Simulation in Materials Engineering

BLOCK 2: Fundamentals of numerical analysis

J. Segurado

Departamento de Ciencia de Materiales
Polytechnic University of Madrid

September 24, 2018

Outline

1. Introduction to programming
2. Linear systems of equations
   - Introduction
   - Direct methods
   - Iterative methods and sparse matrices
3. Non linear equations and systems
4. Ordinary differential equations (ODEs)

Outline

1 Introduction to programming

2 Linear systems of equations
- Introduction
- Direct methods
- Iterative methods and sparse matrices

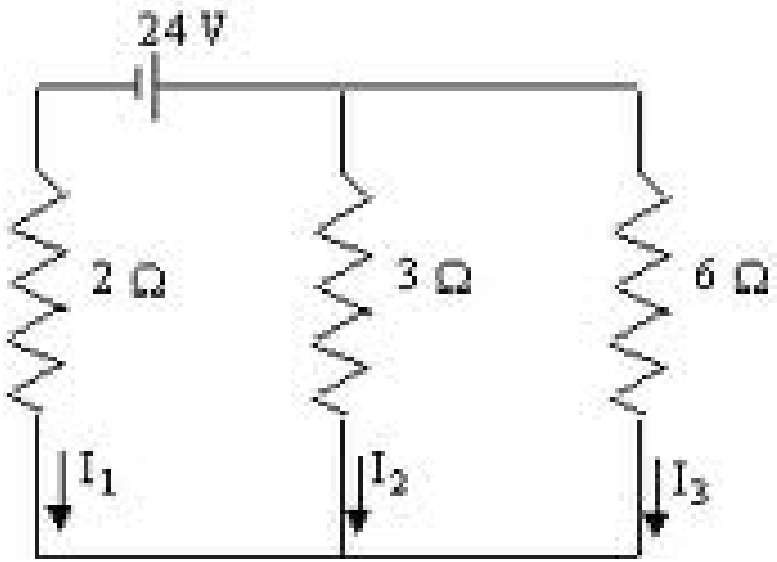
3 Non linear equations and systems

4 Ordinary differential equations (ODEs)

Introduction:Why linear equations are important?

They appear everytime in solving many engineering problems:

- The resolution of an electric network using Kirchoff laws leads to a system of equations of the intensities.
- The resolution of a hydraulic network, where the presures of each node are the unknowns



Introduction:Why linear equations are important?

- The numerical solution of any Partial Differential Equation, generally leads to a system of linear equations: i.e. heat equation in 1D solved by finite differences

$$U_t = U_{xx}$$
$$U(0,t) = U(1,t) = 0$$
$$U(x,0) = U_0(x)$$

The domain in space is discretized using a mesh $x_0, ..., x_J$ and in time using a mesh $t_0, ..., t_N$. The solution at each point/time is $u(x_j, t_n) = u_j^n$.
The unknowns here are the nodal temperatures at $n+1$ and the linear equation that result is

$$(1+2r)u_j^{n+1} - ru_{j-1}^{n+1} - ru_{j+1}^{n+1} = u_j^n$$

, r depending on a relation between spatial and time increment

Introduction

A linear system of equations is a set of m equations, each of them consisting on a linear combination of n unknowns, $x_i, i = 1...n$ and an indpendent term $b_i, i = 1...m$

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &= b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n &= b_2 \\ &\vdots \\ a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n &= b_m \end{aligned}$$

1. If the system has a solution and this solution is unique. This in known as a compatible determined system
2. If the system has solution, but there exist infinite solutions, the system is called compatible indetermined system
3. If the system does not have a solution, the system is incompatible

Introduction

The linear system can be rewritten using matrices,

$$\mathbf{Ax} = \mathbf{b}$$

where

$$\mathbf{A}_{mn} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ & \vdots & & \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix}; \quad \mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}; \quad \mathbf{b} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix}$$

In this case, the type of system can be obtained using the *Rouchee-Fröbenius* theorem

1. if $\text{rank}(\mathbf{A}) = \text{rank}(\mathbf{A}|\mathbf{b}) = n$ the system is compatible determined
2. if $\text{rank}(\mathbf{A}) = \text{rank}(\mathbf{A}|\mathbf{b}) < n$ the system is compatible undetermined
3. if $\text{rank}(\mathbf{A}) \neq \text{rank}(\mathbf{A}|\mathbf{b})$ the system is uncompatible

