ITERATIVE METHODS

Numerical methods in chemical engineering

Ivo Roghair

TU/e
Technische Universiteit
Eindhoven
University of Technology

Where innovation starts

OVERVIEW

- Sparse matrices
- Large systems of linear equations
- Iterative methods for large systems of equations
  - Jacobi method
  - Gauss-Seidel method

Sparse matrices

- In many engineering cases, we deal with sparse matrices (as opposed to dense matrices)
  - A matrix is sparse when it mostly consists of zeros
  - Linear systems where equations depend on a limited number of variables (e.g. spatial discretization)
- Storing zeros is not very efficient:

```
>> A = eye(10000);
>> whos A
>> S = sparse(A);
>> whos S
```

- Can you think of a way to achieve this?
- Sparse matrix formats: Yale, CRS, CCS

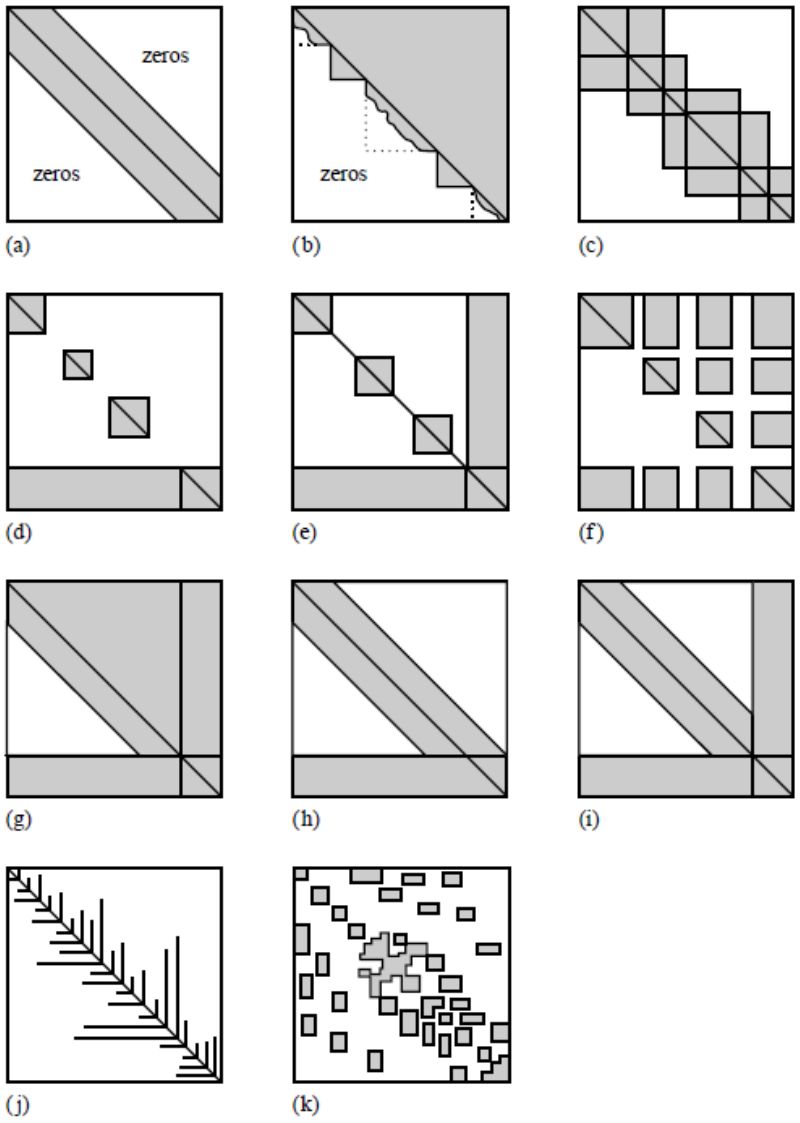
Sparse matrices

- **Example: Yale storage format, storing 3 vectors:**
  - **A = [5 8 3 6]**
  - **IA = [0 0 2 3 4]**
  - **JA = [0 1 2 1]**

$$\begin{bmatrix} 0 & 0 & 0 & 0 \\ 5 & 8 & 0 & 0 \\ 0 & 0 & 3 & 0 \\ 0 & 6 & 0 & 0 \end{bmatrix}$$

- **A stores the nonzero values**
- **IA(i) stores the index in A of the first non-zero in row i**
- **JA stores the column index**
- **Zero-based indices**

