Whitepaper:
Diffusion Solver in ChiTech

August, 2018

Jan Vermaak

Rev 1.00

Contents

1 Introduction

The "generalized diffusion solver" in Chi-Tech is based on a well known partial differential equation known as the **diffusion equation**. This equation can be used to model the behavior of many physical systems and has been studied in great depth for many years by engineers, scientists and mathematicians. The equation for a single unknown, ϕ, has the general time dependent linear form

$$\frac{d\phi}{dt} = \boldsymbol{\nabla} \cdot (D\boldsymbol{\nabla}\phi) - \sigma\phi + q \tag{1.1}$$

and can be outfitted with a number of different boundary conditions including Dirichlet-type, Neumann-type, and Robin-type boundary conditions. Different spatial discretizations can be applied to the equation including Finite Difference (FD), Finite Volume (FV), Continuous Galerkin Finite Element Method (CFEM), and Discontinuous Galerkin Finite Element (DFEM). In Chi-Tech the available spatial discretizations are FV, CFEM and DFEM. Additionally, different time discretizations can be applied for which Chi-Tech for now supports Forward Euler, Backward Euler, and Crank-Nicholson.

The steady state form of the diffusion equation will be used in this whitepaper to detail the different spatial discretization techniques and is denoted by

$$-\boldsymbol{\nabla} \cdot (D\boldsymbol{\nabla}\phi) + \sigma\phi = q. \tag{1.2}$$

This equation can be related to techniques contained in literature under certain conditions. When the removal term ($\sigma\phi$) is removed and D=1 then the equation becomes Poisson's equation

$$\boldsymbol{\nabla} \cdot (\boldsymbol{\nabla}\phi) = -q, \tag{1.3}$$

and if the spatial source term, q, is also removed then the equation becomes Laplace's equation

$$\boldsymbol{\nabla} \cdot (\boldsymbol{\nabla}\phi) = 0. \tag{1.4}$$

In both these forms the divergence of the gradient ($\boldsymbol{\nabla} \cdot \boldsymbol{\nabla}\phi$) is replaced by the Laplacian operator, $\boldsymbol{\nabla}^2$, which can generally denoted by the symbol Δ after which the following are equivalent

$$\begin{aligned} \boldsymbol{\nabla} \cdot \boldsymbol{\nabla}\phi &= 0 \\ \boldsymbol{\nabla}^2\phi &= 0 \\ \Delta\phi &= 0 \end{aligned} \tag{1.5}$$

2 Finite Volume

2.1 Discretization

The FV method associates only a single value of the unknown per cell, located at the cell centroid, and is discretized by first integrating equation (1.2) over the cell volume

$$-\int_V \boldsymbol{\nabla} \cdot (D\boldsymbol{\nabla}\phi)\ dV + \int_V \sigma\phi\ dV = \int_V q\ dV \tag{2.1}$$

after which we apply Gauss' divergence theorem to the first term

$$-\int_V \boldsymbol{\nabla} \cdot (D\boldsymbol{\nabla}\phi)\ dV = -\int_S \mathbf{n} \cdot (D\boldsymbol{\nabla}\phi)\ dA \tag{2.2}$$

and then apply discretized integration over cell volume, V, and cell face areas, A_f,

$$-\sum_f \mathbf{A}_f \cdot (D\boldsymbol{\nabla}\phi)_f + V(\sigma\phi) = V\ q, \tag{2.3}$$

where $\mathbf{A}_f = A_f\mathbf{n}_f$ is the face area vector. This formulation is not complete until we have a discretized expression for $(D\boldsymbol{\nabla}\phi)_f$. The problem in depicting the details of this term in a Finite Volume discretization scheme is that it has a tremendously simplistic representation on orthogonal grids, and a significantly more complicated representation in unstructured grids. Generally the prior (on orthogonal grids) exhibits 2nd order convergence, however, the latter requires more care because the possibility of skewed faces can cause the method to become less than 1st order accurate.

The next two sections are devoted to separately describe the discretization on orthogonal and unstructured grids, respectively. Ultimately the unstructured implementation will be used since it is unifying, however, the general process can be observed on the orthogonal grid detail.

2.2 Discretization on Orthogonal grids

2.2.1 Terminology

Figure 1 below contains a graphic depiction of orthogonal 2D cells which we will use the define numerous concepts.

