Minicourse:
Solvers in COMSOL Multiphysics

Martin Kožíšek
kozisek@humusoft.cz

Solvers in COMSOL Multiphysics

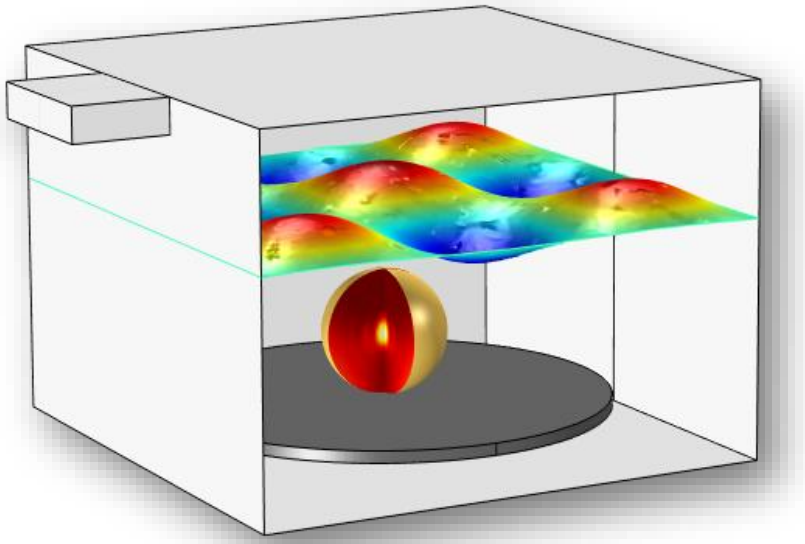
1. How to solve multiphysics models
   - One-way Coupling vs. Two-way Coupling
   - Fully Coupled vs. Segregated Approach
   - Comparison
2. How to solve FEM system of algebraic equations
   - Direct Solver
   - Iterative Solver
   - Comparision
3. Conclusions

How to Solve Multiphysics Models

One-way vs. Two-way Coupling of Multiple Physical Phenomena

One-Way Coupling

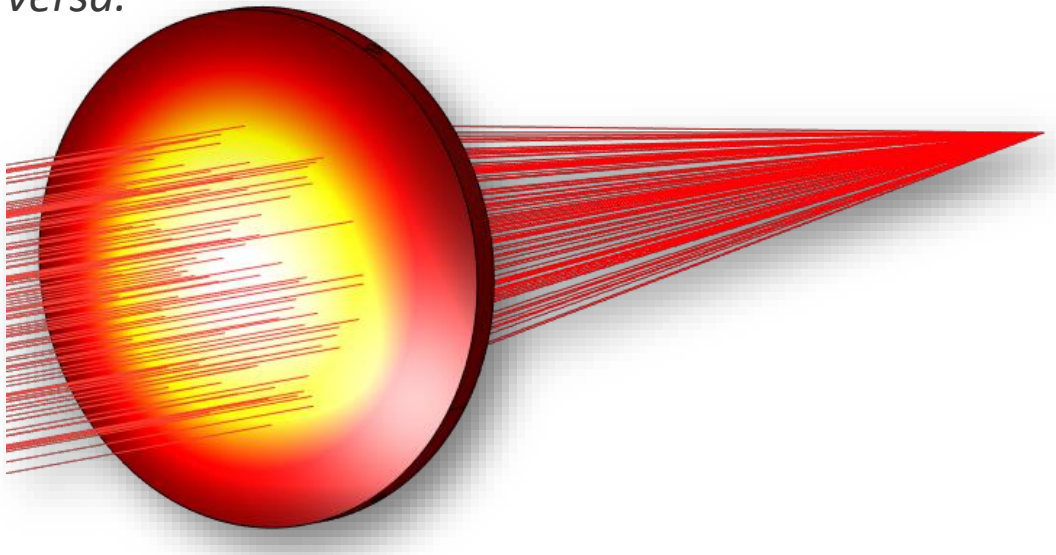
„The physical phenomena do not affect each other, or their mutual influence can be neglected.“



Example: Simulation of microwave heating of a potato. In the first step the electric field is computed, from which the generated heat is then derived. The heat does not feed back to change the field.

Two-Way (or Both-directional) Coupling

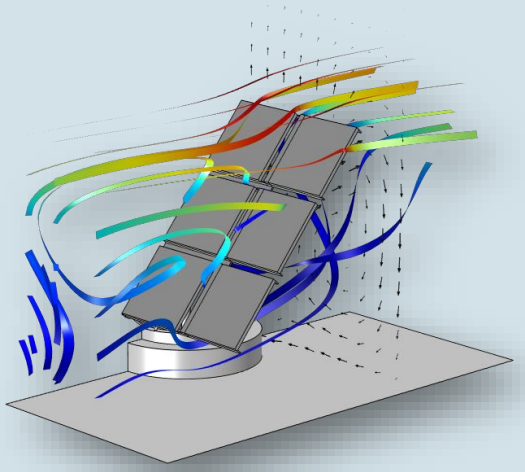
„The physical phenomena affect each other significantly — each one influences the other and vice versa.“



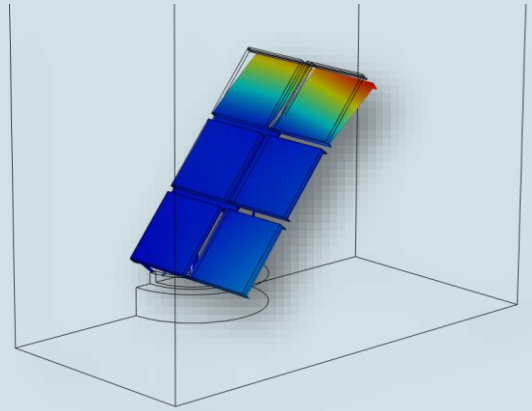
Example: Thermal deformation of a lens during laser beam passage. The beam heats the lens, which deforms. The deformation in turn changes the optical properties and the beam path.

One-way Coupling

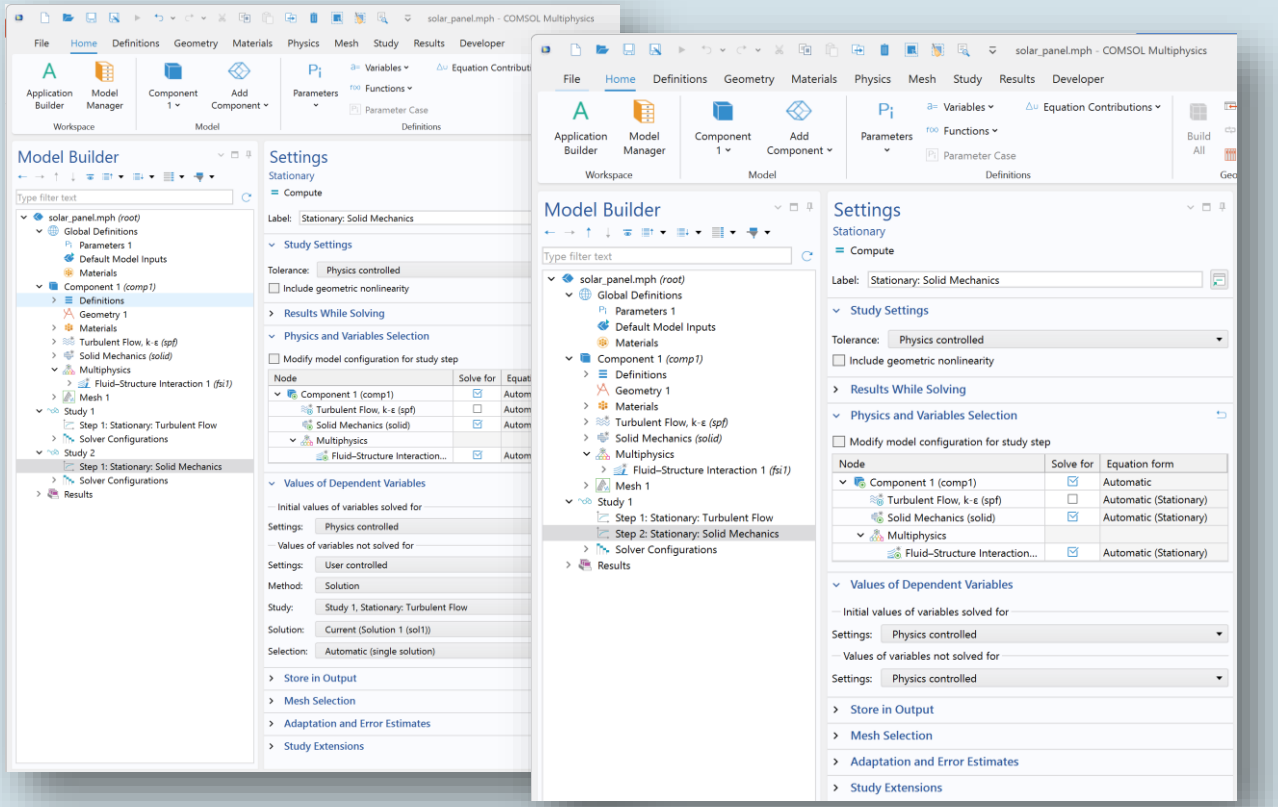
- Consider a model of a solar panel exposed to a windstorm. The airflow causes stress and slight deformation of the panel. We are interested in the panel's deformation in the steady state.
- How to set this?
  - Multiphysics node
  - Two studies
  - Two study steps in one study