Sparse matrices



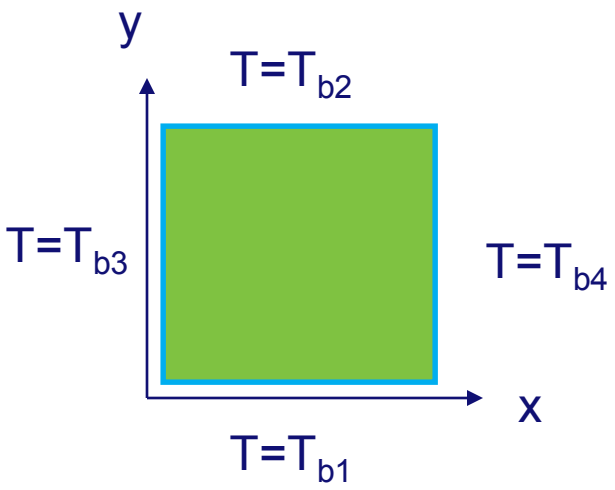
LAPLACE'S EQUATIONS

$$\frac{\partial T}{\partial t} = \alpha \nabla^2 T$$

Thermal diffusivity

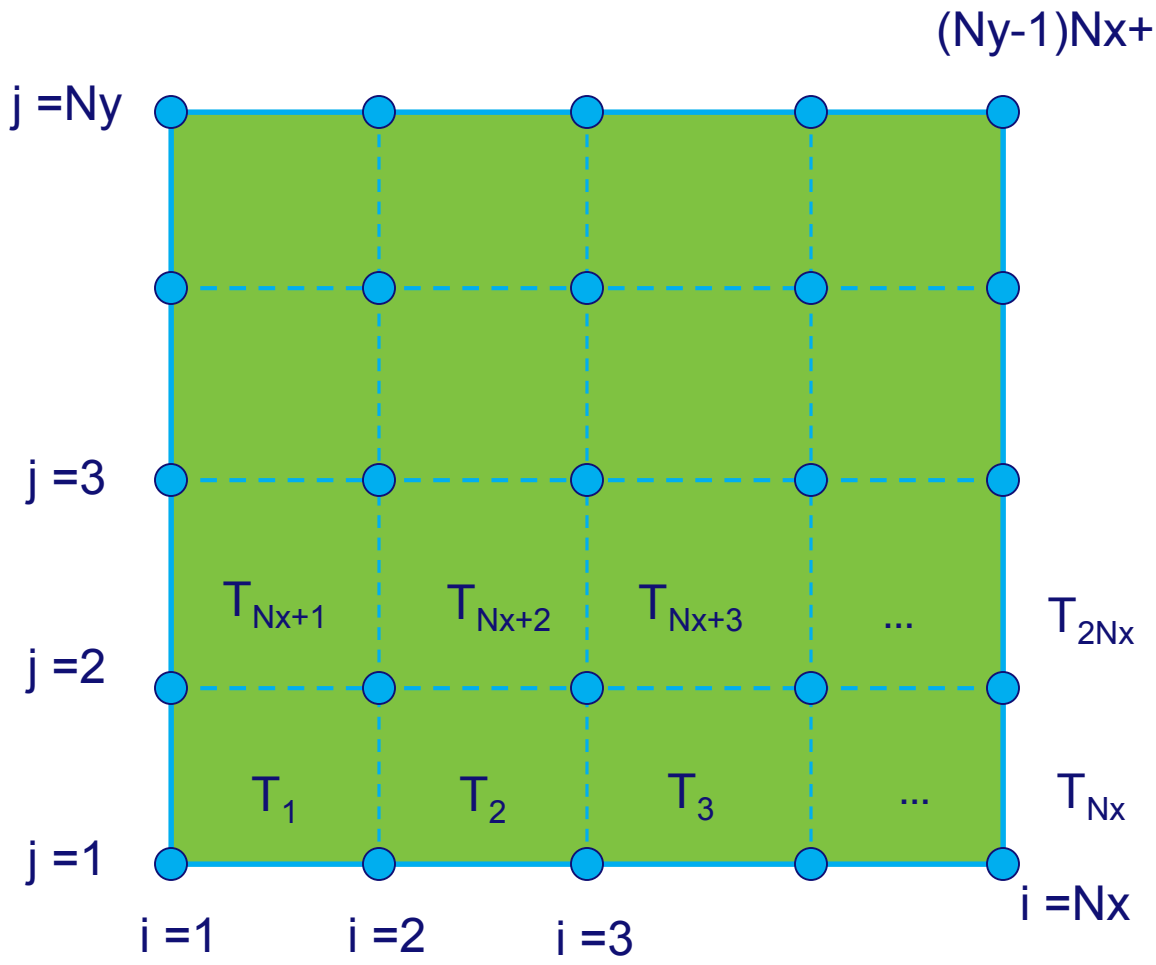
The steady state problem:

$$\nabla^2 T = 0$$



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

TRACK TEMPERATURE ON A GRID



Index of a node is given by:

