHU Press, 1996.
D. S. Watkins, *Fundamentals of Matrix Computations*, 2/e, Wiley, 2002.
L. N. Trefethen and D. Bau, *Numerical Linear Algebra*, SIAM, 1997.
A. Bjork, *Numerical Methods for Least Squares Problems*, SIAM, 1996.

scalar example illustrating the Jacobi method

$$3x = 12$$

rearrange

$$2x + x = 12$$

$$2x = -x + 12$$

rearrange

$$x = -0.5\,x + 6$$

turn it into a recursion

for $k = 1, 2, 3, \ldots$

$$x(k+1) = -0.5\,x(k) + 6$$

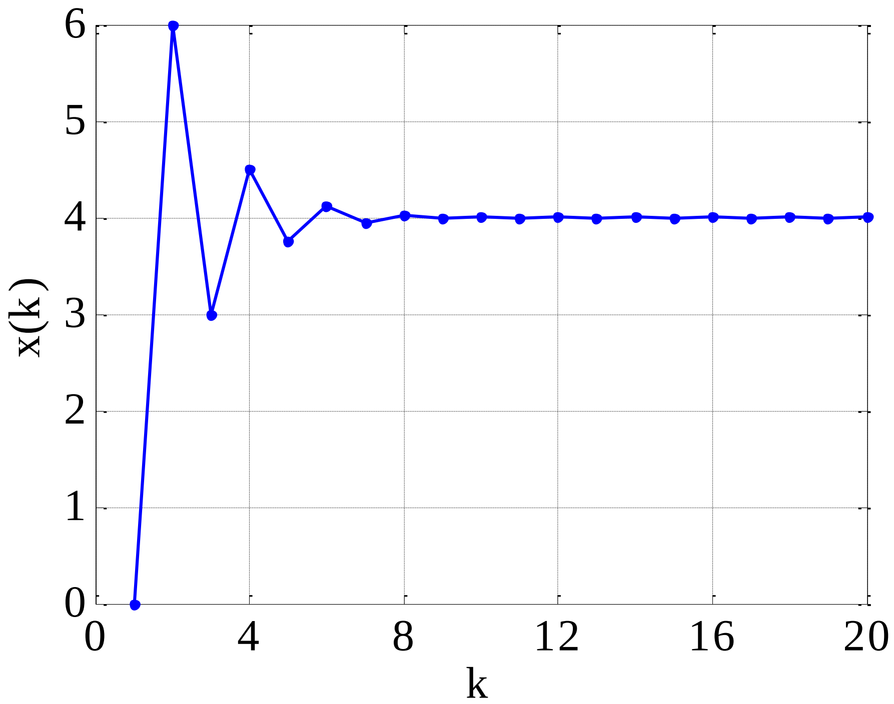
start with any $x(1)$,

$$x(2) = -0.5x(1) + 6$$

$$x(3) = -0.5x(2) + 6$$

$$x(4) = -0.5x(3) + 6, \quad \text{etc.}$$

```
x(1)=0;        % arbitrary
for k=1:19
  x(k+1) = -0.5*x(k) + 6;
end

k=1:20; plot(k,x,'b.-');
```




```
>> [k; x]'
     1   0.0000
     2   6.0000
     3   3.0000
     4   4.5000
     5   3.7500
     6   4.1250
     7   3.9375
     8   4.0313
     9   3.9844
    10   4.0078
    11   3.9961
    12   4.0020
    13   3.9990
    14   4.0005
    15   3.9998
    16   4.0001
    17   3.9999
    18   4.0000
    19   4.0000
    20   4.0000
```

```
tol=1e-10; x0=0;

x=x0; k=1;

while 1
    xnew = -0.5*x + 6;
    if abs(xnew-x)<=tol
        break;
    end
    x = xnew;
    k = k+1;
end

k, abs(x-4)

k =
     37
ans =
   5.8208e-011
```

forever while loop

```
tol=1e-10; x0=0;

x=x0; k=1;
xnew = -0.5*x+6;

while abs(xnew-x)>tol
    x = xnew;
    k = k+1;
    xnew = -0.5*x + 6;
end

k, abs(x-4)

k =
     37
ans =
   5.8208e-011
```

conventional while loop

More general version of $ax=b$, where a,x,b are scalars

choose a splitting $a = d - r$, such that $| r/d | < 1$

$$ax = (d - r)x = b$$

$$dx = rx + b$$

$$x = \frac{r}{d}x + \frac{b}{d}$$

key assumption that guarantees convergence

this is the scalar version of the Jacobi and Gauss-Seidel methods

iterative algorithm:

$$x(k+1) = \frac{r}{d}x(k) + \frac{b}{d}, \quad k = 0, 1, 2, \ldots$$

it has solution that converges to b/a:

$$x(k) = \frac{b}{a} + \left(\frac{r}{d}\right)^k \left(x_0 - \frac{b}{a}\right), \quad k = 0, 1, 2, \ldots$$

