



**University of
Nottingham**

UK | CHINA | MALAYSIA

Computer Modelling Techniques

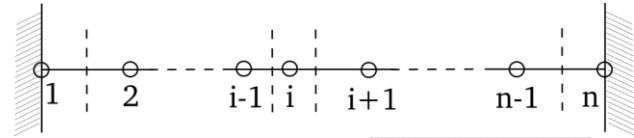
**Numerical Methods
Lecture 2: Solution of linear systems**

Mirco Magnini

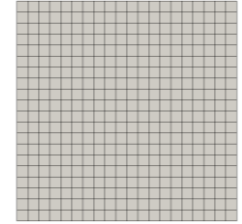
Recap

Steady-state heat conduction:

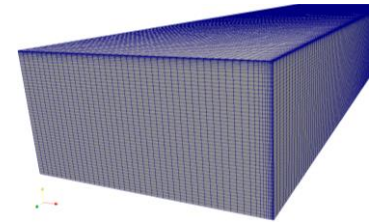
1D: $\frac{d}{dx} \left(\lambda \frac{dT}{dx} \right) + S(x, T) = 0$



2D: $\frac{\partial}{\partial x} \left(\lambda \frac{\partial T}{\partial x} \right) + \frac{\partial}{\partial y} \left(\lambda \frac{\partial T}{\partial y} \right) + S(x, y, T) = 0$

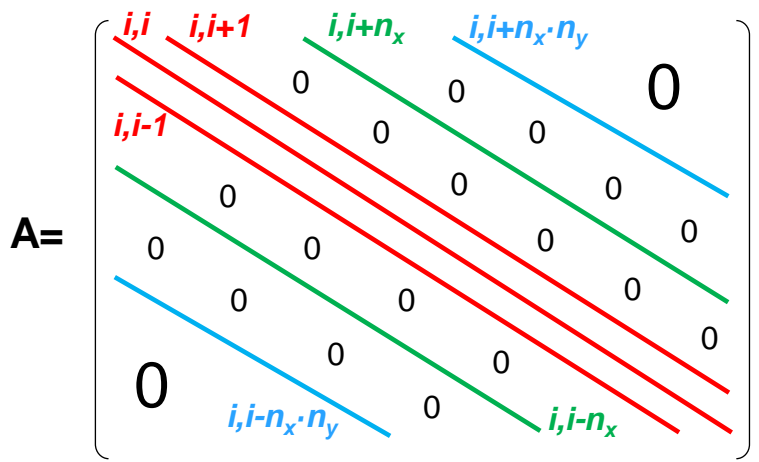


3D: $\frac{\partial}{\partial x} \left(\lambda \frac{\partial T}{\partial x} \right) + \frac{\partial}{\partial y} \left(\lambda \frac{\partial T}{\partial y} \right) + \frac{\partial}{\partial z} \left(\lambda \frac{\partial T}{\partial z} \right) + S(x, y, z, T) = 0$



FV $\rightarrow a_T T_T + a_N T_N + a_W T_W + a_P T_P + a_E T_E + a_S T_S + a_B T_B = b$

1D 2D 3D



A × **T** = **B**: linear algebraic system with n equations
A: n × n matrix; **T**, **B**: n × 1 vectors

Matlab's mldivide: $\rightarrow T = A \setminus B$

But how is a linear system solved numerically?

Today's menu

- Solution of a linear system
- System condition
- Direct methods (Gaussian elimination, TDMA)
- Iterative methods (Gauss-Seidel)
- Iterative methods: Solution relaxation
- Iterative methods: Convergence criteria
- Seminar 2, demo 2: implement the G-S and TDMA methods in Matlab

Expected outcome: know advantages/limitations of direct/iterative methods to solve linear systems; be able to write down solution algorithms; use matlab to solve linear systems.

In general, the numerical solution of a partial differential equation on a computational mesh with n cells involves the solution of a system of n linear equations, $\mathbf{A}\times\mathbf{T}=\mathbf{B}$, with n unknowns (T_1, \dots, T_n) , where:

$$A = \begin{bmatrix} a_{1,1} & \cdots & a_{1,n} \\ \vdots & \ddots & \vdots \\ a_{n,1} & \cdots & a_{n,n} \end{bmatrix} \quad T = \begin{bmatrix} T_1 \\ \vdots \\ T_n \end{bmatrix} \quad B = \begin{bmatrix} b_1 \\ \vdots \\ b_n \end{bmatrix}$$

Depending on the morphology or regularity of \mathbf{A} , e.g. full, tridiagonal, band-diagonal, triangular, symmetric, positive definite, etc. etc., there exist optimal solution algorithms.