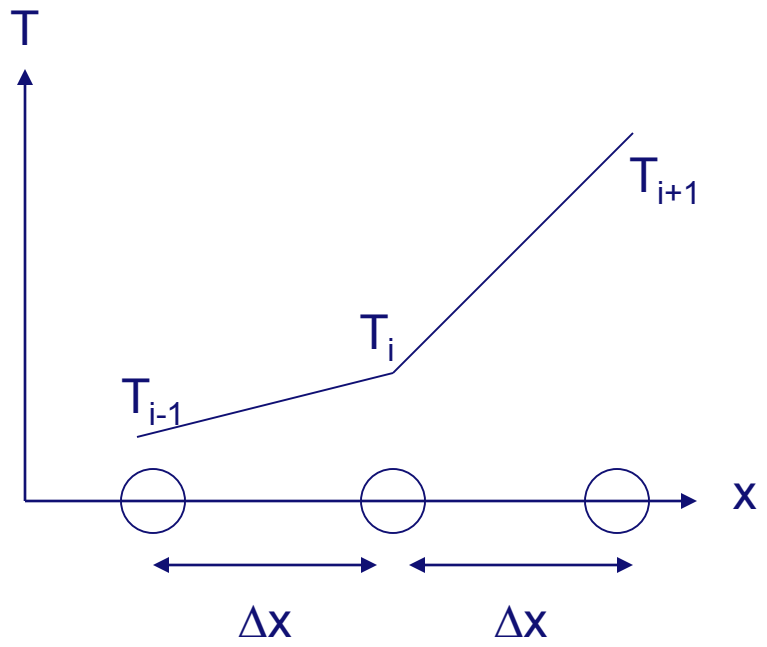
$k = i + Nx(j-1)$

So that:

$T_{i,j} = T_{k=i+Nx(j-1)}$

ESTIMATES OF THE SECOND DIFFERENTIALS FOR x

- **Assume a piece-wise linear profile in the temperature, e.g.:**



$$\frac{\partial^2 T}{\partial x^2} \approx \frac{\left.\frac{\partial T}{\partial x}\right|_{i+1/2} - \left.\frac{\partial T}{\partial x}\right|_{i-1/2}}{\Delta x}$$

$$\approx \frac{\frac{\left(T_{i+1,j} - T_{i,j}\right)}{\Delta x} - \frac{\left(T_{i,j} - T_{i-1,j}\right)}{\Delta x}}{\Delta x}$$

$$= \frac{T_{i+1,j} - 2T_{i,j} + T_{i-1,j}}{\Delta x^2}$$

INCLUDE THE ESTIMATES OF THE SECOND DIFFERENTIALS FOR y

$$\frac{T_{i+1,j} - 2T_{i,j} + T_{i-1,j}}{\Delta x^2} + \frac{T_{i+1,j} - 2T_{i,j} + T_{i-1,j}}{\Delta y^2} = 0$$

$$\frac{T_{k+1} - 2T_k - T_{k-1}}{\Delta x^2} + \frac{T_{k+Nx} - 2T_k - T_{k-Nx}}{\Delta y^2} = 0$$

Equal spaced grid $\Delta x = \Delta y = 1$

$$T_{k-Nx} + T_{k-1} - 4T_k + T_{k+1} + T_{k+Nx} = 0$$

$$AT = b$$

LET'S WRITE THIS STUFF AS MATRIX EQUATIONS

```
Nx=5;          %number of points along x direction
Ny=5;          %number of points in the y direction

e = ones(Nx*Ny,1);
A = spdiags([e,e,-4*e,e,e],[-Nx,-1,0,1,Nx],Nx*Ny,Nx*Ny);
```

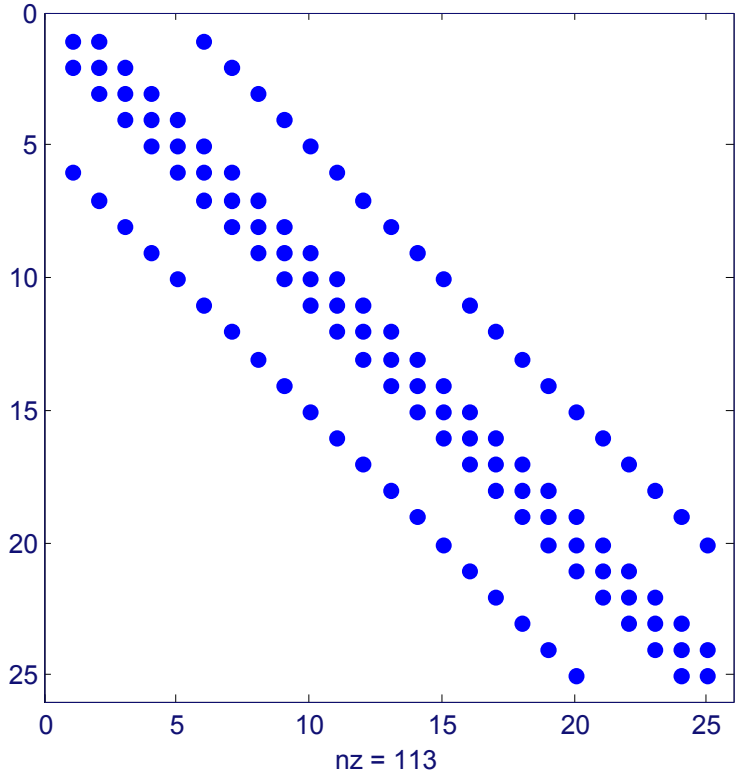
MATRIX SPARSITY

`spy(A)` Nx=Ny = 5

A sparse matrix structure,

which is not tridiagonal: there are offset bands.

