Marko Petek

Univerza v Mariboru
Fakulteta za naravoslovje in matematiko

Maribor,
24th August 2019

Fluid.Internals.Lsfem

Block.cs

The whole purpose of a **Block** is to provide a means of creating nodes by specifying parameters tW and tH on the block's edges.

Vec2 CreatePos(tW, tH)	`position vector`
double tW	`parameter along lower edge`
double tH	`parameter along left edge`

PseudoElement.cs

A grid and a sequence of **PseudoElements**, a 2D array and a 1D array respectively, called Patches and Joints, are created for the purpose of transfering nodes to **Mesh** and later creating **Elements** on it. When the nodes are transfered to **Mesh**, each node inside the **PseudoElement** is mapped via index to a node on the **Mesh**. Later, when **Elements** are created on the **Mesh**, Patches and Joints are used to ease the process of mapping nodes on **Elements** to nodes on **Mesh**.

int[]	LInx	`local indices`
int[]	GInx	`global indices`
Vec2	Pos	`positions`

Simulation.cs

In **Simulation** the manual work of stitching together different blocks is delegated to the user. First, the user must create Patches and Joints which are disjoint sets of positions. In the Patches property the first key specifies a Patch which is a grid of pseudo-elements. Therefore, the next two indices specify the row and column inside this grid where the pseudo-element lies.

Dict<str,PseudoEmt[][]>	Patches	`non-shared nodes`
Dict<str,PseudoEmt[]>	Joints	`shared nodes`

Examples:

Patches["North"][3][2].Pos.X   Joints["NorthToEast"][22][2].GInx

The user must first simultaneously create each **PseudoElement** and put node positions in it, using the **Block**'s node positioning method, and at the same time assign local indices. Then, he must create nodes on **Mesh**, assigning global indices on **PseudoElements**. Finally, he creates **Elements** by mapping local indices to global indices.

Mesh.cs

Two 2nd rank tensors serve as storage for node variable values (node vars):

$\mathbf{u}_\triangleright$	$u_\triangleright^{jl}$	Tensor Uf	`free node vars` ,
$\mathbf{u}_\triangleleft$	$u_\triangleleft^{jl}$	Tensor Uc	`constrained node vars` .

where the slots represent:

$$\begin{aligned} j &\rightarrow N \texttt{ nodes,} \\ l &\rightarrow m \texttt{ variables.} \end{aligned}$$

The two tensors hold mutually exclusive information - if the component $u^{5,4}$ appears in $\mathbf{u}_\triangleright$, it cannot appear in $\mathbf{u}_\triangleleft$ because a variable is either constrained or it isn't. The sum of them thus produces a tensor which holds all values:

$\mathbf{u}_{\bowtie} = \mathbf{u}_\triangleright + \mathbf{u}_\triangleleft$ double U   `all = free + constrained` .

Here U is a method that can access values from both Uf and Uc - given an index, it retrieves the value from the correct source. A third 2nd rank tensor stores all forcing vars (right-hand side od `PDE`):

$\mathbf{f}_{\bowtie}$ $f_{\bowtie}^{jl}$ Tensor F   `forcing vars` .

The dynamic parameters (determining the system's evolution in the next step) are stored in a 4th rank tensor **A**, also known as the stiffness matrix:

$\mathbf{A}$ A^{iphl} Tensor A   `stiffness matrix` ,

where the slots represent:

$$\begin{aligned} i &\rightarrow N \texttt{ nodes,} \\ p &\rightarrow 3 \texttt{ partials,} \\ h &\rightarrow m \texttt{ equations,} \\ l &\rightarrow m \texttt{ variables .} \end{aligned}$$

Dynamics must not depend on element shapes. This is properly accounted for with node-to-node influence weights in the form of overlap integrals. Triple overlap integrals reside in a 7th rank tensor **S**, while double overlap integrals reside in a 5th rank tensor **T**:

S	$S_{\varepsilon\beta\alpha p\gamma\delta q}$	Tensor S	`tripple overlap integrals ,`
T	$T_{\varepsilon\beta\alpha p\gamma}$	Tensor T	`double overlap integrals ,`

where the slots represent:

$\varepsilon \rightarrow n$ `elements,`

$\beta \rightarrow$ `12 basis funcs of 1st` **A** ,

$\alpha \rightarrow$ `12 basis funcs of` **v** ,

$p \rightarrow$ `3 partials of` **v** ,

$\gamma \rightarrow$ `12 basis funcs of` $\mathbf{u}_{\triangleleft}$ `or` **f** ,

$\delta \rightarrow$ `12 basis funcs of` $\mathbf{u}_{\triangleright}$,

$q \rightarrow$ `3 partials of` $\mathbf{u}_{\triangleright}$.