Jacobi's method for *A x* = *b*

Define the following 'Jacobi iteration parameters'

$$D = \text{diag}\big(\text{diag}(A)\big) = \text{diagonal part of } A$$

$$B = I - D^{-1}A\,, \quad I = \text{identity matrix}$$

$$\mathbf{c} = D^{-1}\,\mathbf{b}$$

$$A = D - DB$$

$$A\mathbf{x} = \mathbf{b} \quad \Rightarrow \quad D\mathbf{x} - DB\mathbf{x} = \mathbf{b} \quad \Rightarrow \quad D\mathbf{x} = DB\mathbf{x} + \mathbf{b}$$

$$\mathbf{x} = B\mathbf{x} + \mathbf{c}$$

Jacobi iteration:

$$\mathbf{x}(k+1) = B\mathbf{x}(k) + \mathbf{c}\,, \quad k = 0, 1, 2, \ldots$$

Convergence requires the condition that the so-called 'spectral radius' of B be strictly less than unity:

$$\rho(B) = \max\big(\mathrm{abs}(\mathrm{eig}(B))\big) < 1$$

This is hard to calculate for large matrices, but a sufficient condition for convergence is that a matrix norm of B, such as the L_1, L_2, L_∞, or Frobenius norms, be less than unity because of the inequality:

$$\rho(B) \le \|B\|$$

so that

$$\|B\| < 1 \quad \Rightarrow \quad \rho(B) < 1$$

Note: if the method fails for a system **`A,b`**, then, try it on the rearranged system

```
A --> flipud(A)
b --> flipud(b)
```

which sometimes works

Jacobi iteration:

$$\mathbf{x}(k+1) = B\mathbf{x}(k) + \mathbf{c}\,, \quad k = 0, 1, 2, \ldots$$

Exact solution demonstrates the convergence:

$$\mathbf{x}(k) = A^{-1}\mathbf{b} + B^{k}(\mathbf{x}_0 - A^{-1}\mathbf{b})\,, \quad k = 0, 1, 2, \ldots$$

Jacobi iteration:

$$\mathbf{x}(k+1) = B\mathbf{x}(k) + \mathbf{c}\,, \quad k = 0, 1, 2, \ldots$$

Jacobi iteration – alternative form:

$$\mathbf{x}(k+1) = \mathbf{x}(k) + D^{-1}\left[\mathbf{b} - A\mathbf{x}(k)\right], \quad k = 0, 1, 2, \ldots$$

Jacobi iteration – relaxation form:

$$\mathbf{x}(k+1) = \mathbf{x}(k) + \omega\, D^{-1}\left[\mathbf{b} - A\mathbf{x}(k)\right], \quad k = 0, 1, 2, \ldots$$

relaxation parameter

MATLAB implementation

```
% given A = NxN matrix, b = Nx1 vector
% construct the Jacobi parameters

I = eye(size(A));            % identity matrix
D = diag(diag(A));           % diagonal part of A

B = I - D\A;                 % iteration matrix
c = D\b;                     % Nx1 vector

rho = max(abs(eig(B)));      % check if rho < 1
```

Next, implement the Jacobi iteration using a forever while-loop with a stopping condition,

$$\mathbf{x}(k+1) = B\mathbf{x}(k) + \mathbf{c}\,, \quad k = 0, 1, 2, \ldots$$

MATLAB implementation – v.1

```
tol = 1e-10;           % error tolerance, our choice
x = x0; k = 1;         % x0 = arbitrary Nx1 vector

while 1                        % forever loop
   xnew = B*x + c;
   if norm(xnew-x) < tol
      break;
   end
   x = xnew;
   k = k+1;
end

k, x, norm(A*x-b)    % print & verify solution
```

`norm()` measures the difference between the vectors **`xnew`** and **`x`**,

could also use L_∞ norm **`max(abs(xnew-x))`**

Note that it breaks out of the loop just before the tolerance is reached, one more iteration after the loop would meet or exceed the tolerance.