Offset bands can cause your trouble!!

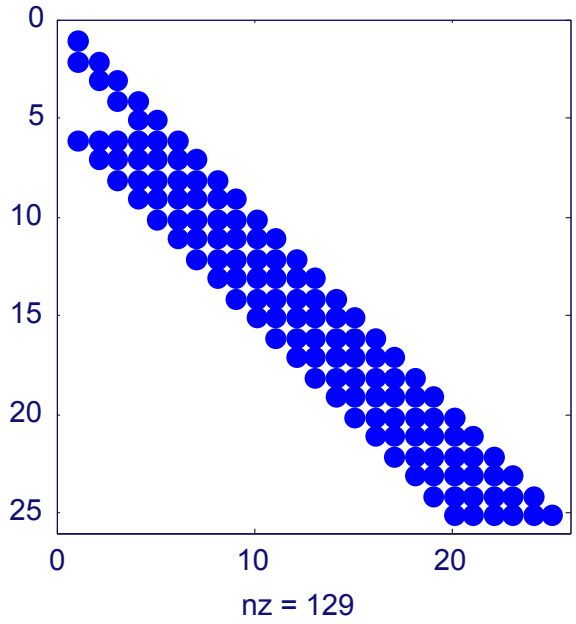


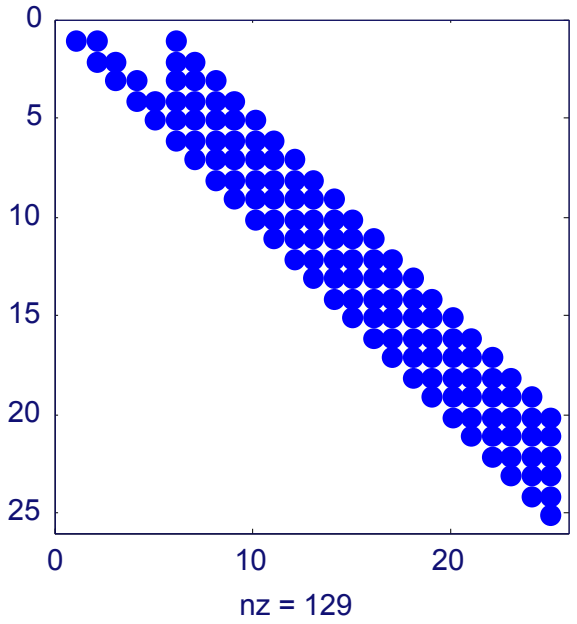
LU decomposition

```
[L,U,P] = lu(A)
subplot(1,2,1)
spy(L)
subplot(1,2,2)
spy(U)
```

With LU decomposition we produce matrices that are less sparse than the original matrix.

Sparse storage often required, and also numerical techniques that fully utilizes this!





How to impose boundary conditions?

- **For the nodes on the boundary, we have a simple equation:**

$$T_{k,boundary} = \text{Some fixed value}$$

- **However, we have set *all nodes* to be a function of their neighbors…**
  - **Find the boundary node indices using k = i + Nx(j-1)**
    - **i = 1, j = 1:Ny**
    - **i = Nx, j = 1:Ny**
    - **j = 1, i = 1:Nx**
    - **j = Ny, i = 1:Nx**
  - **Reset the row in A to zeros, set $A_{kk} = 1$**
  - **Set value in rhs: $b_k = T_{boundary}$**
- **Boundary conditions are often more elaborate to implement!**

A full program, including solver, may look like (OASE):

```
function [x,y,T,A] = solveLaplaceEq(Nx,Ny)
% Solves the Laplace equation for steady-state heat conduction

if (nargin < 2)
    Nx = 5;
    Ny = 5;
end

d = 1/Nx;                                % Grid spacing
alpha = 1;                               % Heat conduction
Tb = [10 20 30 40];                      % Fixed boundary temperatures

% Fill sparse matrix with [1 1 -4 1 1]
e = ones(Nx*Ny,1);
A = spdiags([e,e,-4*e,e,e],[-Nx,-1,0,1,Nx],Nx*Ny,Nx*Ny);
A = A*alpha/d^2;                 % This doesnt matter, steady state...
b = zeros(Nx*Ny,1);              % Right hand side vector

[A,b] = setBoundaryConditions(A,b,Tb,Nx,Ny);% Set boundary conditions

T = A\b;                                          % Solve matrix
Tc = reshape(T,[Nx,Ny]);                          % Reshape x-vec to mat Nx,Ny
[xc yc] = meshgrid(1:Nx,1:Ny);                    % Get position arrays
surf(xc,yc,Tc)                                    % Surface plot
```