Step 1: Velocity and pressure distribution in the fluid flow field



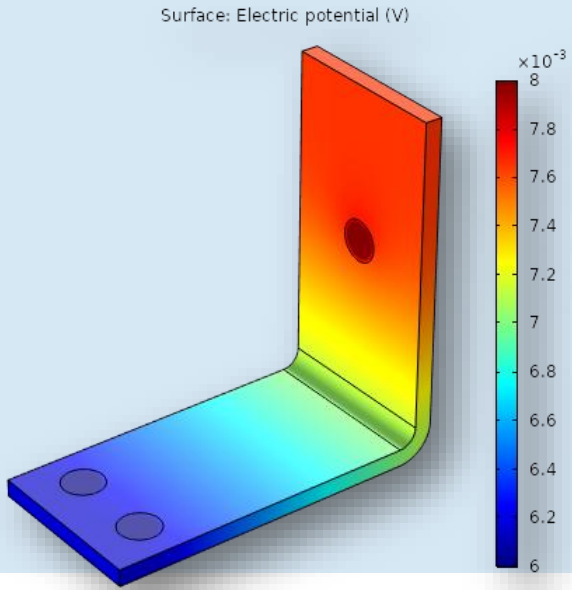
Step 2: Mechanical (elastic) deformations caused by fluid pressure acting on the panel



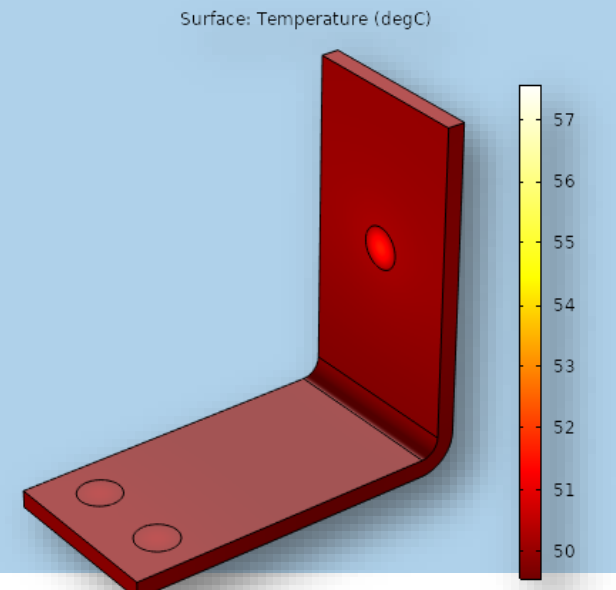
Two-way Coupling

Consider a component under electrical voltage with electric current flowing through it. The component heats up due to Joule heating and deforms. We are interested in the steady state.

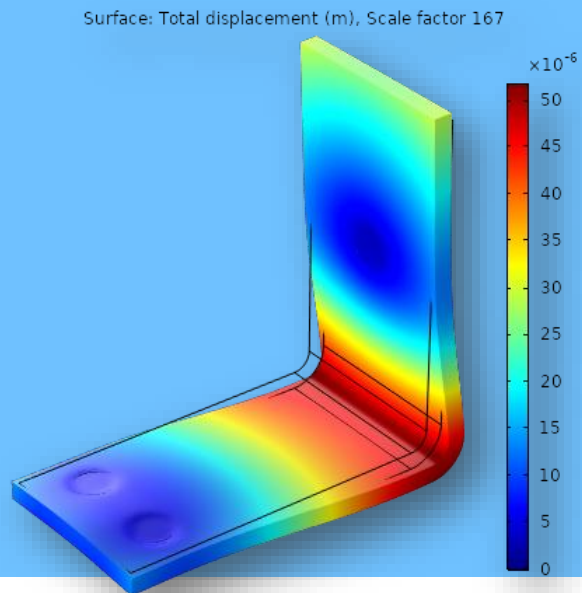
Electric potential distribution V



Temperature distribution T

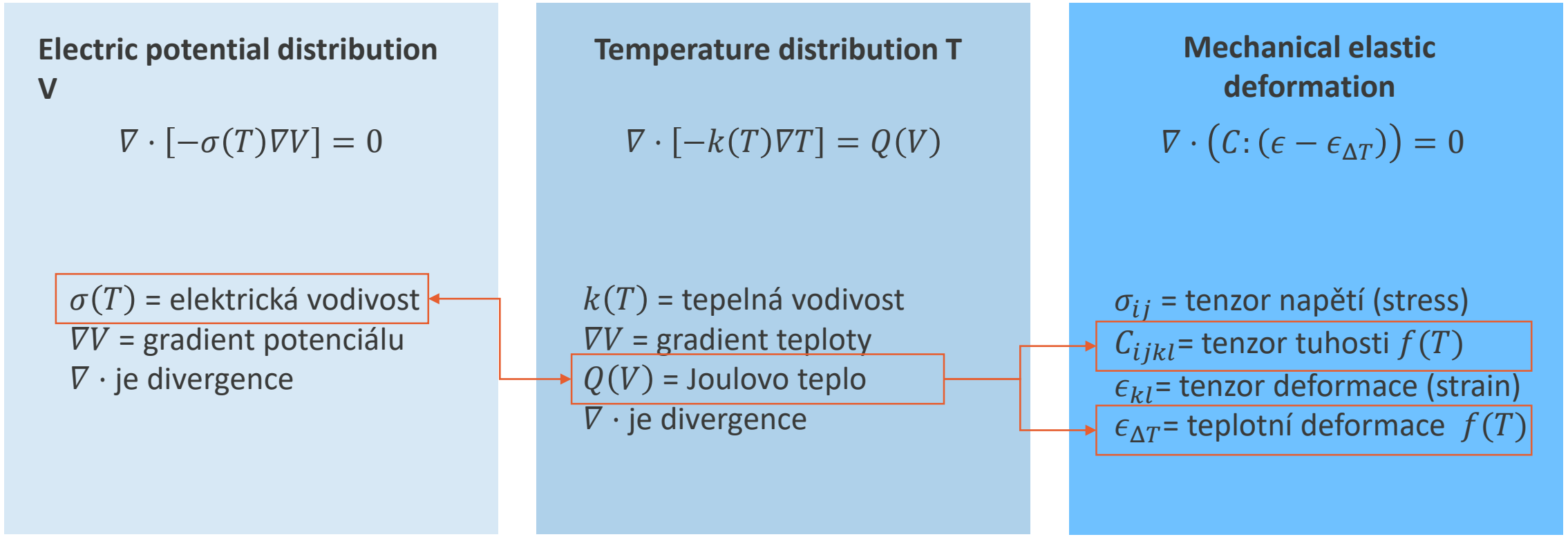


Mechanical elastic deformation



Two-way Coupling

Consider a component under electrical voltage with electric current flowing through it. The component heats up due to Joule heating and deforms. We are interested in the steady state.



Two-way Coupling

Consider a component under electrical voltage with electric current flowing through it. The component heats up due to Joule heating and deforms. We are interested in the steady state.