Introduction

Examples,

$$\begin{aligned} 3x + 2y + z &= 1 \\ 2x + z &= 2 \\ x + y &= 1 \end{aligned}$$

is compatible and determined: has a unique solution
$x = 3, y = -2, z = -4$

$$\begin{aligned} 3x + 2y + z &= 1 \\ 2x + z &= 2 \\ x + 2y &= -1 \end{aligned}$$

is compatible and undetermined, solutions can be parametrized and correspond to $x = 1 - \lambda/2, y = -1 + \lambda/4, z = \lambda$

Introduction

$$\begin{aligned} 3x + 2y + z &= 1 \\ 2x + z &= 2 \\ x + 2y &= 1 \end{aligned}$$

is incompatible and no values of x, y, z can fulfill the 3 equations

Direct solution of determined linear systems

- The classic method (secondary school) to solve the system, Cramers rule

$$x_i = \frac{\det(\mathbf{A}_i)}{\det(\mathbf{A})}$$

  being $\mathbf{A}_i$ the matrix formed substituting the i-th column of $\mathbf{A}$ by $\mathbf{b}$

- The Cramers rule computes n+1 determinants, using Laplace $O(n) = 2(n+1)!$, what means that if $n = 20$, a computer with 10Gflops will take more than 324 years to solve the system!!

Classification of algorithms

In order to obtain the solution of a linear system, different methods can be used. They can be classified in two groups

- *direct methods* s if they yield the solution of the system in a finite number of steps, i.e. Cramer rule, LU decompostion
- *iterative methods* if they require (in principle) an infinite number of steps. In this case, the solution is approached by truncating the method and obtaining an approximate but accurate approach

For a *dense* matrix, no algorithm could provide a solution with less than $\approx n^2$ operations, but the use of one or other type of method will give better performance depending on the structure of the matrix.

Outline

1. Introduction to programming
2. Linear systems of equations
   - Introduction
   - Direct methods
   - Iterative methods and sparse matrices
3. Non linear equations and systems
4. Ordinary differential equations (ODEs)

Direct methods: LU decomposition

Assuming a compatible and determined system of the type

$$\mathbf{Ax} = \mathbf{b} \tag{1}$$

Being **A** non-singular, and if its leading principal minors are non-zero, then it can be proved that the matrix admit a decomposition such that

$$\mathbf{A} = \mathbf{LU}$$

with **L** a lower diagonal matrix and **U** an upper diagonal matrix:

$$\mathbf{L} = \begin{bmatrix} l_{11} & 0 & \cdots & 0 \\ l_{21} & l_{22} & 0 \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ l_{n1} & l_{n2} & \cdots & l_{nn} \end{bmatrix}; \mathbf{U} = \begin{bmatrix} u_{11} & u_{12} & \cdots & u_{1n} \\ 0 & u_{22} & \cdots & u_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & u_{nn} \end{bmatrix}$$

Direct methods: LU decomposition

In this case the equation 1 can be solved by two triangular systems

$$\mathbf{Ly} = \mathbf{b} \tag{2}$$

$$\mathbf{Ux} = \mathbf{y} \tag{3}$$

This systems are much easier to solve, the first system (2) can be solved by a *forward substitution*

$$y_1 = \frac{b_1}{l_{11}}$$

$$y_2 = \frac{b_2}{l_{22}} - \frac{l_{21}}{l_{22}} y_1$$

$$\vdots$$

$$y_i = \frac{1}{l_{ii}}(b_i - \sum_{j=1}^{j=i-1} l_{ij} y_j), \text{ for } i = 2, ..., n$$

Direct methods: LU decomposition

and after that, the second system (3) can also be solved recursively by a *backward substitution* provinding the final solution of the initial problem 1

$$x_n = \frac{y_{nn}}{u_{nn}}$$

$$\vdots$$

$$x_i = \frac{1}{u_{ii}}(b_i - \sum_{j=i+1}^{n} u_{ij}x_j), \text{ for } i = n-1, ..., 1$$

The number of operations for solving by forward substitution the system (2) is n^2 and the same number is necessary for the system (3), giving an order of $O(n^2)$ for the substitution step (yet the decomposition has not been solved and evaluated).