Solution to the Laplace equation

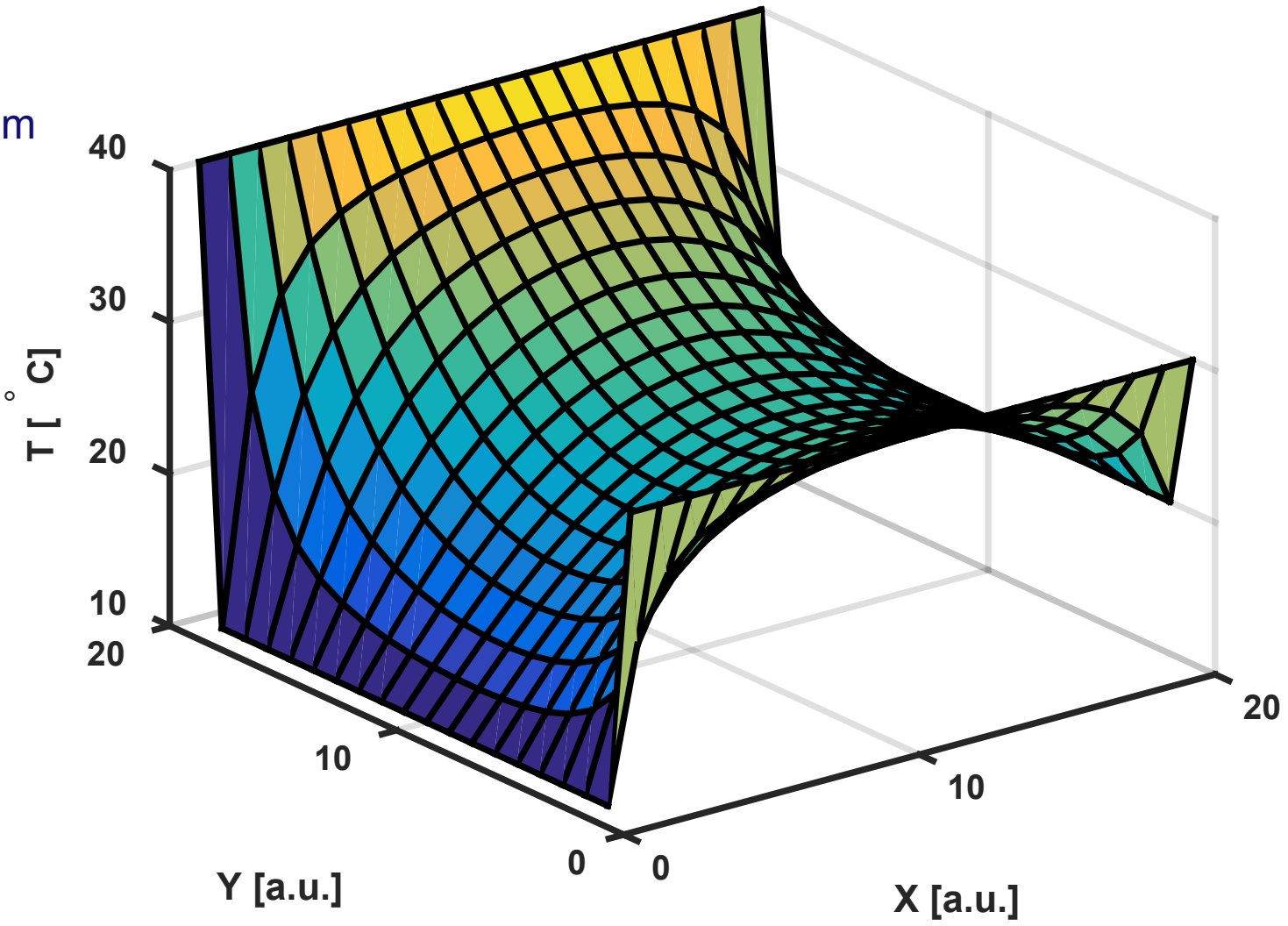
Solve the system with \, here for 20x20 and boundaries:

Tb1 = 10

Tb2 = 20

Tb3 = 30

Tb4 = 40



LU DECOMPOSITION

- Gaussian elimination on a matrix like A requires more memory (with 3D problems, the offset in the diagonal would even be bigger!)
- In general extra memory allocation will not be a problem for MATLAB
- MATLAB is clever, in that sense that it attempts to reorder equations, to move elements closer to the diagonal)

Iterative methods

- Alternatives for Gaussian elimination
- Use iterative methods when systems are large and sparse.
- Often such systems are encountered when we want to solve PDE's of higher dimensions (>1D)

Examples of iterative methods

- Jacobi method
- Gauss-Seidel method
- Succesive over relaxation

- `bicg` – Bi-conjugate gradient method
- `pcg` – preconditioned conjugate gradient method
- `gmres` – generalized minimum residuals method
- `bicgstab` – Bi-conjugate gradient method