In the assembly process we go over each element ε:

$$K_{ikjl} = \sum_{\varepsilon}^{n} \sum_{\substack{\alpha,\delta\,\ni: \\ (\varepsilon,\alpha)=i \\ (\varepsilon,\delta)=j}}^{12} \left(S_{\varepsilon\beta\alpha p\gamma\delta q} A^{\overset{\varepsilon}{\beta} p}{}_{hk} A^{\overset{\varepsilon}{\gamma} qhl} u_{\triangleright}^{\overset{\varepsilon}{\delta} l} \right) \tag{1}$$

$$F_{ik} = \sum_{\varepsilon}^{n} \sum_{\substack{\alpha\,\ni: \\ (\varepsilon,\alpha)=i}}^{12} \left(T_{\varepsilon\beta\alpha p\gamma} A^{\overset{\varepsilon}{\beta} p}{}_{hk} f_{\bowtie}^{\overset{\varepsilon}{\gamma} h} - S_{\varepsilon\beta\alpha p\gamma\delta q} A^{\overset{\varepsilon}{\beta} p}{}_{hk} A^{\overset{\varepsilon}{\gamma} qhl} u_{\triangleleft}^{\overset{\varepsilon}{\delta} l} \right) \tag{2}$$

$$K_{ikjl} u_{\triangleright}^{jl} = F_{ik} \ .$$

The tensor K_{ikjl} is symmetric over pairs of indices (i,k) and (j,l) because the integral between nodes (i,j), for corresponding partials (k,l), must be identical to the integral between nodes (j,i), for corresponding partials (l,k). Therefore, to avoid redundant work, we iterate on each element over node indices α and δ in the following way:

α	δ		
1			0
2		1	0

3									2	1	0
⋮						⋮					
10		9	8	7	6	5	4	3	2	1	0
11	10	9	8	7	6	5	4	3	2	1	0

while going through all 3×3 combinations of derivatives. We add the same value to the pair (i,j) and its symmetric pair (j,i). Then iterate over all repeated indices: (0,0), (1,1), ..., (10,10), (11,11) on the diagonal and add each value for all 3×3 combinations only once.

$$
\begin{aligned}
\Delta x_{\mathrm{D}} &= x_3 - x_0 \\
\Delta x_{\mathrm{U}} &= x_6 - x_9 \\
\Delta x_{\mathrm{L}} &= x_9 - x_0 \\
\Delta x_{\mathrm{R}} &= x_6 - x_3 \\
\Delta y_{\mathrm{D}} &= y_3 - y_0 \\
\Delta y_{\mathrm{U}} &= y_6 - y_9 \\
\Delta y_{\mathrm{L}} &= y_9 - y_0 \\
\Delta y_{\mathrm{R}} &= y_6 - y_3
\end{aligned}
$$

$$
\begin{aligned}
J_{11} &= \frac{1}{4}(\Delta x_{\mathrm{D}}(1-\eta) + \Delta x_{\mathrm{U}}(1+\eta)) \\
J_{12} &= \frac{1}{4}(\Delta x_{\mathrm{L}}(1-\xi) + \Delta x_{\mathrm{R}}(1+\xi)) \\
J_{21} &= \frac{1}{4}(\Delta y_{\mathrm{D}}(1-\eta) + \Delta y_{\mathrm{U}}(1+\eta)) \\
J_{22} &= \frac{1}{4}(\Delta y_{\mathrm{L}}(1-\xi) + \Delta y_{\mathrm{R}}(1+\xi))
\end{aligned}
$$

in:

$$
\begin{aligned}
S_1 &= \frac{1}{4}(1-\xi)(1-\eta) \\
S_2 &= \frac{1}{4}(1+\xi)(1-\eta) \\
S_3 &= \frac{1}{4}(1+\xi)(1+\eta) \\
S_4 &= \frac{1}{4}(1-\xi)(1+\eta)
\end{aligned}
$$