Gaussian elimination

The equation $\mathbf{A} = \mathbf{LU}$ does not have a unique solution, for example in a $3x3$ matrix only 9 equations are available for 12 unknows. In general there exist n^2 equations and $n^2 + n$ unknowns.
If $l_{ii} = 1, i = 1, ..., n$ then the solution is unique and can be computed using *Gaussian elimination*

```
[n m]=size(A);
if n==m
l=eye(n);
for k=1:n-1
   for i=k+1:n
      l(i,k)=A(i,k)/A(k,k);
      for j=k+1:n
         A(i,j)=A(i,j)-l(i,k)*A(k,j);
      endfor
   endfor
endfor
for i=1:n
   for j=i:n
      u(i,j)=A(i,j);
   endfor
endfor
disp(l);
disp(u);
else disp('Non square matrix')
endif
```

Gaussian elimination

The term $A(k,k)$ that appears during the k-th eliminations of the matrix is called *pivot*.The total number is $n-1$ pivots

OCTAVE/MATLAB exercise

- Write as a function the Gaussian elimination algorithm being **A** the input and **L** and **U** the outputs
- Define the matrix `A=[1 2 3 4;2 8 5 -2;3 5 3 0;4 -2 0 1]` and obtain **L** and **U**
- Obtain the value of **A** as the product of **L** and **U**
- Define the matrix `A=[ 1 2 3; 2 4 5; 3 5 6]` and obtain **L** and **U**. What happens? Why?
- Add to the function of **LU** decomposition the to obtention of the determinant
- Obtain the order of magnitude $O(n)$ of the method to decompose a general *nxn* matrix

Gaussian elimination summary

- The **LU** decomposition works only if **A** is non-singular ($\det(\mathbf{A}) \neq 0$) , and if its leading principal minors are non-zero
- Gaussian elimination provides the decomposition and the algorithm is of order $O(n) = 2n^3/3$ (non demonstrated here). In addition, two substitutions, each $O(n^2)$, have to be done for solving the system
- The method serves to obtain the determinant with $O(n^3)$ just by multiplying the diagonal elements of **U**

LU decomposition with pivoting

The **LU** decomposition works only if **A** non-singular ($\det(\mathbf{A}) \neq 0$) , and if its leading principal minors are non-zero. This condition is directly fulfilled for

1. A symmetric and positive definite* matrix **A**, this means that

$$^{*}\text{for all } \mathbf{x} \neq 0,\ \mathbf{x} \in \mathbb{R}^n, \mathbf{x}^T\mathbf{A}\mathbf{x} > 0$$

2. A diagonally dominated matrix>

$$|a_{ii}| \geq \sum_{j \neq i} |a_{ij}| \quad \text{for all } i$$

example, the matrix $\mathbf{A} = \begin{bmatrix} 3 & -2 & 1 \\ 1 & -3 & 2 \\ -1 & 2 & 4 \end{bmatrix}$

If **A** is non-singular but does the condition of the principal minors Gauss elimination does not work but a special type of factorization can be applied. This happens if some pivot becomes 0 during decomposition!

The pivoting technique

Example $\mathbf{A} = \begin{bmatrix} 0 & 1 & 2 \\ 1 & 3 & 4 \\ 2 & 4 & 1 \end{bmatrix}$ does not admit a **LU** decomposition, but the new matrix $\mathbf{A} = \begin{bmatrix} 1 & 3 & 4 \\ 0 & 1 & 2 \\ 2 & 4 & 1 \end{bmatrix}$ consisting of permuting rows 1 and 2 does work.

OCTAVE/MATLAB exercise

Run the Gaussian elimination program with the two matrices of the example
The original matrix **A** is invertible, then, which condition is not fulfilling to admit a **LU** decomposition?
For the matrix $\mathbf{A} = \begin{bmatrix} 1 & 1 + .5E-15 & 3 \\ 2 & 2 & 20 \\ 3 & 6 & 4 \end{bmatrix}$, obtain **L**, **U** using *LU*.
Now compute $\mathbf{A} - \mathbf{LU}$. What happens?

The pivoting technique

- For a general invertible **A** matrix, the rows can be permuted in order to avoid the divisions by zero that the Gaussian elimination algorithm produces (when *pivots* are 0 or very small).
- Moreover, a general algorithm will search for each row during the decomposition the row in which pivot is maximum as absolute value, and interchange both.

```
for k = 1:n
   find r such that |a(r,k)|=max|a(r,k)|, r=k,..,n
   exchange row k with row r
  for i = k+1:n
    l(i,k) = a(i,k)/a(k,k)
    for j = k+1:n
      a(i,j) = a(i,j)-l(i,k)*a(k,j)
    endfor
  endfor
endfor
```

The pivoting technique

- The row permutations of the original **A** are made as soon as this becomes necessary, without carrying out any a priori transformation on A.This technique is given the name of pivoting by row.
- The factorization returns then the original matrix up to a row permutation

$$\mathbf{PA} = \mathbf{LU}$$

, being **P** the permutation matrix.