The Jacobi method

- **In our example we derived the following equation:**

$$T_{k-Nx} + T_{k-1} - 4T_k + T_{k+1} + T_{k+Nx} = 0$$

- **Rearranging gives:**

$$T_k = \frac{T_{k-Nx} + T_{k-1} + T_{k+1} + T_{k+Nx}}{4}$$

THE JACOBI METHOD

- **In the Jacobi scheme the iteration proceeds as follows:**
  - **Start with an initial guess for the values of T at each node, we calculate an new, updated values using the equation of the previous slide and store a new vector:**
  - **Do this for all other nodes, and use new values as guess, Repeat!**

$$T_{k,new} = \frac{T_{k-Nx,old} + T_{k-1,old} + T_{k+1,old} + T_{k+Nx,old}}{4}$$

NOW WITH MATRICES

Consider a matrix A:

$$A = \begin{pmatrix} * & * & & & * & \\ * & * & * & & & * \\ & * & * & * & & \\ & & * & * & * & \\ * & & & * & * & * \\ & * & & & * & * \end{pmatrix}$$

Split it into a diagonal matrix D and another matrix S:

$$A = \begin{pmatrix} * & & & & & \\ & * & & & & \\ & & * & & & \\ & & & * & & \\ & & & & * & \\ & & & & & * \end{pmatrix} + \begin{pmatrix} & * & & & * & \\ * & & * & & & * \\ & * & & * & & \\ & & * & & * & \\ * & & & * & & * \\ & * & & & * & \end{pmatrix}$$

SOLVE AT-b

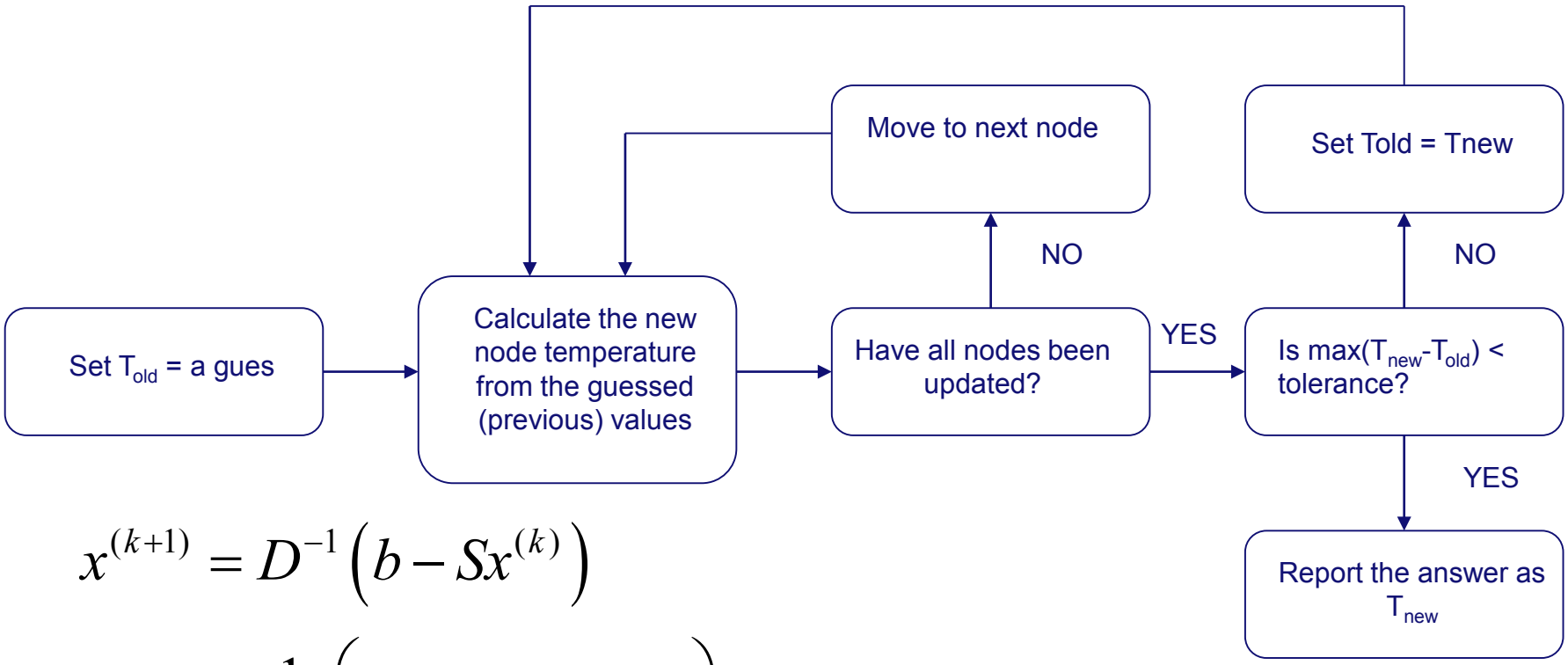
- **Now we can solve AT=b by:**
  - **(D+S).T = b**
  - **D.T = b –S.T**
  - **$D.T^{new} = b - S.T^{old}$**
  - **$T^{new} = D^{-1}(b-S.T^{old})$**

$$x^{(k+1)} = D^{-1}\left(b - Sx^{(k)}\right)$$

$$x_i^{(k+1)} = \frac{1}{A_{ii}}\left(b_i - \sum_{j \neq i} A_{ij} x_j^{(k)}\right)$$