Solution approaches can be grouped into:

- **Direct methods:** allow to compute the solution (T_1, \dots, T_n) within a finite number of operations.
- **Iterative methods:** are based on the sequential evaluation of approximate solutions that are supposed to converge to the exact solution after ∞ iterations.

There are features of **A** that make it easier to solve the system, others that make it impossible.

- **Well-conditioned** system: a small change in one or more of the coefficients results in a similar small change in the solution. This is the most desirable situation.
- **Ill-conditioned** systems: two or more equations of the system are very similar and small changes in coefficients result in large changes in the solution. This is a dangerous situation.

Example:

$$\begin{bmatrix} 1 & 2 \\ 1.1 & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 10 \\ 10.4 \end{bmatrix} \longrightarrow \begin{matrix} x_1 = 4 \\ x_2 = 3 \end{matrix} \quad \text{BUT} \quad \begin{bmatrix} 1 & 2 \\ 1.05 & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 10 \\ 10.4 \end{bmatrix} \longrightarrow \begin{matrix} x_1 = 8 \\ x_2 = 1 \end{matrix}$$

- **Singular** systems: two or more equations are identical, so that $\det(A) = 0$. This way, we have only $n-1$ independent equations, with n unknowns. The system is underdetermined, and **A×T=B** cannot be solved.

Ill-conditioned system represent a problem for the calculation, in particular for direct methods.

Softwares (Matlab) choose the best solution algorithm depending on **A**

The **Gaussian elimination** is a very robust algorithm based on the manipulation of \mathbf{A} to turn it into an upper triangular matrix, where (T_n, \dots, T_1) are then obtained by back-substitution.

$$\begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & a_{2,3} & \cdots & a_{2,n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{n,1} & a_{n,2} & a_{n,3} & \cdots & a_{n,n} \end{bmatrix} \times \begin{bmatrix} T_1 \\ T_2 \\ \vdots \\ T_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}$$

Step 1. Forward elimination

1. Eliminate T_1 from Eq. (i), with $i=2, \dots, n$, by subtracting Eq. (1)* $a_{i,1}/a_{1,1}$ from Eq. (i):

$$A^{(1)} = \begin{pmatrix} a_{1,1} & a_{1,2} & a_{1,3} & \cdots & a_{1,n} \\ 0 & a_{2,2} - \frac{a_{2,1}}{a_{1,1}}a_{1,2} & a_{2,3} - \frac{a_{2,1}}{a_{1,1}}a_{1,3} & \cdots & a_{2,n} - \frac{a_{2,1}}{a_{1,1}}a_{1,n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & a_{n,2} - \frac{a_{n,1}}{a_{1,1}}a_{1,2} & a_{n,3} - \frac{a_{n,1}}{a_{1,1}}a_{1,3} & \cdots & a_{n,n} - \frac{a_{n,1}}{a_{1,1}}a_{1,n} \end{pmatrix} \quad B^{(1)} = \begin{pmatrix} b_1 \\ b_2 - \frac{a_{2,1}}{a_{1,1}}b_1 \\ \vdots \\ b_n - \frac{a_{n,1}}{a_{1,1}}b_1 \end{pmatrix}$$

2. Eliminate T_2 from Eq. (i), with $i=3, \dots, n$, by subtracting Eq. (2)* $a_{i,2}^{(1)}/a_{2,2}^{(1)}$ from Eq. (i).

... and so on until, after **N-1** operations, $\mathbf{A}^{(N-1)}$ is upper triangular



Direct methods – Gaussian elimination

... and so on until, after **N-1** operations, $A^{(N-1)}$ is upper triangular:

$$A^{(N-1)} = \begin{pmatrix} a_{1,1}^{(N-1)} & a_{1,2}^{(N-1)} & a_{1,3}^{(N-1)} & \cdots & a_{1,n}^{(N-1)} \\ 0 & a_{2,2}^{(N-1)} & a_{2,3}^{(N-1)} & \cdots & a_{2,n}^{(N-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & a_{n,n}^{(N-1)} \end{pmatrix} \quad B^{(N-1)} = \begin{pmatrix} b_1^{(N-1)} \\ b_2^{(N-1)} \\ \vdots \\ b_n^{(N-1)} \end{pmatrix}$$

Step 2. Back substitution: solution of the new system $A^{(N-1)} \times T = B^{(N-1)}$

Line n: $a_{n,n}^{(N-1)} T_n = b_n^{(N-1)} \Rightarrow T_n = \frac{b_n^{(N-1)}}{a_{n,n}^{(N-1)}}$