- In this case $\mathbf{Ax} = \mathbf{b} \rightarrow \mathbf{P}^{-1}\mathbf{LUx} = \mathbf{b} \rightarrow \mathbf{LUx} = \mathbf{Pb}$, and the systems to be solved are

$$\mathbf{Ly} = \mathbf{Pb}$$

$$\mathbf{Ux} = \mathbf{y}$$

The pivoting technique

MATLAB and OCTAVE provides a built in algorithm for general LU-decomposition, it can be called by

```
[L U]=lu(A)
[L U P]=lu(A)
```

OCTAVE/MATLAB exercise

- For the matrix of the last example, $\mathbf{A} = \begin{bmatrix} 0 & 1 & 2 \\ 1 & 3 & 4 \\ 2 & 4 & 1 \end{bmatrix}$, obtain $\mathbf{L}, \mathbf{U}, \mathbf{P}$
- Obtain $\mathbf{A}$ back from $\mathbf{L}, \mathbf{U}, \mathbf{P}$
- Write `[L U]=lu(A)`, What is now the value of $\mathbf{L}$?

The pivoting technique

OCTAVE/MATLAB exercise

Write a MATLAB function that solves a linear system using *LU*-decomposition. Built-in function `lu()` can be used, so only the substitution must be programmed.

- Inputs: **A**, **b**
- Output: **x**, cpu time consumed

Check the behavior of the program generating matrices of different indexes using `A=rand(n,n)`, `B=tril(a,0)`, `C=triu(a,0)`

Accuracy of LU-methods

- Although a matrix satisfies the conditions for a *LU* decomposition, big errors might appear if pivot elements are near to zero. This errors are alleviated with the pivoting technique, but results might be yet rather unsatisfactory.
- Let be the system $\mathbf{Ax} = \mathbf{b}$ and let $\hat{\mathbf{x}}$ be the exact solution. Then, if the system is numerically solved, the relative error in the solution $\mathbf{x}$ can be defined as

$$\frac{\|\mathbf{x} - \hat{\mathbf{x}}\|}{\|\mathbf{x}\|}$$

- If the independent term is slightly modified (a rounding error when constructing $\delta\mathbf{b}$) then the new system becomes ($\mathbf{Ax} = \mathbf{b} + \delta\mathbf{b}$). It can be demonstrated that the relative error (difference between exact solution using $\delta\mathbf{b}$ or $\mathbf{b} + \delta\mathbf{b}$) is bounded by

$$\frac{\|\mathbf{x} - \hat{\mathbf{x}}\|}{\|\mathbf{x}\|} \leq K(\mathbf{A}) \frac{\|\delta\mathbf{b}\|}{\|\mathbf{b}\|}$$

Accuracy of LU-methods

- The value $K(\mathbf{A})$ is called *spectral condition number of the matrix* $\mathbf{A}$ and in the particular case of symmetric positive definite $K(\mathbf{A}) = \frac{|\lambda|_{max}}{|\lambda|_{min}}$ being $|\lambda|_{max}$ and $|\lambda|_{min}$ the maximum and minimum modulus of the eigenvalues of $\mathbf{A}$.
- $K(\mathbf{A}) \geq 1$, and the bigger its value, the faster the error amplification.
- When $K(\mathbf{A})$ is large, the matrix is $ill - conditioned$: the error in the solution might be large even with small errors in the system coefficients.
- The MATLAB/OCTAVE expression to obtain the *spectral condition number of the matrix* $\mathbf{A}$, is `cond(A)`.

Accuracy of LU-methods

Let be the systems (a) and (b)

$$(a)\begin{cases} x+y=2 \\ x+1.001y=2 \end{cases} \qquad (b)\begin{cases} x+y=2 \\ x+1.001y=2.001 \end{cases}$$

The solution of (a) is $x=2, y=0$, while the solution of (b) is $x=1, y=1$, so the coefficient matrix **A** is ill-condition ed

OCTAVE/MATLAB exercise

Obtain L and U using gauss substitution and solve the system
Obtain the condition number of the matrix of coefficients
Obtain the eigenvalues of the matrix by using `eig(A)`, which is the relation between condition number and the eigenvalues?

The MATLAB/OCTAVE command for solving linear systems

In order to solve directly a linear system $\mathbf{Ax} = \mathbf{b}$ in the better way in MATLAB/OCTAVE a new command can be used:

```
x=A\b
```

It will call different algorithms to solve the system depending on the type of matrix:

- If **A** is upper or lowe triangular, it will just use the substitution algorithm
- In case **A** is symmetric and with real positive diagonal elements will use Cholesky decomposition (a less general but faster method than LU)
- Otherwise, pivoting LU decomposition will be used

OCTAVE/MATLAB exercise

Solve again the different examples of previous exercises for random value of **b**

Outline

1. Introduction to programming
2. Linear systems of equations
   - Introduction
   - Direct methods
   - Iterative methods and sparse matrices
3. Non linear equations and systems
4. Ordinary differential equations (ODEs)

Iterative methods

- An iterative method for the solution of the linear system $\mathbf{Ax} = \mathbf{b}$ consists in setting up a sequence of $\{\mathbf{x}^{(k)}, k \geq 0\}$ that converges to the exact solution

$$\lim_{k \to \infty} \mathbf{x}^{(k)} = \mathbf{x} = \mathbf{A}^{-1}\mathbf{b}$$

, for any initial vector $\mathbf{x}^{(0)}$.