MATLAB implementation – v.2

```
% in order to save individual iterates x(k),
% use a matrix X whose columns are x(k)

tol = 1e-10;           % error tolerance
X(:,1)=x0; k=1;        % x0 = arbitrary Nx1 vector

while 1
   X(:,k+1) = B*X(:,k) + c;
   if norm(X(:,k+1)-X(:,k)) < tol
      break;
   end
   k = k+1;
end

k, x = X(:,k)           % x = converged solution
norm(A*x-b)             % verify solution
```

MATLAB implementation – v.3

```
% another version that saves the iterates

tol=1e-10;      % error tolerance
x=x0; k=1;      % x0 = arbitrary Nx1 vector
X=x;            % X columns are the iterates x(k)

while 1
   xnew = B*x + c;
   if norm(xnew-x) < tol
      break;
   end
   x = xnew;
   k = k+1;
   X = [X,x];          % append new column to X
end

k, x, norm(A*x-b)    % print & verify solution
```

Example

```
A = [2 1 0; 1 5 -1; 1 -2 4];
b = [4 8 9]';
xsol = A\b;

>> A, b, xsol
```

$$\begin{bmatrix} 2 & 1 & 0 \\ 1 & 5 & -1 \\ 1 & -2 & 4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 4 \\ 8 \\ 9 \end{bmatrix}$$

```
A =                 b =        xsol =
    2   1   0           4              1
    1   5  -1           8              2
    1  -2   4           9              3
```

```
I = eye(size(A));          % 3x3 identity matrix
D = diag(diag(A));         % diagonal part of A

B = I - D\A;               % 3x3 iteration matrix
c = D\b;                   % 3x1 vector

rho = max(abs(eig(B)));
```

```
>> D, B

D =
     2   0   0
     0   5   0
     0   0   4

B =
     0.00 -0.50   0.00
    -0.20  0.00   0.20
    -0.25  0.50   0.00
```

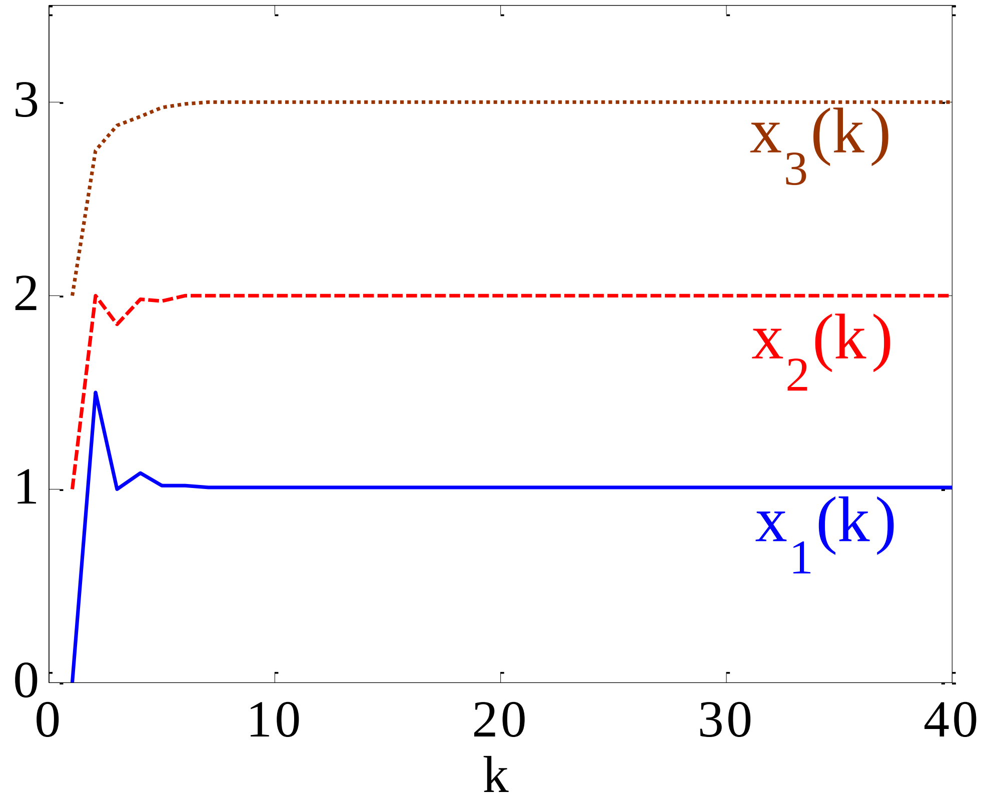
```
>> c, rho, norm(B,inf)

c =
    2.00
    1.60
    2.25

rho =
    0.5000
ans =
    0.7500
```