See Wikipedia: Jacobi_method

DIAGRAM OF JACOBI METHOD



$$x^{(k+1)} = D^{-1}\left(b - Sx^{(k)}\right)$$

$$x_i^{(k+1)} = \frac{1}{A_{ii}}\left(b_i - \sum_{j \neq i} A_{ij} x_j^{(k)}\right)$$

Only two vectors need to be stored: the old x (k) and the new x (k+1)

Exercise: implement this method and solve a (small) Ax=b system

The core of the solver may look like:

```
while ( norm(A*x-b, 2) > err && it_jac < 1000 )
    x_old = x;
    for i=1:N
        s = 0;
        for j = 1:N
            if (j ~= i)
                % Sum off-diagonal*x_old
                s = s+A(i,j)*x_old(j);
            end
        end
        % Compute new x value
        x(i) = (b(i)-s)/A(i,i);
    end
    % Increase number of iterations
    it_jac = it_jac+1;
end
```

Convergence criterion

Protection against infinite loops

Store the previous solution

Resetting the sum for each row

Nested loops, conditional statement, scalar computations. This makes Matlab slow. It is much better at performing matrix operations as a whole!

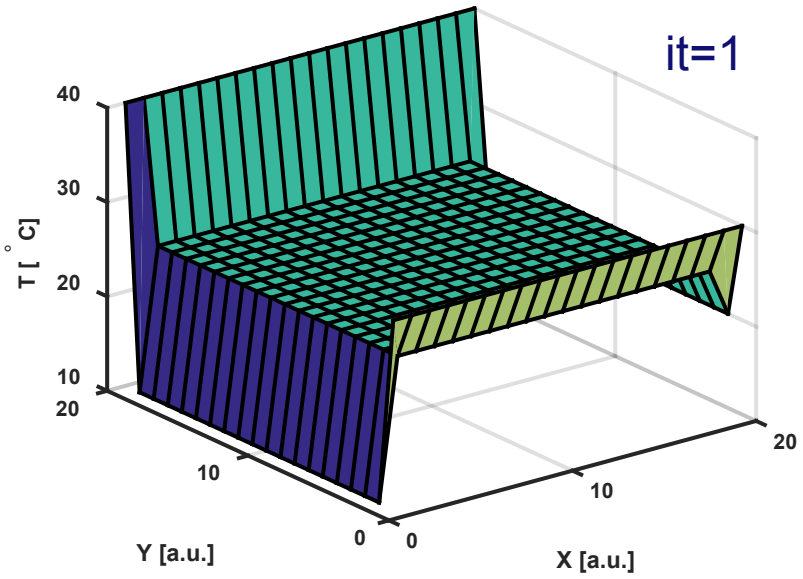
A few remarks on the implementation

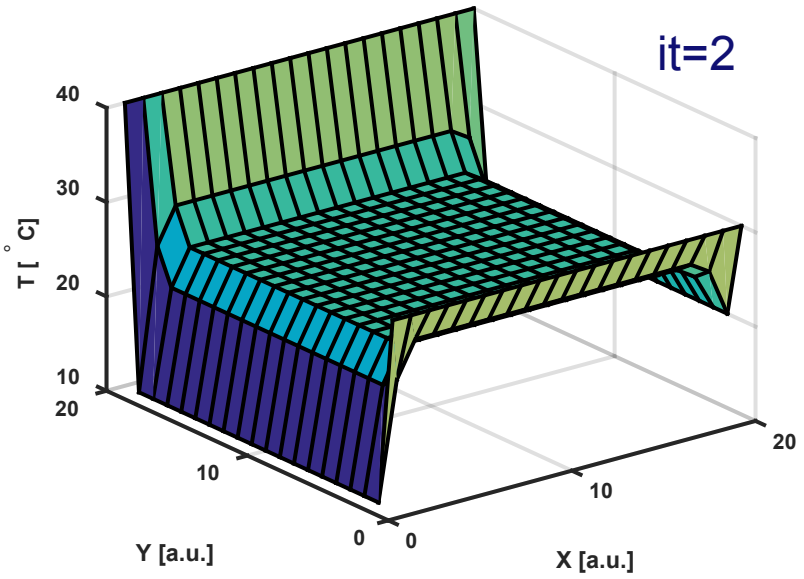
- Convergence criterion considerations
  - Compare current and previous solution
  - Compare Ax and b
  - Based on the norm of a vector. Which one?
    - L1-norm (sum)
    - L2-norm (Euclidian distance)
    - L∞-norm (max)
- Number of iterations
  - May be used to detect non-converging cases.
- Start vector x
  - Not shown in the previous example, but should be there
  - Can have huge impact on performance!