- A possible solution is the succession

$$\mathbf{x}^{(k+1)} = \mathbf{B}\mathbf{x}^{(k)} + \mathbf{g}$$

, where $\mathbf{B}$ is a suitable matrix and $\mathbf{g}$ is a vector depending on $\mathbf{b}$ and $\mathbf{A}$ (operating $\mathbf{g} = (\mathbf{I} - \mathbf{B})\mathbf{A}^{-1}\mathbf{b}$)

- A valid $\mathbf{B}$ should ensure convergency for any $\mathbf{x}^{(0)}$, and that is fulfilled in the case of symmetric, definite positive $\mathbf{B}$ with maximum absolute value of eigenvalues (spectral ratio)$\rho(\mathbf{B}) \leq 1$.

Iterative methods: Jacobi

If the diagonal entries of A are nonzero, let **D** be the diagonal matrix $\mathbf{D} = \mathrm{diag}(a_{11}, a_{22}, \cdots, a_{nn})$, the Jacobi iterative method consist on this series

$$\mathbf{x}^{(k+1)} = \mathbf{D}^{-1}(\mathbf{D} - \mathbf{A})\mathbf{x}^{(k)} + \mathbf{D}^{-1}\mathbf{b}$$

where $\mathbf{D}^{-1}$ is just the inverse of the diagonal terms.
If the matrix **A** is strictly diagonally dominant by row, then the Jacobi method converges.

OCTAVE/MATLAB exercise

Write a MATLAB function that provides the Jacobi iterative solution for a given n term. Apply the method to $\mathbf{A} = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 2 & 8 & 5 & -2 \\ 3 & 5 & 3 & 0 \\ 4 & -2 & 0 & 1 \end{bmatrix}$ and

$\mathbf{b} = \mathbf{0}$

Iterative methods: Other methods and when to stop

- The error of a given iterative solution $\mathbf{x}^{(k)}$ can be measured by the *residual*, $r = \|\mathbf{b} - \mathbf{A}\mathbf{x}^{(k)}\|$.
- Once fixed the desired precision, a given method needs a certain number of iterations to reach that error (converge).
- The convergence rate depends on the method and on the matrix!
- Several methods can be found on the literature: *Gauss-Seidel, Richardson, Gradient* and *Conjugate Gradient*
- In general, iterative methods are faster than direct methods to obtain a solution when the matrix $\mathbf{A}$ is dense.
- For *sparse banded* linear system with small band-with, direct methods are faster

Sparse matrices

- In Engineering and Physic, large systems of linear equations (thousands of unknowns) usually appear as a result of the discretization of a partial derivative equation as the heat conduction, elasticity equations, etc.
- Fortunately, the coefficient matrices of the systems are full of zero terms, what is known as sparse matrices
- The memory needed to save a sparse system in a clever way may be of $O(n)$ instead of the $O(n^2)$ needed when all the matrix terms are saved.
- In addition, direct methods as LU can be adapted for sparse matrices resulting in computational costs much smaller than $O(n^2)$

Sparse matrices

A sparse matrix can be defined in several ways. MATLAB uses a system consisting on saving 3 vectors instead a full *nxn* matrix. The *nz* non-zero terms are saved in a vector of dimension *m*, and the positions of that terms are saved in the original matrix by other 2 vectors.

- In order to store a full $\mathbf{A}_{nxn}$ matrix as a sparse matrix, the function `AA=sparse(A)`
- In order to define directly the sparse matrix, declare it first with the maximum numbers of *nz* non-zero terms `AA=spalloc(n,n,nz)`, and then add each non-zero term using $A(i,j) = a_{ij}$
- Any matrix function of MATLAB (i.e. `inv(A)`, `det(A)`, `...` can be made with a sparse stored matrix, and the algorithms used in that case will be addapted for sparse matrices

Sparse matrices

OCTAVE/MATLAB exercise

Create a matrix 100x100 with diagonal terms equal to 1. and with 1000 terms of random position i, j and value
Inverse the matrix using `inv(A)`, and check the time
Create a sparse matrix from the original one, invert it and check the time
Compare the time for both cases