Electric potential distribution V

$$\nabla \cdot [-\sigma(T)\nabla V] = 0$$

for N degrees of freedom (DOF):

$$\begin{bmatrix} K_{V_{11}}(\tilde{T}) & \dots & K_{V_{1N}}(\tilde{T}) \\ \vdots & \ddots & \vdots \\ K_{V_{N1}}(\tilde{T}) & \dots & K_{V_{NN}}(\tilde{T}) \end{bmatrix} \begin{bmatrix} \tilde{V}_1 \\ \vdots \\ \tilde{V}_N \end{bmatrix} = \begin{bmatrix} b_{V_1} \\ \vdots \\ b_{V_N} \end{bmatrix}$$

Temperature distribution T

$$\nabla \cdot [-k(T)\nabla T] = Q(V)$$

for M degrees of freedom (DOF):

$$\begin{bmatrix} K_{T_{11}}(\tilde{T}) & \dots & K_{T_{1M}}(\tilde{T}) \\ \vdots & \ddots & \vdots \\ K_{T_{M1}}(\tilde{T}) & \dots & K_{T_{MM}}(\tilde{T}) \end{bmatrix} \begin{bmatrix} \tilde{T}_1 \\ \vdots \\ \tilde{T}_M \end{bmatrix} = \begin{bmatrix} b_{T_1}(\tilde{V}) \\ \vdots \\ b_{T_M}(\tilde{V}) \end{bmatrix}$$

Mechanical elastic deformation

$$\nabla \cdot \left(C : (\epsilon - \epsilon_{\Delta T})\right) = 0$$

for P degrees of freedom (DOF):

$$\begin{bmatrix} K_{d_{11}}(\tilde{T}) & \dots & K_{d_{1P}}(\tilde{T}) \\ \vdots & \ddots & \vdots \\ K_{d_{P1}}(\tilde{T}) & \dots & K_{d_{PP}}(\tilde{T}) \end{bmatrix} \begin{bmatrix} \tilde{d}_1 \\ \vdots \\ \tilde{d}_P \end{bmatrix} = \begin{bmatrix} b_{d_1}(\tilde{T}) \\ \vdots \\ b_{d_P}(\tilde{T}) \end{bmatrix}$$

How to Solve the Two-way Coupling System?

The Fully Coupled Two-way Coupling

Electric potential distribution V

$$\begin{bmatrix} K_{V_{11}}(\tilde{T}) & \dots & K_{V_{1N}}(\tilde{T}) \\ \vdots & \ddots & \vdots \\ K_{V_{N1}}(\tilde{T}) & \dots & K_{V_{NN}}(\tilde{T}) \end{bmatrix} \begin{bmatrix} \tilde{V}_1 \\ \vdots \\ \tilde{V}_N \end{bmatrix} = \begin{bmatrix} b_{V_1} \\ \vdots \\ b_{V_N} \end{bmatrix}$$

Temperature distribution T

$$\begin{bmatrix} K_{T_{11}}(\tilde{T}) & \dots & K_{T_{1M}}(\tilde{T}) \\ \vdots & \ddots & \vdots \\ K_{T_{M1}}(\tilde{T}) & \dots & K_{T_{MM}}(\tilde{T}) \end{bmatrix} \begin{bmatrix} \tilde{T}_1 \\ \vdots \\ \tilde{T}_M \end{bmatrix} = \begin{bmatrix} b_{T_1}(\tilde{V}) \\ \vdots \\ b_{T_M}(\tilde{V}) \end{bmatrix}$$

Mechanical elastic deformation

$$\begin{bmatrix} K_{d_{11}}(\tilde{T}) & \dots & K_{d_{1P}}(\tilde{T}) \\ \vdots & \ddots & \vdots \\ K_{d_{P1}}(\tilde{T}) & \dots & K_{d_{PP}}(\tilde{T}) \end{bmatrix} \begin{bmatrix} \tilde{d}_1 \\ \vdots \\ \tilde{d}_P \end{bmatrix} = \begin{bmatrix} b_{d_1}(\tilde{T}) \\ \vdots \\ b_{d_P}(\tilde{T}) \end{bmatrix}$$

All physical effects are solved in a single system. Everything depends on everything.

The Newton–Raphson method is used for the calculation.

$$\begin{bmatrix} K_{V_{11}}(\tilde{T}) & \dots & K_{V_{1N}}(\tilde{T}) & 0 & 0 \\ \vdots & \ddots & \vdots & & \\ K_{V_{N1}}(\tilde{T}) & \dots & K_{V_{NN}}(\tilde{T}) & & \\ 0 & & & K_{T_{11}}(\tilde{T}) & \dots & K_{T_{1M}}(\tilde{T}) & 0 \\ & & & \vdots & \ddots & \vdots & \\ & & & K_{T_{M1}}(\tilde{T}) & \dots & K_{T_{MM}}(\tilde{T}) & \\ 0 & & 0 & & & K_{d_{11}}(\tilde{T}) & \dots & K_{d_{1P}}(\tilde{T}) \\ & & & & & \vdots & \ddots & \vdots \\ & & & & & K_{d_{P1}}(\tilde{T}) & \dots & K_{d_{PP}}(\tilde{T}) \end{bmatrix} \begin{bmatrix} \tilde{V}_1 \\ \vdots \\ \tilde{V}_N \\ \tilde{T}_1 \\ \vdots \\ \tilde{T}_M \\ \tilde{d}_1 \\ \vdots \\ \tilde{d}_P \end{bmatrix} = \begin{bmatrix} b_{V_1} \\ \vdots \\ b_{V_N} \\ b_{T_1}(\tilde{V}) \\ \vdots \\ b_{T_N}(\tilde{V}) \\ b_{d_1}(\tilde{T}) \\ \vdots \\ b_{d_P}(\tilde{T}) \end{bmatrix}$$

The Fully Coupled Two-way Coupling

- Simplified notation of the „fully coupled“ system

$$\mathbf{f} = \begin{bmatrix} K_V(\tilde{T}) & 0 & 0 \\ 0 & K_T(\tilde{T}) & 0 \\ 0 & 0 & K_d(\tilde{T}) \end{bmatrix} \begin{bmatrix} \tilde{V} \\ \tilde{T} \\ \tilde{d} \end{bmatrix} - \begin{bmatrix} b_V \\ b_T(\tilde{V}) \\ b_d(\tilde{T}) \end{bmatrix} = 0$$

- Jacobian matrix required by the Newton-Raphson method (searching for **f = 0** along direction **f‘**)

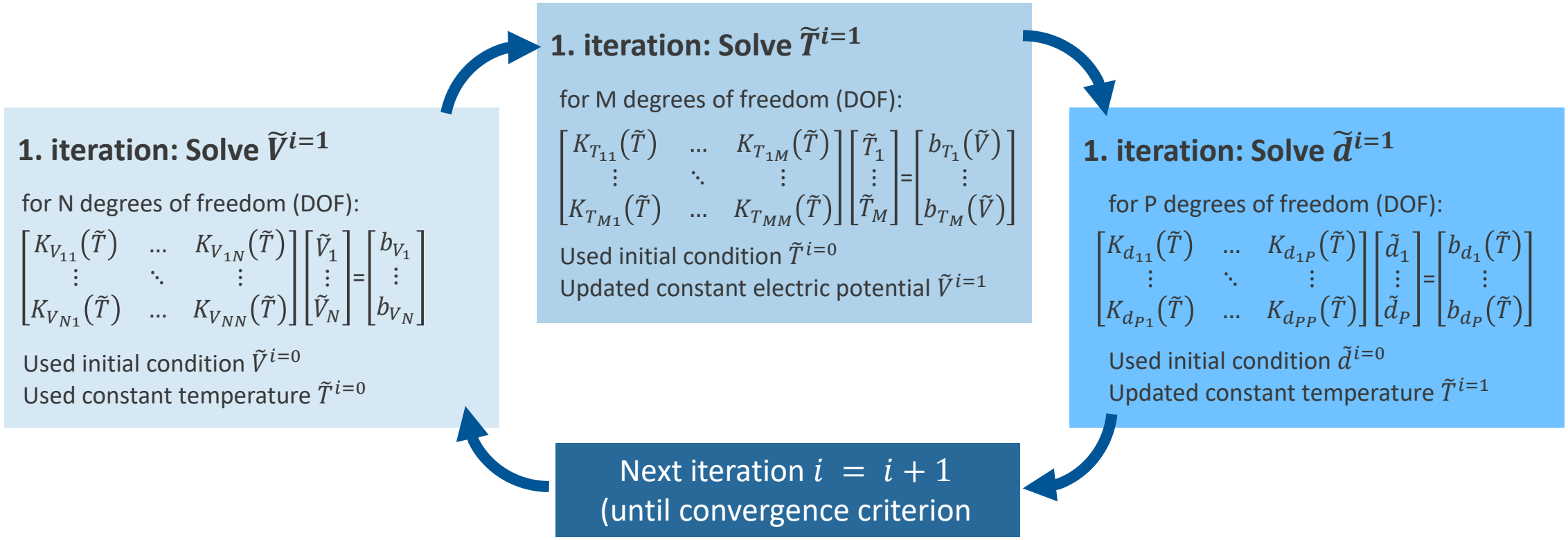
$$\mathbf{f}' = \begin{bmatrix} K_V & K_{V,\tilde{T}} & 0 \\ -b_{T,\tilde{V}} & K_T & 0 \\ 0 & \left(K_{d,\tilde{T}} - b_{d,\tilde{T}}\right) & K_d \end{bmatrix}, \text{ where } K_{V,\tilde{T}} = \frac{\delta K_V(\tilde{T})}{\delta \tilde{T}}$$