$$\mathbf{x}(k+1) = B\mathbf{x}(k) + \mathbf{c}\,, \quad k = 0, 1, 2, \ldots$$

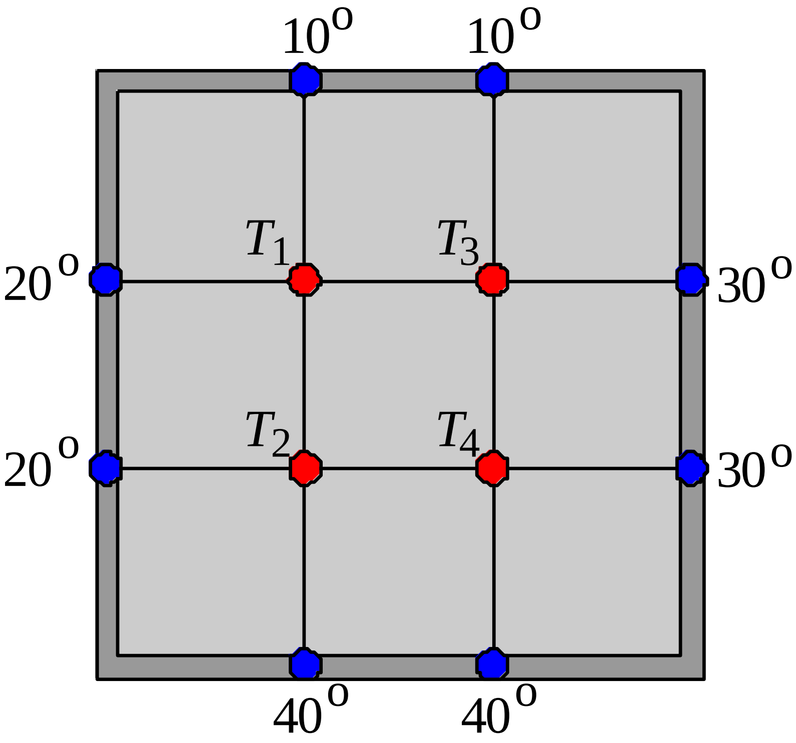
$$\begin{bmatrix} x_1(k+1) \\ x_2(k+1) \\ x_3(k+1) \end{bmatrix} = \begin{bmatrix} 0.00 & -0.50 & 0.00 \\ -0.20 & 0.00 & 0.20 \\ -0.25 & 0.50 & 0.00 \end{bmatrix} \begin{bmatrix} x_1(k) \\ x_2(k) \\ x_3(k) \end{bmatrix} + \begin{bmatrix} 2.00 \\ 1.60 \\ 2.65 \end{bmatrix}$$

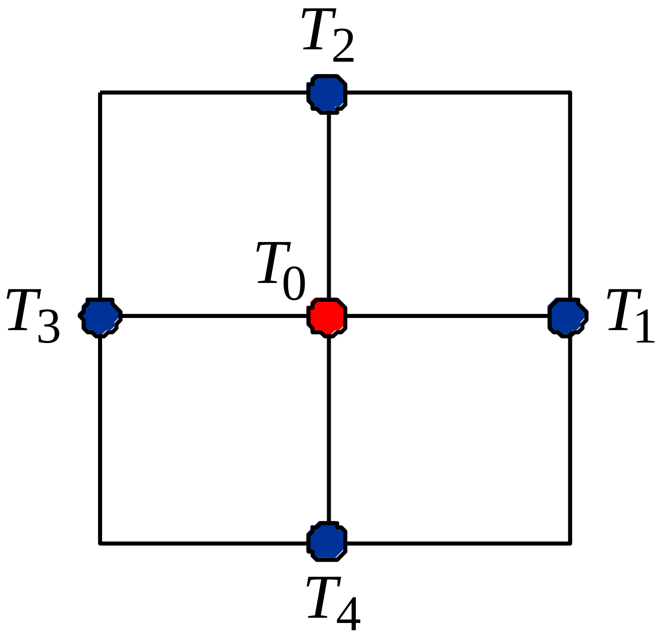
```
tol=1e-12;                 % error tolerance
x=[0 1 2]'; k=1;           % x0 = [0 1 2]' = arbitrary
X=x;                       % X = save all iterates

while 1
   xnew = B*x + c;
   if norm(xnew-x)<tol         % norm() measures
      break;                   % the distance
   end                         % between x,xnew
   x = xnew;
   k = k+1;
   X = [X,x];                  % append new column
end

k, x, norm(A*x-b), norm(x-xsol), size(X)
```

```
>> k, x, norm(A*x-b), norm(x-xsol), size(X)

k =
    40
x =
    1.0000
    2.0000
    3.0000
ans =
  2.6657e-012
ans =
  1.3634e-012
ans =
     3    40

>> K=1:k;
>> plot(K,X');
```



