Advanced Numerical Analysis

Lecture 7

Spring 2025

Daniel Kressner

Linear systems

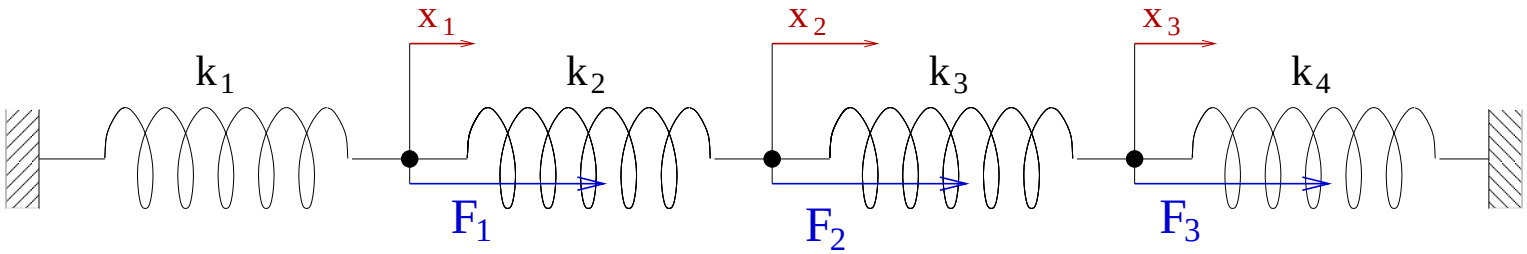
$$A\mathbf{x} = \mathbf{b}$$

with invertible matrix $A \in \mathbb{R}^{n \times n}$ and right-hand side vector $\mathbf{b} \in \mathbb{R}^n$.

$$A = \begin{pmatrix} a_{11} & a_{12} & \cdots & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & & \vdots \\ \vdots & \vdots & & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & \cdots & a_{nn} \end{pmatrix}, \quad \mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ \vdots \\ x_n \end{pmatrix}, \quad \mathbf{b} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ \vdots \\ b_n \end{pmatrix}.$$

A simple example

Configuration of coupled springs under some forces. Aim: Compute elongation of each spring.



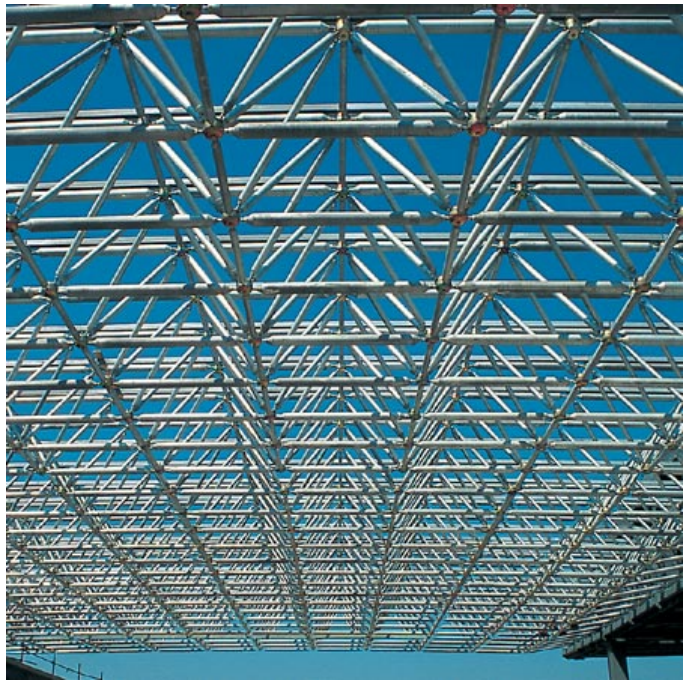
The equilibrium of forces at every node gives

$$\begin{cases} k_1 x_1 + k_2(x_1 - x_2) = F_1 \\ k_2(x_2 - x_1) + k_3(x_2 - x_3) = F_2 \\ k_3(x_3 - x_2) + k_4 x_3 = F_3 \end{cases}$$

$$\Rightarrow \quad \begin{cases} (k_1 + k_2)x_1 - k_2 x_2 = F_1 \\ -k_2 x_1 + (k_2 + k_3)x_2 - k_3 x_3 = F_2 \\ -k_3 x_2 + (k_3 + k_4)x_3 = F_3 \end{cases}$$

Linear system of 3 equations and 3 unknowns: $K\mathbf{x} = \mathbf{F}$

More complex applications



By imposing the equilibrium of forces at each node, we get a large linear system of equations (static case) or a large system of ordinary differential equations system (dynamic case). Hopeless to solve by hand. Need to use numerical algorithms!