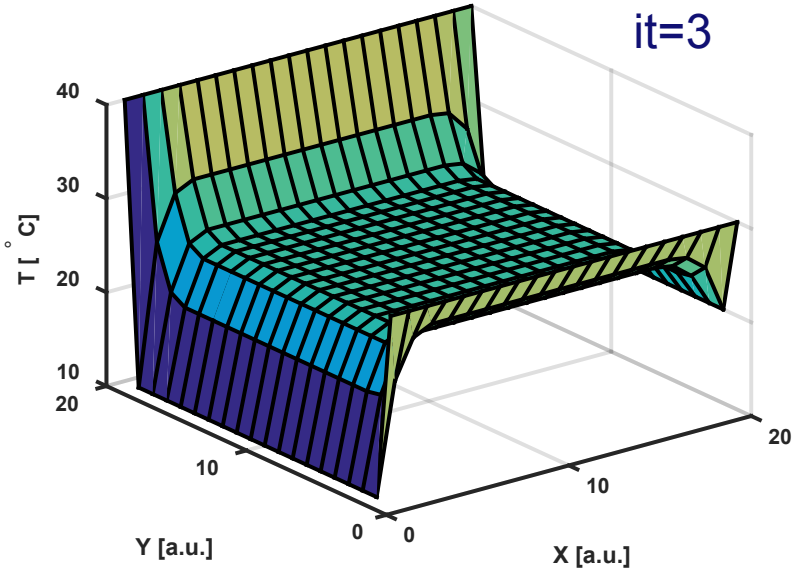
The solver using array indices:

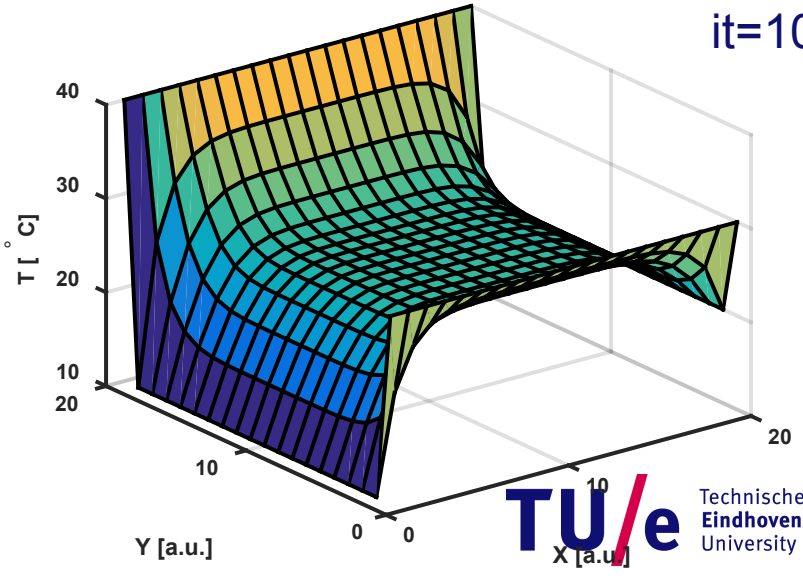
```
function [x0,it_jac] = solveJacobi(A,b,tol)
% Set default error
if nargin < 3
    tol = 1e-6;
end

x0 = 25*ones(size(b));    % Initial guess
x = zeros(size(b));       % Pre-allocate vector x
N = length(A);            % Number of equations
it_jac = 1;               % Init number of iterations

% --- Initial iteration to get x here ---

while ( norm(x-x0, 1) > tol > tol && it_jac < 1000 )
    x0 = x;
    for i = 1:N
        x(i) = (1/A(i,i))*((b(i) - A(i,[1:i-1,i+1:N]) * ...
                x0([1:i-1,i+1:N])));
    end
    it_jac = it_jac + 1;
end

it_jac
```

Iterations…









Gauss-Seidel method

- Gauss-Seidel method is quite similar to Jacobi method
- Define a lower and strictly upper triangular matrix, such that A = L + U
  - (L+U)x=b
  - Lx = b-Ux
  - $Lx^{new} = b - Ux^{old}$
  - $x^{new} = L^{-1}(b-Ux^{old})$

See Wikipedia: Gauss-Seidel_method

$$x^{(k+1)} = L^{-1}\left(b - Ux^{(k)}\right)$$

$$x_i^{(k+1)} = \frac{1}{A_{ii}}\left(b_i - \sum_{j<i} A_{ij} x_j^{(k+1)} - \sum_{j>i} A_{ij} x_j^{(k)}\right)$$

SUMMARY

- Partial differential equations can be written as sparse systems of linear equations
- Sparse systems can be handled with a direct method like Gaussian elimination
- If you have systems of more than 1 dimension, a direct method still can be used, if there are no memory issues, otherwise an iterative method may be attractive.
- The Jacobi method was introduced. Many other methods are based on the Jacobi method (SOR method, for example)