Temperature Distribution

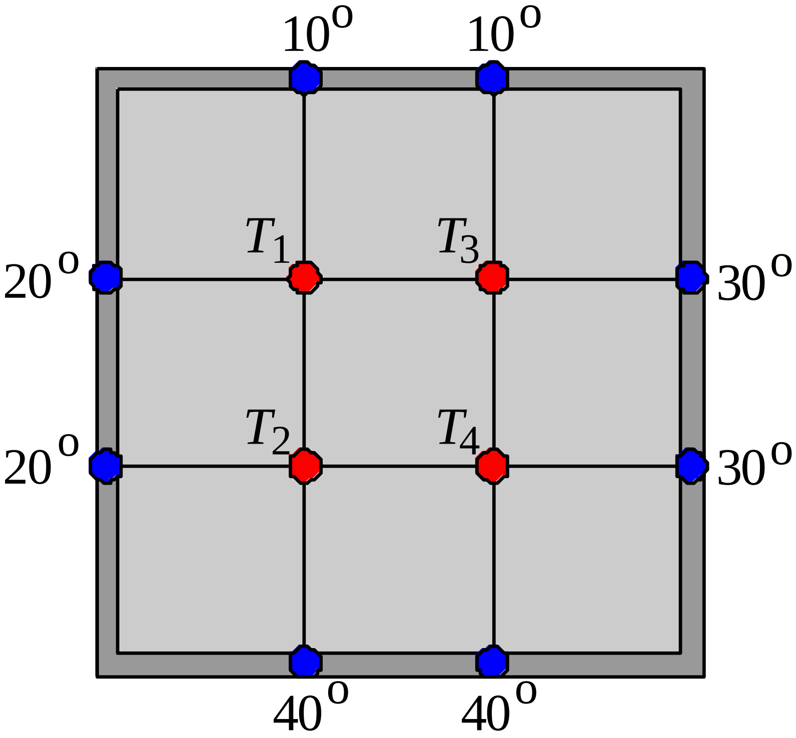




$$T_0 = \frac{1}{4}(T_1 + T_2 + T_3 + T_4)$$

follows from discretizing the Laplace equation

$$\nabla^2 T = \frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} = 0$$



$$T_1 = \frac{1}{4}(10 + 20 + T_2 + T_3)$$

$$T_2 = \frac{1}{4}(20 + 40 + T_1 + T_4)$$

$$T_3 = \frac{1}{4}(10 + 30 + T_1 + T_4)$$

$$T_4 = \frac{1}{4}(30 + 40 + T_2 + T_3)$$

$$4T_1 - T_2 - T_3 = 30$$

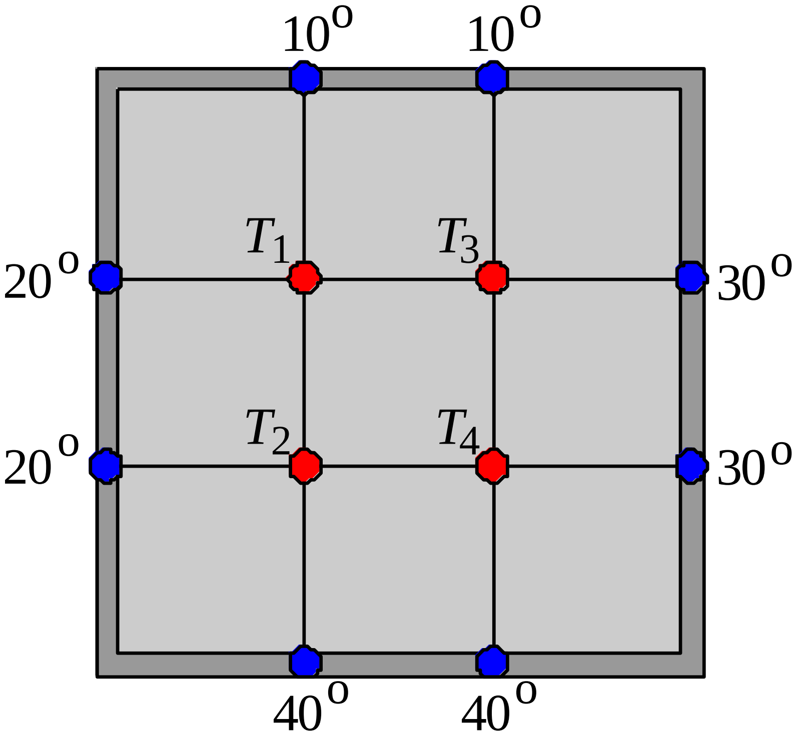
$$4T_2 - T_1 - T_4 = 60$$

$$4T_3 - T_1 - T_4 = 40$$

$$4T_4 - T_2 - T_3 = 70$$