The Jacobian matrix is non-symmetric and can be difficult to solve!

In the case of strongly nonlinear dependencies, computing the Jacobian matrix can be a time- and memory-intensive operation.

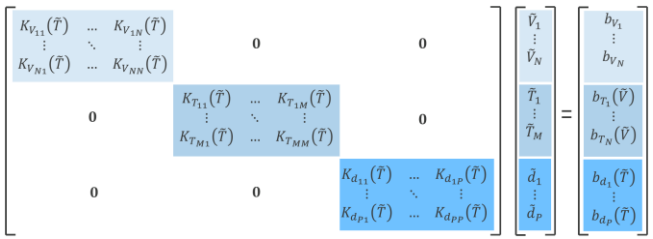
The Segregated Two-way Coupling

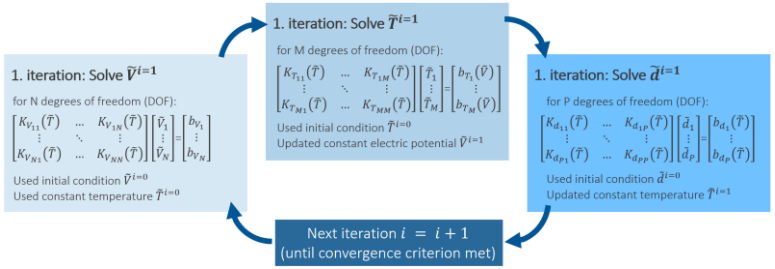
- Solves individual physical effects sequentially and separately, each time for a constant initial condition.
- The computation consists of simpler subsequences that have the nature of a linear problem.



Comparison: Fully Coupled vs. Segregated Approach

	Fully Coupled	Segregated
Computational cost per one iteration	High	Low
Required number of iterations	Low	High
Total computational time	Usually higher	Lower
RAM memory demand	Higher	Lower
Setup complexity	Low	Higher





How to Solve FEM System of Algebraic Equations

Micro-credential Course: Introduction to FEM

1. PDE:

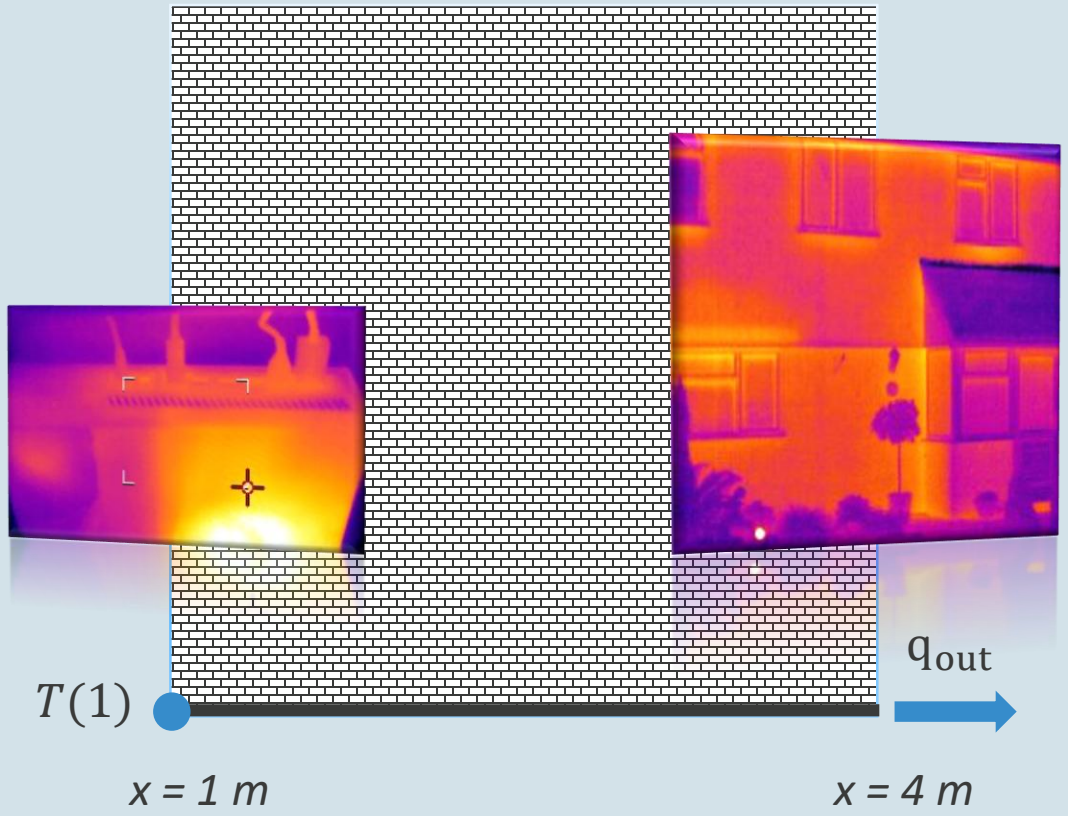
$$k\frac{\partial^2 T(x)}{\partial x^2} = 0$$

2. Weak Form

3. Discretization

4. System of algebraic equations:

$$\begin{bmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 1 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ -10 \end{bmatrix}$$



From chapter 1: Introduction to the Finite Element Method
https://www.vut.cz/vav/transfer/firmy/vzdelavani/mikrocertifikat

Manual Solution

- System of algebraic equations
- Forward: expressing variables
- Backward: substituting values
- Or command „A\x“ in MATLAB (using Livelink for MATLAB)

Matrix form $A\,x = b$:

$$A = \begin{bmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 1 \end{bmatrix} \qquad x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \qquad b = \begin{bmatrix} 0 \\ 0 \\ -10 \end{bmatrix}$$

System of equations:

$$\begin{aligned} 2x_1 - x_2 &= 0 \\ -x_1 + 2x_2 - x_3 &= 0 \\ -x_2 + x_3 &= -10 \end{aligned}$$

Forward:

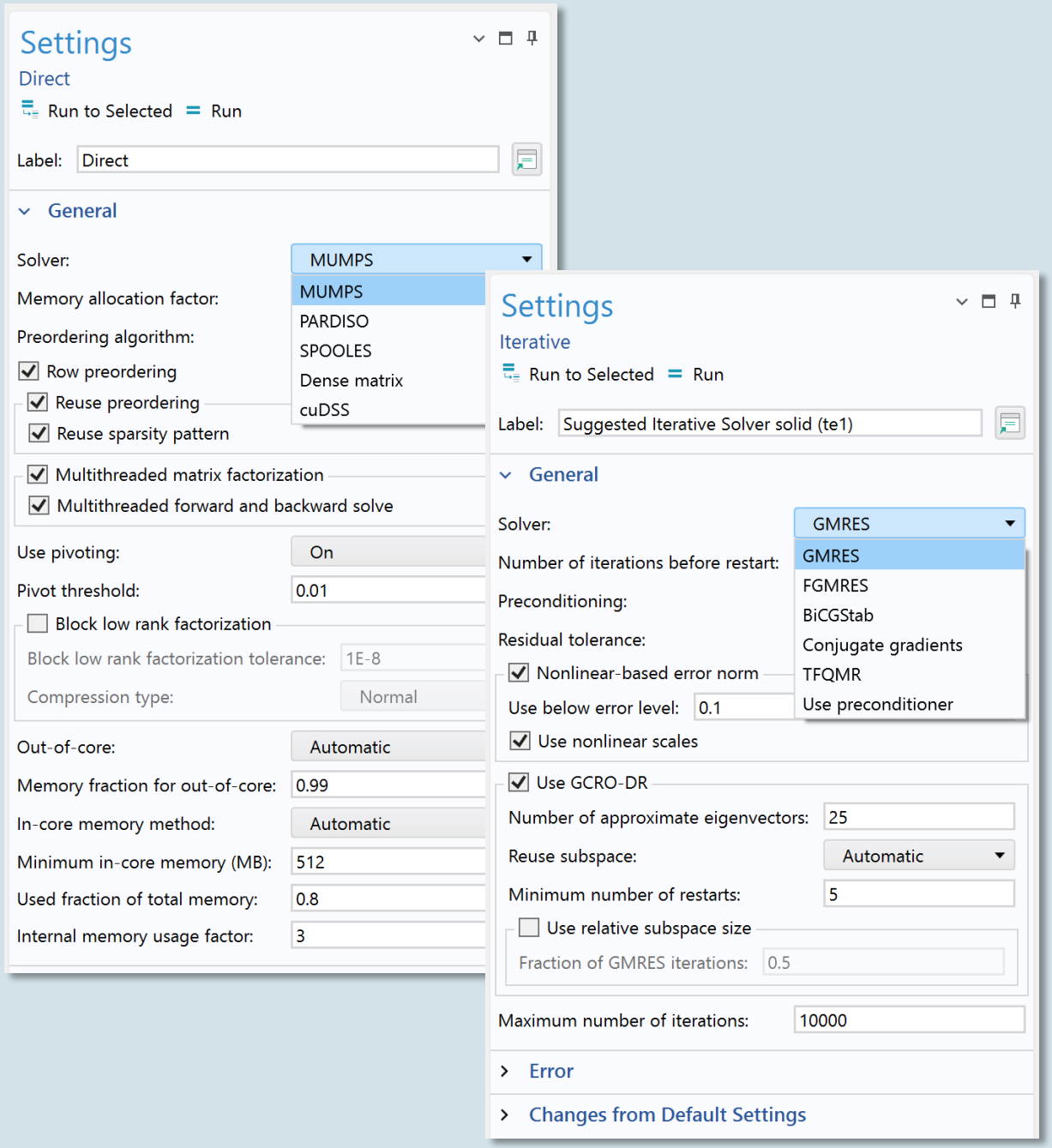
$$\begin{aligned} x_1 &= x_2/2 \\ x_2 &= 2x_3/3 \\ x_3 &= -30 \end{aligned}$$

Backward:

$$\begin{aligned} x_1 &= -10 \\ x_2 &= -20 \\ x_3 &= -30 \end{aligned}$$

Available Solvers in COMSOL Multiphysics

- Direct
- Iterative



Direct Solvers – LU Factorization

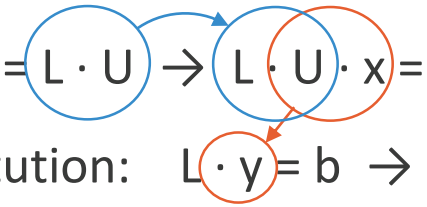
Concept of LU Factorization

1. System of algebraic equations $A \cdot x = b$
2. Decompose: $A = L \cdot U \rightarrow L \cdot U \cdot x = b$
3. Forward substitution: $L \cdot y = b \rightarrow$ solve for y
4. Back substitution: $U \cdot x = y \rightarrow$ solve for x

Direct vs. Iterative Solvers

Direct Solvers

1. System of algebraic equations $A \cdot x = b$
2. Decompose: $A = L \cdot U \rightarrow L \cdot U \cdot x = b$
3. Forward substitution: $L \cdot y = b \rightarrow$ solve for y
4. Back substitution: $U \cdot x = y \rightarrow$ solve for x



Iterative Solvers

1. Start with initial guess x_0 (usually $x_0 = 0$)
2. Compute residual: $r = b - A \cdot x$
3. Update x “somehow” using residual information
4. Repeat until $\|r\| <$ tolerance

How to compute L and U?

- Lower triangular matrix: L
- Upper triangular matrix: U
- The product of matrices L and U yields all 9 unknowns

Matrix $A = L \cdot U$

$$A = \begin{bmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ L_{21} & 1 & 0 \\ L_{31} & L_{32} & 1 \end{bmatrix} \cdot \begin{bmatrix} U_{11} & U_{12} & U_{13} \\ 0 & U_{22} & U_{23} \\ 0 & 0 & U_{33} \end{bmatrix}$$

Matrix multiplication:

$$A_{11} = 2 = 1 \cdot U_{11} + 0 \cdot 0 + 0 \cdot 0 \quad \Rightarrow U_{11} = 2$$

How to compute L and U?

- Lower triangular matrix: L
- Upper triangular matrix: U
- The product of matrices L and U yields all 9 unknowns

Matrix $A = L \cdot U$

$$A = \begin{bmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ L_{21} & 1 & 0 \\ L_{31} & L_{32} & 1 \end{bmatrix} \cdot \begin{bmatrix} U_{11} & U_{12} & U_{13} \\ 0 & U_{22} & U_{23} \\ 0 & 0 & U_{33} \end{bmatrix}$$

Matrix multiplication:

$A_{11} = 2 = 1 \cdot U_{11} + 0 \cdot 0 + 0 \cdot 0 \quad \Rightarrow U_{11} = 2$

$A_{12} = -1 = 1 \cdot U_{12} + 0 \cdot U_{22} + 0 \cdot 0 \quad \Rightarrow U_{12} = -1$

How to compute L and U?

- Lower triangular matrix: L
- Upper triangular matrix: U
- The product of matrices L and U yields all 9 unknowns

Matrix $A = L \cdot U$

$$A = \begin{bmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ L_{21} & 1 & 0 \\ L_{31} & L_{32} & 1 \end{bmatrix} \cdot \begin{bmatrix} U_{11} & U_{12} & U_{13} \\ 0 & U_{22} & U_{23} \\ 0 & 0 & U_{33} \end{bmatrix}$$

Matrix multiplication:

$$A_{11} = 2 = 1 \cdot U_{11} + 0 \cdot 0 + 0 \cdot 0 \quad \Rightarrow U_{11} = 2$$
$$A_{12} = -1 = 1 \cdot U_{12} + 0 \cdot U_{22} + 0 \cdot 0 \quad \Rightarrow U_{12} = -1$$
$$A_{13} = 0 = 1 \cdot U_{13} + 0 \cdot U_{23} + 0 \cdot U_{33} \quad \Rightarrow U_{13} = 0$$

How to compute L and U?

- Lower triangular matrix: L
- Upper triangular matrix: U
- The product of matrices L and U yields all 9 unknowns

Matrix $A = L \cdot U$

$$A = \begin{bmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ L_{21} & 1 & 0 \\ L_{31} & L_{32} & 1 \end{bmatrix} \cdot \begin{bmatrix} U_{11} & U_{12} & U_{13} \\ 0 & U_{22} & U_{23} \\ 0 & 0 & U_{33} \end{bmatrix}$$

Matrix multiplication:

$$A_{11} = 2 = 1 \cdot U_{11} + 0 \cdot 0 + 0 \cdot 0 \quad \Rightarrow U_{11} = 2$$
$$A_{12} = -1 = 1 \cdot U_{12} + 0 \cdot U_{22} + 0 \cdot 0 \quad \Rightarrow U_{12} = -1$$
$$A_{13} = 0 = 1 \cdot U_{13} + 0 \cdot U_{23} + 0 \cdot U_{33} \quad \Rightarrow U_{13} = 0$$
$$A_{21} = -1 = L_{21} \cdot U_{11} + 1 \cdot 0 + 0 \cdot 0 \quad \Rightarrow L_{21} = -1/2$$

How to compute L and U?