Line n - 1: $a_{n-1,n-1}^{(N-1)} T_{n-1} + a_{n-1,n}^{(N-1)} T_n = b_{n-1}^{(N-1)} \Rightarrow T_{n-1} = \frac{b_{n-1}^{(N-1)} - a_{n-1,n}^{(N-1)} T_n}{a_{n-1,n-1}^{(N-1)}}$

Line i: $T_i = \frac{b_i^{(N-1)} - \sum_{k=i+1}^n a_{i,k}^{(N-1)} T_k}{a_{i,i}^{(N-1)}}$

Example: Consider the following system with 3 equations

$$\begin{bmatrix} 2 & 1 & -1 \\ 1 & 3 & 2 \\ 1 & -1 & 4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 13 \\ 11 \end{bmatrix}$$

From the 2nd and 3rd line, subtract the 1st line multiplied by $a_{i,1}/a_{1,1}$:

$$\begin{bmatrix} 2 & 1 & -1 \\ 0 & 2.5 & 2.5 \\ 0 & -1.5 & 4.5 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 12.5 \\ 10.5 \end{bmatrix}$$

From the 3rd line, subtract the 2nd line multiplied by $a_{3,2}/a_{2,2}$:

$$\begin{bmatrix} 2 & 1 & -1 \\ 0 & 2.5 & 2.5 \\ 0 & 0 & 6 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 12.5 \\ 18 \end{bmatrix} \xrightarrow{\text{Back substitution}} \begin{matrix} x_3 = 3 \\ x_2 = 2 \\ x_1 = 1 \end{matrix}$$

- The method is very reliable, it always calculates a solution (if a solution exists).
- The number of operations is **proportional to n^3** .
- Because of the many operations to manipulate **A**, the method suffers from round-off errors, i.e. the loss of significant figures as calculations are repeated on the same coefficient.
- Ill-conditioned systems pose significant problems. Small variations on the coefficients of **A** due to round-off errors during the forward elimination procedure, may cause large changes in the solution. Remedies: pivoting, i.e. rearrange rows/columns to have larger coefficients along diagonals; scaling: rescale rows such that the largest coefficient is close to 1.
- The matrix of the coefficients **A** needs to be entirely stored even if there are many zeros, requiring memory space **proportional to n^2** .
- The algorithm is more complex to program than iterative methods.

A Gaussian elimination method specific for tridiagonal matrices:

TDMA (TriDiagonal Matrix Algorithm)

Node i) $a_i T_i = b_i T_{i+1} + c_i T_{i-1} + d_i$

Node 1) $c_1 = 0$ Node n) $b_n = 0$

Translation:

$$\begin{aligned} a_i &\rightarrow a_{P,i} \rightarrow a_{i,i} \\ b_i &\rightarrow -a_{E,i} \rightarrow -a_{i,i+1} \\ c_i &\rightarrow -a_{W,i} \rightarrow -a_{i,i-1} \\ d_i &\rightarrow b_i \end{aligned}$$

in lecture 1

Forward elimination:

Node 1) $a_1 T_1 = b_1 T_2 + d_1 \Rightarrow T_1 = \frac{b_1 T_2 + d_1}{a_1} = P_1 T_2 + Q_1$

Node 2) $a_2 T_2 = b_2 T_3 + c_2 (P_1 T_2 + Q_1) + d_2 \Rightarrow T_2 = \frac{b_2 T_3 + c_2 Q_1 + d_2}{a_2 - c_2 P_1} = P_2 T_3 + Q_2$

Node i) $a_i T_i = b_i T_{i+1} + c_i T_{i-1} + d_i \Rightarrow T_i = P_i T_{i+1} + Q_i,$

$$P_i = \frac{b_i}{a_i - c_i P_{i-1}}, Q_i = \frac{c_i Q_{i-1} + d_i}{a_i - c_i P_{i-1}}$$

Node n) $T_n = Q_n,$ $Q_n = \frac{c_n Q_{n-1} + d_n}{a_n - c_n P_{n-1}}$

Back substitution:

$$T_n = Q_n$$

$$T_i = P_i T_{i+1} + Q_i \Rightarrow T_i$$

$$T_{n-1} = P_{n-1} T_n + Q_n \Rightarrow T_{n-1}$$

$$T_1 = P_1 T_2 + Q_1 \Rightarrow T_1$$

Iterative methods start with an initial guess for the solution and use the algebraic equations to systematically improve it, until the solution is sufficiently close to the exact solution of the system.