$$\begin{bmatrix} 4 & -1 & -1 & 0 \\ -1 & 4 & 0 & -1 \\ -1 & 0 & 4 & -1 \\ 0 & -1 & -1 & 4 \end{bmatrix} \begin{bmatrix} T_1 \\ T_2 \\ T_3 \\ T_4 \end{bmatrix} = \begin{bmatrix} 30 \\ 60 \\ 40 \\ 70 \end{bmatrix}$$

$$\mathbf{A}\,\mathbf{x} = \mathbf{b}$$

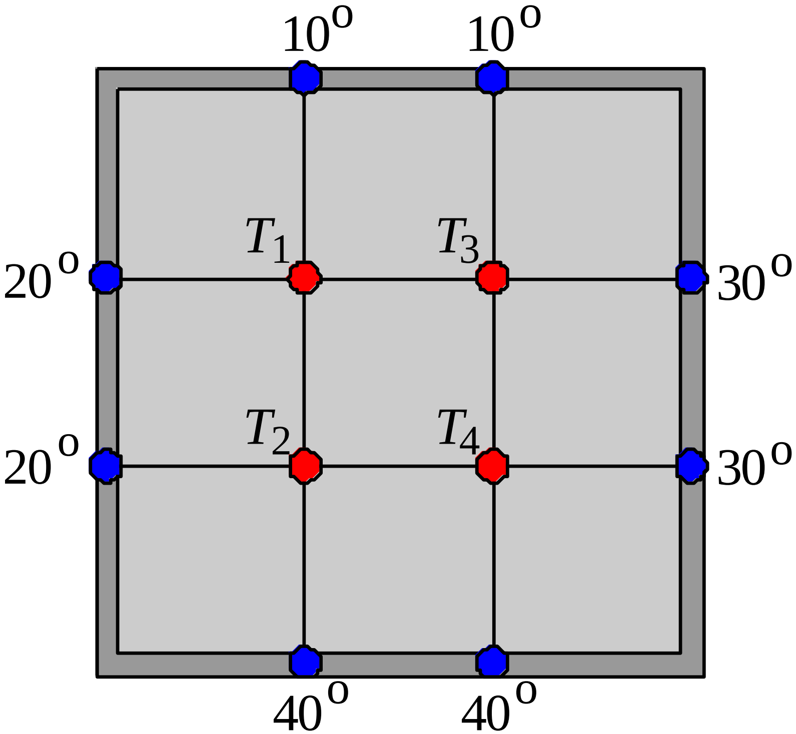


$$\begin{bmatrix} 4 & -1 & -1 & 0 \\ -1 & 4 & 0 & -1 \\ -1 & 0 & 4 & -1 \\ 0 & -1 & -1 & 4 \end{bmatrix} \begin{bmatrix} T_1 \\ T_2 \\ T_3 \\ T_4 \end{bmatrix} = \begin{bmatrix} 30 \\ 60 \\ 40 \\ 70 \end{bmatrix}$$

$$\mathbf{A}\,\mathbf{x} = \mathbf{b}$$

```
>> A = [ 4 -1 -1  0
        -1  4  0 -1
        -1  0  4 -1
         0 -1 -1  4];
>> b = [30; 60; 40; 70];
```