- Lower triangular matrix: L
- Upper triangular matrix: U
- The product of matrices L and U yields all 9 unknowns

Matrix $A = L \cdot U$

$$A = \begin{bmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ L_{21} & 1 & 0 \\ L_{31} & L_{32} & 1 \end{bmatrix} \cdot \begin{bmatrix} U_{11} & U_{12} & U_{13} \\ 0 & U_{22} & U_{23} \\ 0 & 0 & U_{33} \end{bmatrix}$$

Matrix multiplication:

$$\begin{aligned}
A_{11} &= 2 = 1 \cdot U_{11} + 0 \cdot 0 + 0 \cdot 0 && \Rightarrow U_{11} = 2 \\
A_{12} &= -1 = 1 \cdot U_{12} + 0 \cdot U_{22} + 0 \cdot 0 && \Rightarrow U_{12} = -1 \\
A_{13} &= 0 = 1 \cdot U_{13} + 0 \cdot U_{23} + 0 \cdot U_{33} && \Rightarrow U_{13} = 0 \\
A_{21} &= -1 = L_{21} \cdot U_{11} + 1 \cdot 0 + 0 \cdot 0 && \Rightarrow L_{21} = -1/2 \\
A_{22} &= 2 = L_{21} \cdot U_{12} + 1 \cdot U_{22} + 0 \cdot 0 && \Rightarrow U_{22} = 3/2 \\
A_{23} &= -1 = L_{21} \cdot U_{13} + 1 \cdot U_{23} + 0 \cdot U_{33} && \Rightarrow U_{23} = -1 \\
A_{31} &= 0 = L_{31} \cdot U_{11} + L_{32} \cdot 0 + 1 \cdot 0 && \Rightarrow L_{31} = 0 \\
A_{32} &= -1 = L_{31} \cdot U_{12} + L_{32} \cdot U_{22} + 1 \cdot 0 && \Rightarrow L_{32} = -2/3 \\
A_{33} &= 1 = L_{31} \cdot U_{13} + L_{32} \cdot U_{23} + 1 \cdot U_{33} && \Rightarrow U_{33} = 1
\end{aligned}$$

Result:

$$L = \begin{bmatrix} 1 & 0 & 0 \\ -1/2 & 1 & 0 \\ 0 & -2/3 & 1 \end{bmatrix} \qquad U = \begin{bmatrix} 2 & -1 & 0 \\ 0 & 3/2 & -1 \\ 0 & 0 & 1/3 \end{bmatrix}$$

LU Factorization: Forward Substitution

- Substitution $L \cdot y = b$
- Solve for y

Matrix form $L\,y = b$:

$$L = \begin{bmatrix} 1 & 0 & 0 \\ -1/2 & 1 & 0 \\ 0 & -2/3 & 1 \end{bmatrix} \quad y = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} \quad b = \begin{bmatrix} 0 \\ 0 \\ -10 \end{bmatrix}$$

Easy to solve:

$$y_1 = 0$$
$$y_2 = 0$$
$$y_3 = -10$$

LU Factorization: Backward Substitution

- Substitution $U \cdot x = y$
- Solve for x
- This is the direct solution!

Matrix form $U\,x = y$:

$$U = \begin{bmatrix} 2 & -1 & 0 \\ 0 & 3/2 & -1 \\ 0 & 0 & 1/3 \end{bmatrix} \quad x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \quad b = \begin{bmatrix} 0 \\ 0 \\ -10 \end{bmatrix}$$

Easy to solve:

$x_1 = -10$
$x_2 = -20$
$x_3 = -30$

LU Factorization: Requirements

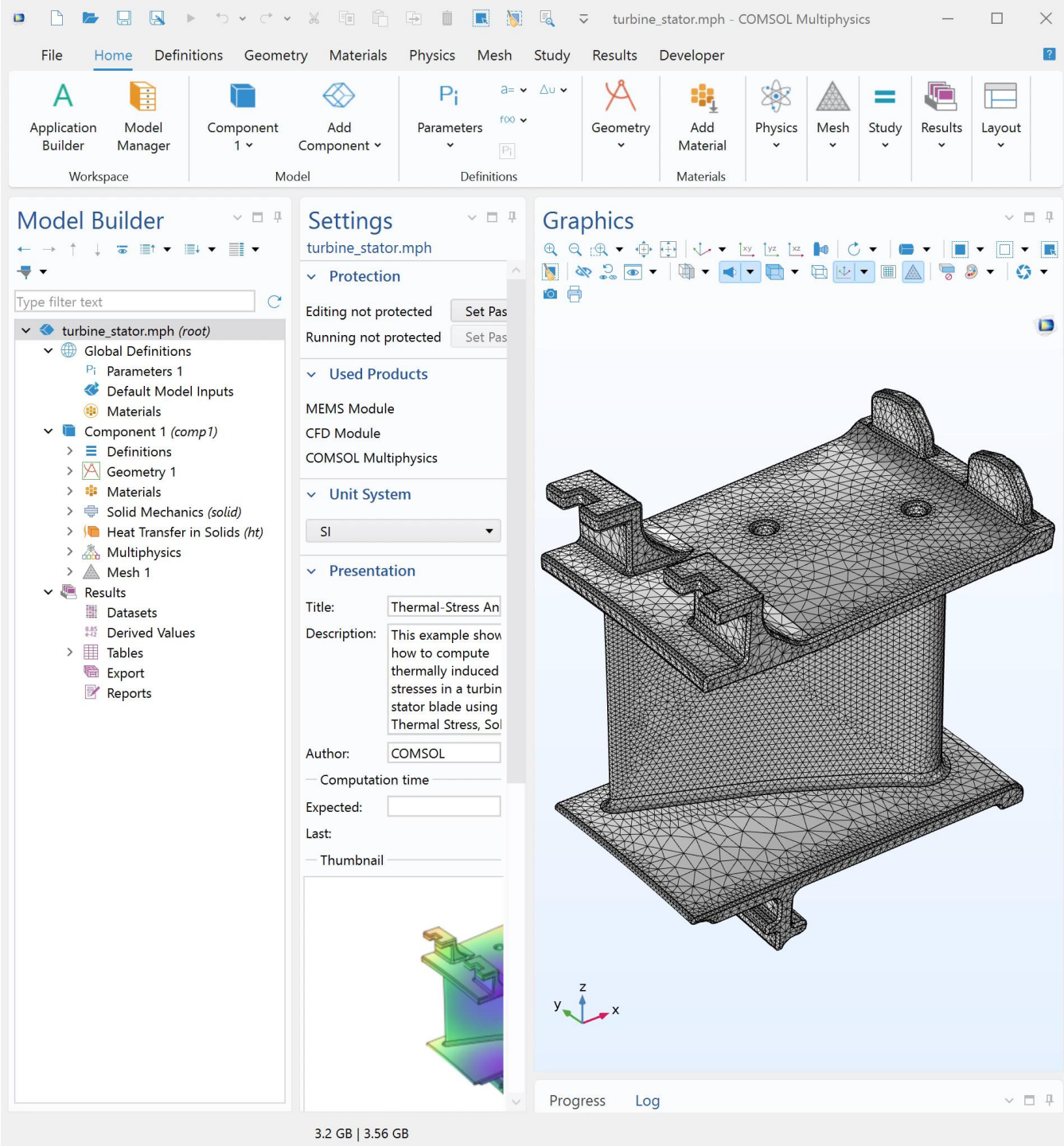
- Requirements for $Ax = b$, matrix A must be :
  - non-singular ($\det(A) \neq 0$),
  - square,
  - and well-conditioned

- What is the Condition Number?
  - Measures sensitivity of x to perturbations in A or b
  - $\kappa(A) = \|A\| \cdot \|A^{-1}\|$ (any consistent matrix norm)
  - 2-norm: $\kappa_2(A) = \sigma_{max} / \sigma_{min}$ (largest / smallest singular value of $\sigma_i = \sqrt{\lambda(A^T A)}$)
  - $\kappa \approx 1 \rightarrow$ well-conditioned; $\kappa \gg 1 \rightarrow$ ill-conditioned