$\mathbf{A} \times \mathbf{T} = \mathbf{B}$, if \mathbf{T} is the exact solution

After m iterations we have an approximate solution $\mathbf{T}^{(m)}$: $\mathbf{A} \times \mathbf{T}^{(m)} = \mathbf{B} - \mathbf{r}^{(m)}$

where \mathbf{r} is the residual vector, and the error vector $\mathbf{e}^{(m)} = \mathbf{T} - \mathbf{T}^{(m)}$

The iterative procedure is successful if $\lim_{m \rightarrow \infty} |\mathbf{e}^{(m)}| = 0$, so that $\lim_{m \rightarrow \infty} |\mathbf{r}^{(m)}| = 0$

In general:

- Number of operations for each iteration is **proportional to n** , but the number of iterations m is not known beforehand.
- Convergence is not guaranteed (will see).
- There is no need to store the entire \mathbf{A} but only the nonzero elements, memory $\sim n$.
- For non-linear problems with a large and sparse matrix (typical CFD case), they are preferable to direct methods because less computationally expensive.
- They are less sensitive to round-off errors than direct methods.
- They are much easier to code.

Iterative methods – Point-iterative methods

At each m^{th} iteration, equations are sequentially solved from 1st to n^{th} using guess values for the non- i^{th} unknowns of each i^{th} equation:

$$a_{i,1}T_1 + \dots + a_{i,i}T_i + \dots + a_{i,n}T_n = b_i \Rightarrow T_i = \frac{b_i}{a_{i,i}} - \sum_{j=1, j \neq i}^n \frac{a_{i,j}}{a_{i,i}} T_j^*, \quad T_j^*: \text{guess values}$$

Gauss-Seidel method: guess are the most recently available values: $T_j^* = \begin{cases} T_j^{(m-1)}, & \text{if } j > i \\ T_j^{(m)}, & \text{if } j < i \end{cases}$

$$\longrightarrow T_i^{(m)} = \frac{b_i}{a_{i,i}} - \sum_{j=1}^{i-1} \frac{a_{i,j}}{a_{i,i}} T_j^{(m)} - \sum_{j=i+1}^n \frac{a_{i,j}}{a_{i,i}} T_j^{(m-1)}$$

Example:

$$T_1 = 0.4T_2 + 0.2$$

$$T_2 = T_1 + 1$$

Accuracy to the 2nd decimal

n	0	1	2	3	4	5	6	7
T1	0	0.2	0.68	0.872	0.949	0.98	0.992	0.997
T2	0	1.2	1.68	1.872	1.949	1.98	1.992	1.997

7 iterations

Exact solution: $T_1 = 1, T_2 = 2$

Iterative methods – Point-iterative methods

Gauss-Seidel method: guess are the most recently available values: $T_j^* = \begin{cases} T_j^{(m-1)}, & \text{if } j > i \\ T_j^{(m)}, & \text{if } j < i \end{cases}$

$$\longrightarrow T_i^{(m)} = \frac{b_i}{a_{i,i}} - \sum_{j=1}^{i-1} \frac{a_{i,j}}{a_{i,i}} T_j^{(m)} - \sum_{j=i+1}^n \frac{a_{i,j}}{a_{i,i}} T_j^{(m-1)}$$

Example:

$$T_1 = 0.4T_2 + 0.2$$

$$T_2 = T_1 + 1$$

Accuracy to the 2nd decimal

n	0	1	2	3	4	5	6	7
T1	0	0.2	0.68	0.872	0.949	0.98	0.992	0.997
T2	0	1.2	1.68	1.872	1.949	1.98	1.992	1.997

But convergence depends on the form of the coefficient matrix. Rearranging the equations:

$$T_1 = T_2 - 1$$

$$T_2 = 2.5T_1 - 0.5$$


n	0	1	2	3	4	5	...	∞
T1	0	-1	-4	-11.5	-30.25	-77.13	...	divergence
T2	0	-3	-10.5	-29.25	-76.13	-193.32	...	divergence

Scarborough criterion: a sufficient condition for the convergence of Gauss-Seidel is

$$\frac{\sum_{j=1, j \neq i}^n |a_{i,j}|}{|a_{i,i}|} \begin{cases} \leq 1 & \text{for all equations} \\ < 1 & \text{for at least 1 eq.} \end{cases} \longrightarrow A \text{ must be } \mathbf{diagonally\ dominant}$$



Relaxation techniques: control of the rate of change of the variables in point-iterative methods:

$$T_i^{(m)} = \frac{b_i}{a_{i,i}} - \sum_{j=1, j \neq i}^n \frac{a_{i,j}}{a_{i,i}} T_j^* = T_i^{(m-1)} + \frac{b_i}{a_{i,i}} - \sum_{j=1}^n \frac{a_{i,j}}{a_{i,i}} T_j^*$$



$$T_i^{(m)} = T_i^{(m-1)} + \omega \left[\frac{b_i}{a_{i,i}} - \sum_{j=1}^n \frac{a_{i,j}}{a_{i,i}} T_j^* \right]$$