- Numerical solution of linear systems.
- Discretization of differential equations (dynamic case).

More complex applications

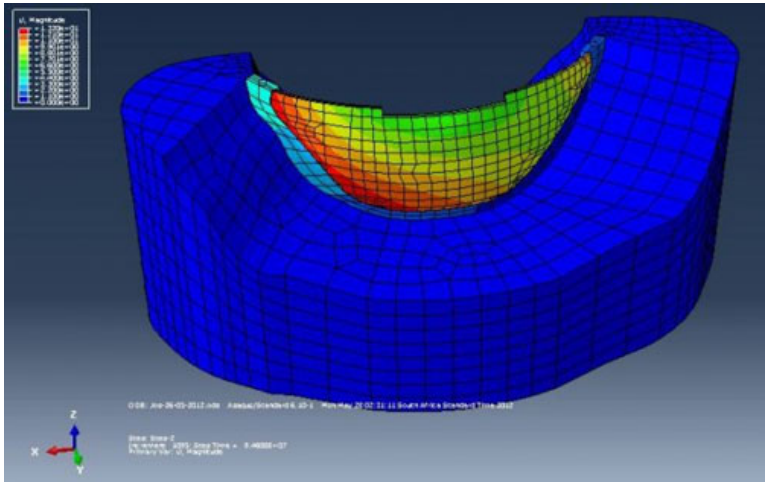


Photo credit: CoMSIRU, University of Cape Town

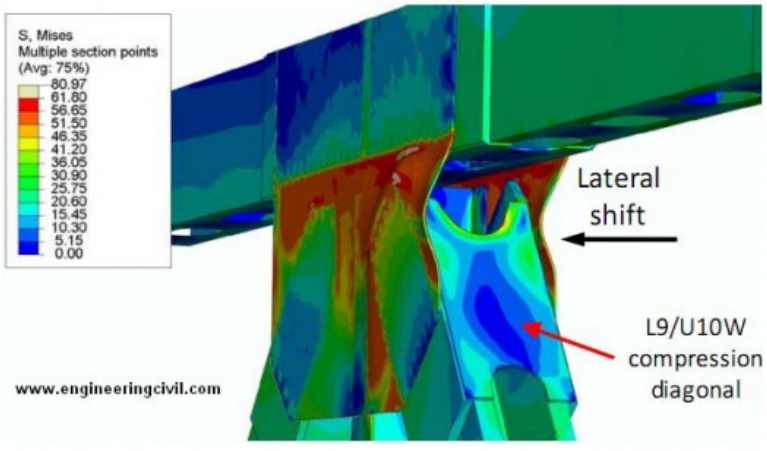


Figure 14. Indicating the lateral shift of the L9/U10W diagonal, with Mises stress contours.

Photo credit: www.engineeringcivil.com

The structure is divided in many small “cubes” (finite elements). By imposing the equilibrium of forces at each one of them a large linear system is obtained (it may have million of unknowns).
Dealing with such an application relies on the following ingredients:

- Approximation / Interpolation.
- Discretizations of differential equations.
- Numerical solution of linear systems.

Construction of LU factorization

LU factorization proceeds in $n-1$ steps, reducing the kth column of A in Step k.

Before Step k, the modified matrix A takes the form

$$A^{(k-1)} = \begin{pmatrix} a_{11} & a_{12} & a_{13} & \cdots & \cdots & a_{1n} \\ & a_{22}^{(1)} & a_{23}^{(1)} & \cdots & \cdots & a_{2n}^{(1)} \\ & & \ddots & \ddots & & \vdots \\ & & & a_{kk}^{(k-1)} & \cdots & a_{kn}^{(k-1)} \\ & & & \vdots & & \vdots \\ & & & a_{nk}^{(k-1)} & \cdots & a_{nn}^{(k-1)} \end{pmatrix}.$$

(For Step 1, we simply set $A^{(0)} := A$.)

Construction of LU factorization

For performing Step k, the coefficients

$$\ell_{ik} := \frac{a_{ik}^{(k-1)}}{a_{kk}^{(k-1)}}, \qquad i = k+1, \ldots, n,$$

are computed. Only possible if **pivot element** $a_{kk}^{(k-1)}$ is nonzero.