Complexity of LU Factorization

- N – Number of DOFs
- Memory to store LU factors
- 2D / 3D – difference in matrix density (rise in non-zero matrix elements)
- In 3D: *„2x more N requires $2^{(4/3)}$x more memory“*

	2D	3D
Memory	$O(N \log N)$	$O(N^{4/3})$
CPU time	$O(N^{3/2})$	$O(N^2)$



LU Factorization Based Direct Solvers in COMSOL Multiphysics

- PARDISO (fastest for shared memory parallelization – single processor workstation)
- MUMPS (only direct solver for distributed memory parallelization – cluster, MPI)
- SPOOLES (slowest for shared memory parallelization, but most memory efficient)
- CuDSS (only for GPU accelerated computation)

	PARDISO	MUMPS	SPOOLES	CuDSS
Multicore	Fastest	Yes	Slowest	✓
Cluster	✗	yes	✗	✓
GPU	✗	✗	✗	✓

Read more: https://www.comsol.com/blogs/solutions-linear-systems-equations-direct-iterative-solvers

Direct Solvers – Conclusions

- There are the specific requirements for the solved matrix
- User usually cannot display the matrix and its properties
- Leave the decision to algorithms! It works 99%

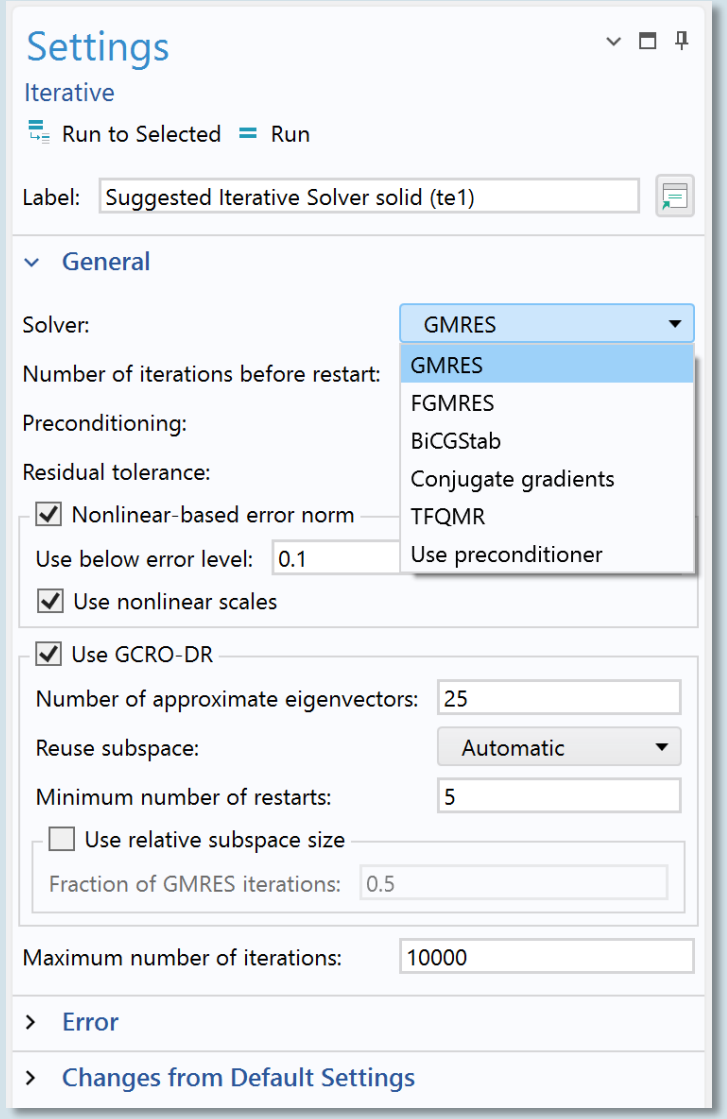
```
Failed to find a solution.
The relative residual (0.06) is greater than the relative tolerance.
Returned solution is not converged.
```

There is one specific error message that clearly indicates there is no solution.
You should check if the problem is correctly constrained:
https://www.comsol.com/blogs/solutions-linear-systems-equations-direct-iterative-solvers

Iterative Solvers

Iterative Solvers: Concept

1. Start with initial guess x_o (usually $x_o = 0$)
2. Compute residual: $r = b - A \cdot x$
3. Update x “somehow” using residual information
4. Repeat until $\|r\| <$ tolerance



Iterative Solvers: Jacobi iteration

- $x^{(n+1)}$ update
  - Jacobi: *x* from previous iteration (n)
  - Faster methods on similar principle: Gauss-Seidel, Conjugate Gradients, Krylov (GMRES)
- Residual is decreasing
- Residual ‖r‖ shrinks → it converges

Matrix form A x = b, residual r = b − A x

$$A = \begin{bmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 1 \end{bmatrix} \quad x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \quad b = \begin{bmatrix} 0 \\ 0 \\ -10 \end{bmatrix}$$

Jacobi iteration method:

$$x_i^{(n+1)} = (b_i - \sum_{j \neq i} a_{ij} x_j^{(n)}) / a_{ii}$$

Starting from $x^0 = [0, 0, 0]$:

Iter 1: x = [0.000, 0.000, -10.000] ‖r‖ = 10.00

Iterative Solvers: Jacobi iteration

- $x^{(n+1)}$ update
  - Jacobi: *x* from previous iteration (n)
  - Faster methods on similar principle: Gauss-Seidel, Conjugate Gradients, Krylov (GMRES)
- Residual is decreasing
- Residual ‖r‖ shrinks → it converges

Matrix form A x = b, residual r = b − A x

$$A = \begin{bmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 1 \end{bmatrix} \quad x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \quad b = \begin{bmatrix} 0 \\ 0 \\ -10 \end{bmatrix}$$

Jacobi iteration method:

$$x_i^{(n+1)} = (b_i - \sum_{j \neq i} a_{ij} x_j^{(n)}) / a_{ii}$$

Starting from $x^0 = [0, 0, 0]$:

```
  Iter  1:  x = [0.000,   0.000, -10.000]  ‖r‖ = 10.00
  Iter  2:  x = [0.000,  -5.000, -10.000]  ‖r‖ =  7.07
```

Iterative Solvers: Jacobi iteration

- $x^{(n+1)}$ update
  - Jacobi: *x* from previous iteration (n)
  - Faster methods on similar principle: Gauss-Seidel, Conjugate Gradients, Krylov (GMRES)
- Residual is decreasing
- Residual ‖r‖ shrinks → it converges

Matrix form A x = b, residual r = b − A x

$$A = \begin{bmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 1 \end{bmatrix} \quad x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \quad b = \begin{bmatrix} 0 \\ 0 \\ -10 \end{bmatrix}$$

Jacobi iteration method:

$$x_i^{(n+1)} = (b_i - \sum_{j \neq i} a_{ij} x_j^{(n)}) / a_{ii}$$

Starting from $x^0 = [0, 0, 0]$:

```
  Iter  1:  x = [0.000,   0.000, -10.000]  ‖r‖ = 10.00
  Iter  2:  x = [0.000,  -5.000, -10.000]  ‖r‖ =  7.07
  Iter  5:  x = [-4.375,  -8.750, -18.750] ‖r‖ =  5.63
```

Iterative Solvers: Jacobi iteration

- $x^{(n+1)}$ update
  - Jacobi: *x* from previous iteration (n)
  - Faster methods on similar principle: Gauss-Seidel, Conjugate Gradients, Krylov (GMRES)
- Residual is decreasing
- Residual ‖r‖ shrinks → it converges

Matrix form A x = b, residual r = b − A x

$$A = \begin{bmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 1 \end{bmatrix} \quad x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \quad b = \begin{bmatrix} 0 \\ 0 \\ -10 \end{bmatrix}$$

Jacobi iteration method:

$$x_i^{(n+1)} = (b_i - \sum_{j \neq i} a_{ij} x_j^{(n)}) / a_{ii}$$

Starting from $x^0 = [0, 0, 0]$:

```
  Iter  1:  x = [0.000,   0.000, -10.000]  ‖r‖ = 10.00
  Iter  2:  x = [0.000,  -5.000, -10.000]  ‖r‖ =  7.07
  Iter  5:  x = [-4.375,  -8.750, -18.750] ‖r‖ =  5.63
  Iter 10:  x = [-6.836, -15.254, -23.672] ‖r‖ =  2.24
```