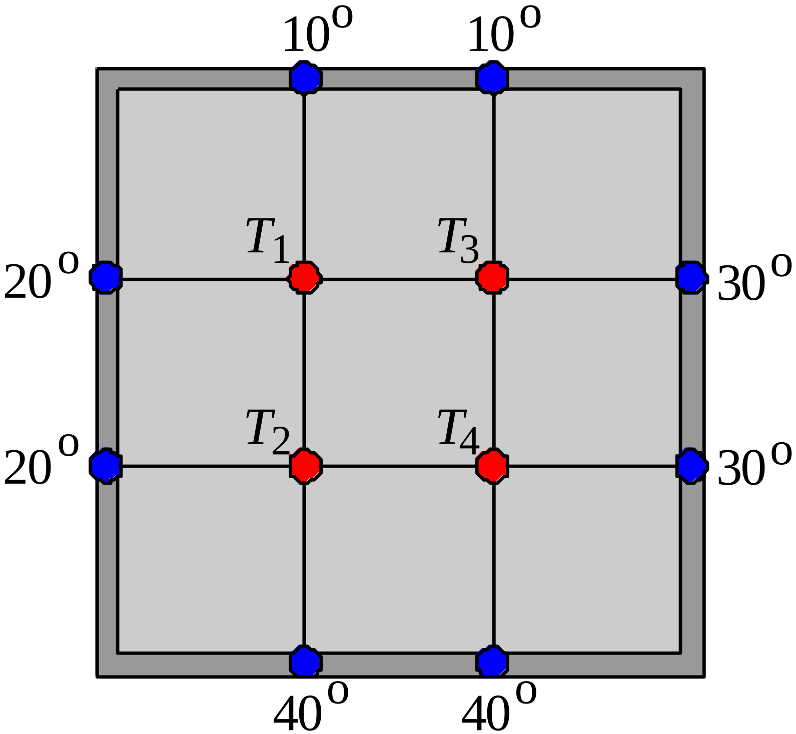
```
>> x = A\b

x =

   20.0000
   27.5000
   22.5000
   30.0000
```



display solution **T** in a rectangular pattern

nodes were numbered in column order

```
T = zeros(2,2);  % shape of T
T(:) = x

T =
    20.0000    22.5000
    27.5000    30.0000
```



$$\begin{bmatrix} 4 & -1 & -1 & 0 \\ -1 & 4 & 0 & -1 \\ -1 & 0 & 4 & -1 \\ 0 & -1 & -1 & 4 \end{bmatrix} \begin{bmatrix} T_1 \\ T_2 \\ T_3 \\ T_4 \end{bmatrix} = \begin{bmatrix} 30 \\ 60 \\ 40 \\ 70 \end{bmatrix}$$

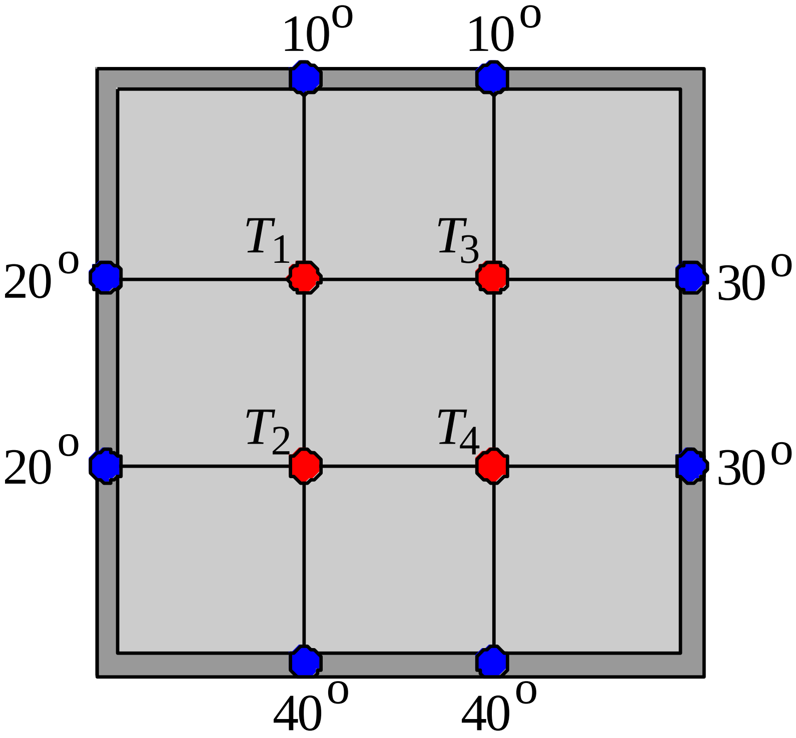
A x = b

Rules for constructing **A** and **b**:

1) main diagonal is 4

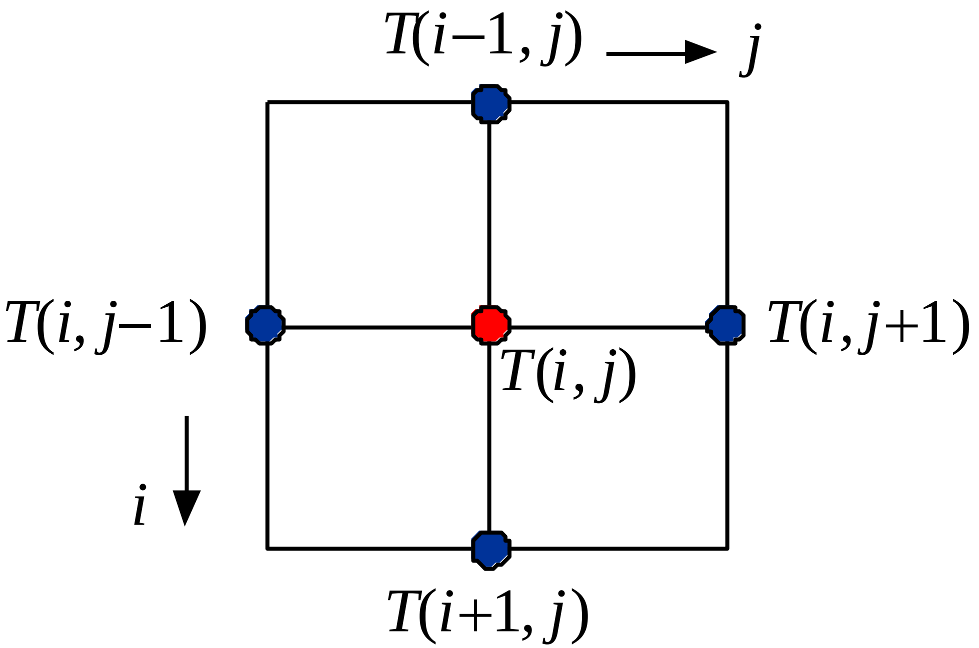
2) if nodes **i,j** are connected, then, -1, otherwise, 0

3) **b(i)** is sum of boundary values connected to node **i**



Iterative Solution

convenient for large number of subdivisions



used also to solve 2D electrostatics problems

$$T(i,j) = \frac{1}{4}\left[T(i+1,j) + T(i-1,j) + T(i,j+1) + T(i,j-1)\right]$$

```
N=4; M=4;
left=20; right=30; up=10; dn=40;
T(1,:) = repmat(up,1,M);   T(N,:) = repmat(dn,1,M);
T(:,1) = repmat(left,N,1); T(:,M) = repmat(right,N,1);

Tnew = T;

tol = 1e-4; K = 100;
for k=1:K,
  for i=2:N-1,
    for j=2:M-1,
      Tnew(i,j) = (T(i-1,j) + T(i+1,j) +...
                   T(i,j-1) + T(i,j+1))/4;
    end
  end
  if norm(Tnew-T) < tol, break; end
  T = Tnew;
end

T(1,[1,end]) = nan; T(end,[1,end]) = nan;
```

boundary values

iterate over internal nodes only

```
T =                                    % start-up
   NaN      10      10    NaN
    20       0       0     30
    20       0       0     30
   NaN      40      40    NaN

% converged after k = 19 iterations
% to within the specified tol = 1e-4

T =
       NaN   10.0000   10.0000       NaN
   20.0000   19.9999   22.4999   30.0000
   20.0000   27.4999   29.9999   30.0000
       NaN   40.0000   40.0000       NaN
```

```
T =             % after k=1 iteration
       NaN   10.0000   10.0000       NaN
   20.0000    7.5000   10.0000   30.0000
   20.0000   15.0000   17.5000   30.0000
       NaN   40.0000   40.0000       NaN

T =             % after k=2 iterations
       NaN   10.0000   10.0000       NaN
   20.0000   13.7500   16.2500   30.0000
   20.0000   21.2500   23.7500   30.0000
       NaN   40.0000   40.0000       NaN

T =             % after k=3 iterations
       NaN   10.0000   10.0000       NaN
   20.0000   16.8750   19.3750   30.0000
   20.0000   24.3750   26.8750   30.0000
       NaN   40.0000   40.0000       NaN
```

```
N=30; M=30;
left=0; right=0; up=0; dn=60;
tol = 1e-6; K = 5000;

% breaks out at k = 2475

[X,Y] = meshgrid(2:M-1, 2:N-1);

Z = T(2:M-1, 2:N-1);
surf(X,Y,Z);
```