The multiplication of the matrix

$$L_k := \begin{pmatrix} 1 & & & & \\ & \ddots & & & \\ & & 1 & & \\ & & -\ell_{k+1,k} & 1 & \\ & & \vdots & & \ddots \\ & & -\ell_{n\,k} & & 1 \end{pmatrix}$$

with $A^{(k-1)}$ performs Step k and eliminates all entries below the $(k-1)$th diagonal entry.

Construction of LU factorization

The whole procedure is then repeated with the resulting matrix

$$A^{(k)} = L_k A^{(k-1)}. \tag{1}$$

After $n-1$ steps, we obtain

$$A^{(n-1)} = L_{n-1} L_{n-2} \cdots L_1 A^{(0)} = L_{n-1} L_{n-2} \cdots L_1 A.$$

This can be rewritten as $LU = A$, with

$$L := L_1^{-1} L_2^{-1} \cdots L_{n-1}^{-1}, \qquad U = A^{(n-1)}.$$

Note that U is upper triangular by construction. The factors L_k^{-1} of the matrix L are given by

$$L_k^{-1} = \begin{pmatrix} 1 & & & & \\ & \ddots & & & \\ & & 1 & & \\ & & \ell_{k+1,k} & 1 & \\ & & \vdots & & \ddots \\ & & \ell_{n\,k} & & & 1 \end{pmatrix},$$

which can be verified by checking $L_k^{-1} L_k = I$.

Construction of LU factorization

Due to the special structure of L_k^{-1}, the sub-diagonal entries of L are obtained from collecting the subdiagonal entries of all L_k^{-1}:

$$L = L_1^{-1} L_2^{-1} \cdots L_{n-1}^{-1} = \begin{pmatrix} 1 & & & & & \\ \ell_{21} & \ddots & & & & \\ \ell_{31} & \ddots & 1 & & & \\ \vdots & & \ell_{k+1,k} & 1 & & \\ \vdots & & \vdots & \ddots & \ddots & \\ \ell_{n1} & \cdots & \ell_{nk} & \cdots & \ell_{n,n-1} & 1 \end{pmatrix}.$$

Abstract form of LU factorization

Algorithm 4.7

$A^{(0)} := A$

for $k = 1, \ldots, n-1$ **do**

Determine matrix L_k by computing the coefficients

$$\ell_{ik} := \frac{a_{ik}^{(k-1)}}{a_{kk}^{(k-1)}}, \qquad i = k+1, \ldots, n.$$

Set $A^{(k)} := L_k A^{(k-1)}$.

end for

- Naive implementation of matrix product $L_k A^{(k-1)} \rightsquigarrow O(n^4)$ complexity.

Practical form of LU factorization

Changes:

- Exploit structure of L_k when performing
- Update A in place.

Algorithm 4.9

Set $L := I_n$.
for $k = 1, \ldots, n-1$ **do**
  for $i = k+1, \ldots, n$ **do**
    $\ell_{ik} \leftarrow \dfrac{a_{ik}}{a_{kk}}$
    for $j = k+1, \ldots, n$ **do**
      $a_{ij} \leftarrow a_{ij} - \ell_{ik} a_{kj}$
    end for
  end for
end for
Set U to upper triangular part of A.

Number of elementary operations (flops) of Algorithm 4.9:

$$\sum_{k=1}^{n-1} \big(1 + 2(n-k)\big)(n-k) = \frac{2}{3}n^3 - \frac{1}{2}n^2 - \frac{1}{6}n = \frac{2}{3}n^3 + O(n^2).$$

Python implementation

```
import numpy as np
def mylu(A):
    n = A.shape[0]
    L = np.eye(n)
    for k in range(n):
        L[k+1:n, k] = A[k+1:n, k] / A[k,k]
        A[k+1:n, k+1:n] = A[k+1:n, k+1:n]
          - np.outer(L[k+1:n, k], A[k, k+1:n])
    U = np.triu(A)
    return L, U
```

Python implementation

```
import numpy as np, scipy as sp
A = np.array([[-0.370,  0.050,  0.050,  0.070],
              [ 0.050, -0.116,  0.000,  0.050],
              [ 0.050,  0.000, -0.116,  0.050],
              [ 0.070,  0.050,  0.050, -0.202]])
b = np.array([[-2], [0], [0], [0]])
L, U = mylu(A)
y = sp.linalg.solve_triangular(L, b, lower=True,
                                     unit_diagonal=True)
x = sp.linalg.solve_triangular(U, y)
```

This produces the output

```
x =
    8.1172
    5.9893
    5.9893
    5.7779
```