Iterative Solvers: Jacobi iteration

- $x^{(n+1)}$ update
  - Jacobi: *x* from previous iteration (n)
  - Faster methods on similar principle: Gauss-Seidel, Conjugate Gradients, Krylov (GMRES)
- Residual is decreasing
- Residual ‖r‖ shrinks → it converges

Matrix form A x = b, residual r = b − A x

$$A = \begin{bmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 1 \end{bmatrix} \quad x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \quad b = \begin{bmatrix} 0 \\ 0 \\ -10 \end{bmatrix}$$

Jacobi iteration method:

$$x_i^{(n+1)} = (b_i - \sum_{j \neq i} a_{ij} x_j^{(n)}) / a_{ii}$$

Starting from $x^0 = [0, 0, 0]$:

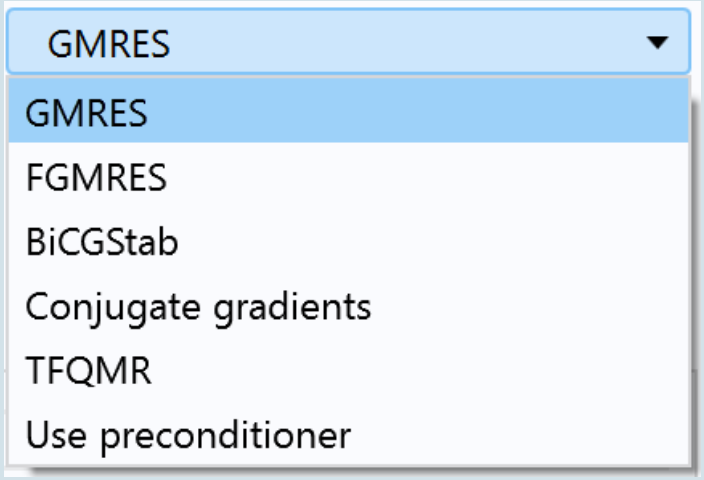
```
  Iter  1:  x = [0.000,   0.000, -10.000]  ‖r‖ = 10.00
  Iter  2:  x = [0.000,  -5.000, -10.000]  ‖r‖ =  7.07
  Iter  5:  x = [-4.375,  -8.750, -18.750] ‖r‖ =  5.63
  Iter 10:  x = [-6.836, -15.254, -23.672] ‖r‖ =  2.24
  Iter 20:  x = [-9.249, -18.874, -28.498] ‖r‖ =  0.53
```

Iterative Solver: Requirements and Properties

- More requirements in comparison with the Direct approach.
- Convergence conditions (not always guaranteed!):
  - **Diagonal dominance** — $|a_{ii}| > \sum_{j \neq i} |a_i|$ for each row
  - **Symmetric positive definite** (SPD)
  - **Spectral radius ρ(M) < 1** (maximum of the absolute values of its eigenvalues: decreasing error) valid for stationary methods (Jacobi, Gauss-Seidel)
  - In reality, you need a good **preconditioner** to compute effectively.

Iterative Solvers in COMSOL Multiphysics

- GMRES – *Generalized Minimum Residual*. Uses the Krylov method, it is default and universal iterative solver, works even with non-symmetry matrices
- FGMRES – *Flexible GMRES.*
- BiCGStab – *Biconjugate Gradient Stabilized.*
- CG – only for SPD matrices (heat conduction, elasticity)
- Jacobi method, Gauss-Seidel – The simplest of COMSOL‘s preconditioners



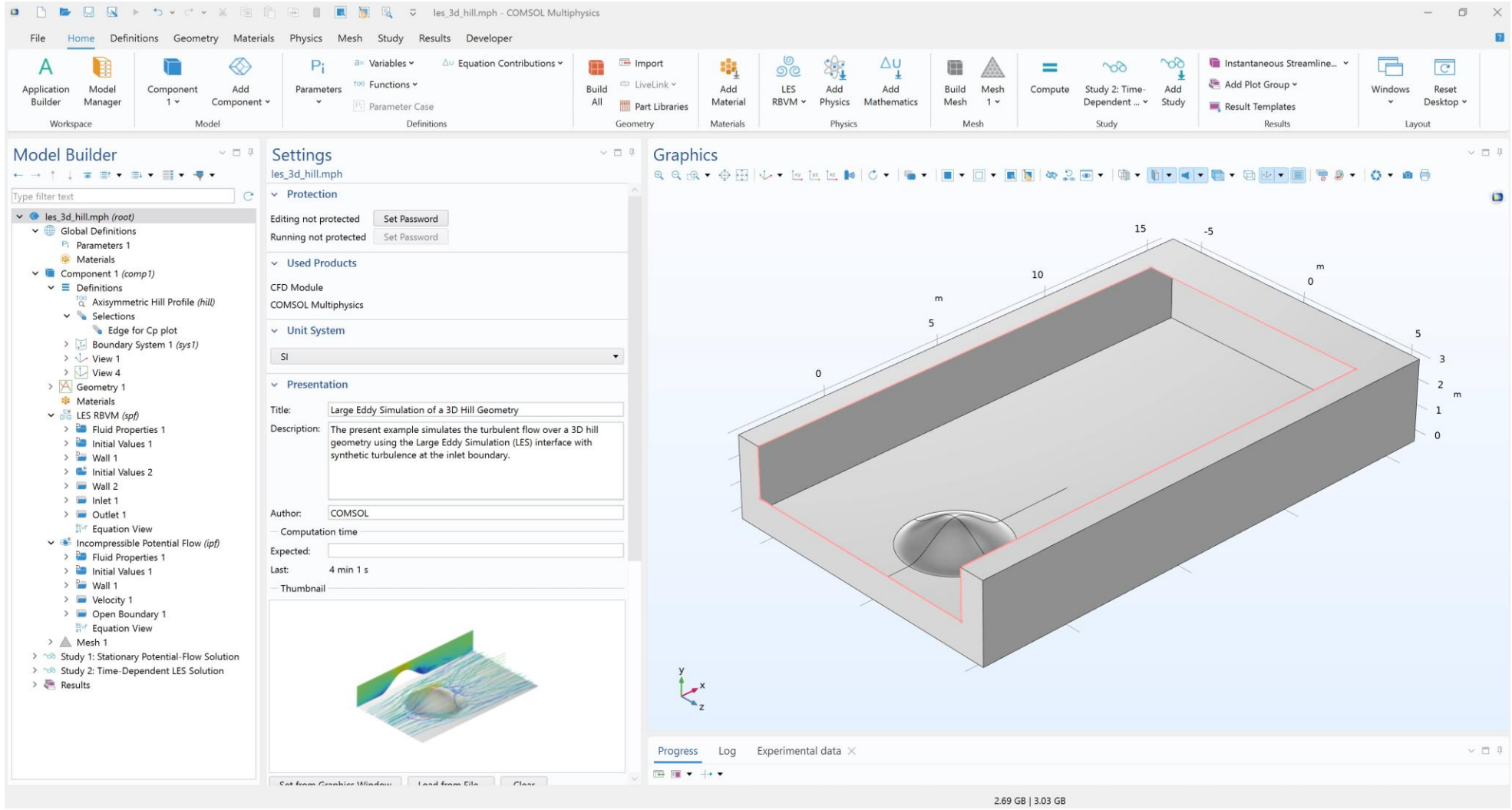
Direct vs. Iterative — Comparison

Property	Direct Solver	Iterative Solver
Method	LU Factorization	Krylov (GMRES, CG)
Solution	Exact	Approximate
Memory	$O(n^2)$ — high	O(n) — low
Speed (small)	Fast	Comparable
Speed (large 3D)	Slow / impossible	Often faster
Tuning	None needed	Preconditioner required
Robustness	Very robust	May not converge

Conclusions

- Default Solver Settings are the best option (almost every time)
- You can choose between:
  - One-way or Two-way coupling of multiple physics
- For two-way coupling, you can choose:
  - Fully Coupled or Segregated approach
- Final resulting system of algebraic equations can be solved:
  - With direct or iterative solver

How to Store Only Part of Computed Data



Thank you for your attention!