↓ new guess ↓ old guess } calculated rate of change

- $\omega > 1$  **over-relaxation:** it speeds-up the otherwise slow convergence rate of iterative methods (never above 2)
- $\omega < 1$  **under-relaxation:** it slows down changes thus stabilizing the iterative procedure, sometimes necessary for nonlinear equations

The best ω which guarantees convergence with a minimum number of iterations is problem-dependent and it is not known beforehand!

When is the iterative procedure for solution arrested?

Note: exact solution T is not known beforehand.

- Absolute iteration error $|\delta^{(m)}| = |T^{(m)} - T^{(m-1)}| < tol$ Gives the rate of solution change, but no indication if we are converging to the correct solution
- Scaled iteration error $\frac{|\delta^{(m)}|}{|T^{(m-1)}|} < tol$ Rescales the iteration error to make it nondimensional, i.e. independent of the scale of the problem
- Absolute residuals $|r^{(m)}| = |B - A \times T^{(m)}| < tol$ Indicates how far are we from an exact solution of the linear system
- Scaled residuals $\frac{|r^{(m)}|}{|diag(A) \cdot T^{(m)}|} < tol$ Rescales the residual error to make it nondimensional, i.e. independent of the scale of the problem
- Scaled residuals (alternative) $\frac{|r^{(m)}|}{|r^{(1)}|} < tol$ Rescales the residual error with the error after the 1st iteration, i.e. it indicates how much has the solution improved since the beginning of the iterative process

Direct vs iterative methods

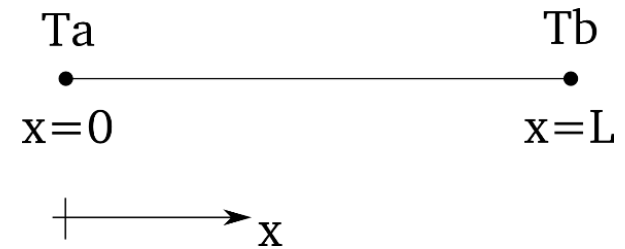
	Gaussian Elimination	Iterative Gauss-Seidel
Number of operations:	$(n^3)/3 + \text{Order of } (n^2)$	Order of $(m \times n)$ where $m = \text{number of iterations}$
Programming effort:	More difficult to program.	Easier to program.
Memory requirement:	Requires large storage (n^2) .	Requires minimal storage. Very suitable for very large sparse matrices.
Round-off errors:	Round-off errors accumulate in the elimination process. However, this is only an issue if the matrices are ill-conditioned.	Round-off errors are insignificant, since the final iteration may be regarded as an initial guess.
Reliability:	Robust, almost guaranteed to compute a solution (if there is one).	Less reliable, since it may not converge. Also, it requires diagonal dominance to converge.

What to take home from today's lecture

- Advantages/limits of direct/indirect methods to solve linear system
- Solution algorithm based on the Gaussian elimination
- Solution algorithm based on the Gauss-Seidel method
- Sufficient condition for G-S convergence (Scarborough criterion)
- How to under/over-relax the G-S solution
- Convergence criteria for iterative methods
- How to use Matlab to implement the G-S method

Worked example 1

Consider the finite-volume code to implement 1D steady-state heat conduction of We2 of L1. Instead of using matlab's backslash operator to solve the linear system, implement the iterative Gauss-Seidel method as a function. Use 10000 max iterations; tolerance 1e-12; initial guess $T = T_a$ everywhere.



Matlab function syntax: `[output1,output2,...]=functionName(input1,input2,...);`

Steps of the algorithm:

1. Receive initial guess, **A** and **B**, max number of iterations and tolerance as input.
2. Start a loop where equations are solved in sequence from $i=1$ to n .
3. Update the value of the residuals.
4. If residuals are above the tolerance, update guess values and go back to step 2.
5. Once tolerance is met, arrest the procedure and return the solution vector **T**.

Continues in the notes...

Repeat We1, implementing the TDMA.

Steps of the algorithm:

1. Receive **A** and **B**.
2. Calculate P_1 and Q_1 .
3. Calculate all the other P_i and Q_i in ascending order ($i=2, \dots, n-1$).
4. Calculate Q_n and set $T_n = Q_n$.
5. Calculate all the other T_i in descending order ($i=n-1, \dots, 1$).

Continues in the notes...