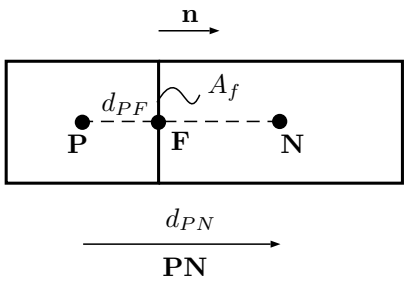

Figure 1: Reference vector layout for neighboring cells on an orthogonal grid.

The points **P**, **N** and **F** are the present-cell-, neighboring-cell- and adjoining-face-centroids, respectively. The vector **PF** is from **P** to **F** and its length is used as d_{PF}. Similarly the vector **PN** is used to define d_{PN}.

2.2.2 Weighted harmonic mean of a quantity D at a face

Some quantities, like the diffusion coefficient, require the harmonic mean at the faces. For orthogonal grids this requires a weighted harmonic mean where the weight requires the definition of

$$r_O = \frac{d_{PF}}{d_{PN}} \tag{2.4}$$

after which the harmonic mean is defined as

$$D_f = \left(\frac{r_O}{D_P} + \frac{1 - r_O}{D_P} \right)^{-1} \tag{2.5}$$

2.2.3 Flux of a gradient

With the definition of face diffusion coefficient the required term in equation (2.3) becomes

$$\mathbf{A}_f \cdot (D \boldsymbol{\nabla} \phi)_f = D_f \frac{A_f}{d_{PN}} \left(\phi_N - \phi_P \right) \tag{2.6}$$

2.2.4 Assembling the linear system

We now proceed by inserting eq. (2.6) into eq. (2.3), and mark values at iteration (n) to be implicit and values at $(n-1)$ as explicit, to get

$$-\sum_f D_f \frac{A_f}{d_{PN}} \Big(\phi_N - \phi_P \Big)^{(n)} + V\sigma\phi_P^{(n)} = V\ q_P^{(n-1)} \tag{2.7}$$

This can then be simplified into a coefficient formulation

$$a_P\phi_P + \sum_f a_{Nf}\phi_N = b_N$$

where

$$\begin{aligned} a_P &= \sum_f D_f \frac{A_f}{d_{PN}} + V\sigma \\ a_{Nf} &= -D_f \frac{A_f}{d_{PN}} \\ b_N &= V\ q_P^{(n-1)} \end{aligned}$$

The final process is to assemble this coefficient formulation for each cell P where each index N and P maps to a given i, j matrix or i vector index. The methodology to obtain these i, j indices requires some machinery to be developed relating to sparsity patterns which will be discussed next.

2.3 Discretization on Unstructured grids

2.3.1 Terminology

Figure 2 below contains a graphic depiction of unstructured 2D cells which we will use to define numerous concepts.

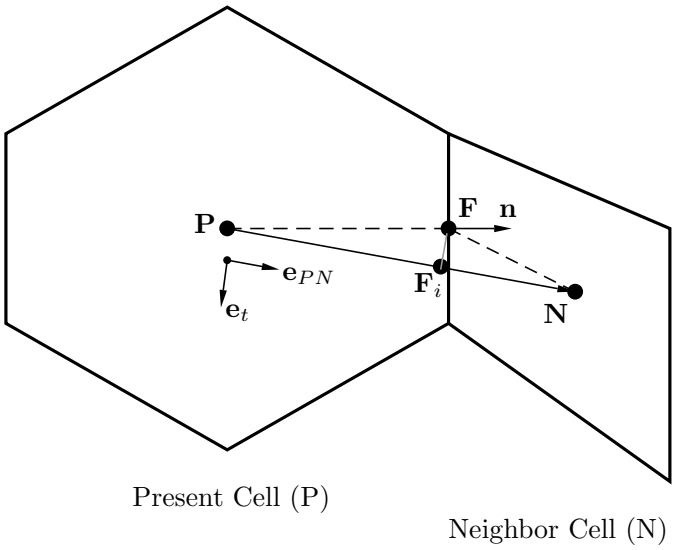

Figure 2: Reference vector layout for neighboring cells on an unstructured grid.

The points **P**, **N** and **F** are the present-cell-, neighboring-cell- and adjoining-face-centroids, respectively. The vector **PN** is from **P** to **N**, and the projection of **PF** onto **PN** is at point $\mathbf{F}_i$ (not actually at the intersection of **PN** with the face). The vector $\mathbf{e}_{PN}$ is the unit vector along **PN** computed from $\mathbf{PN}/d_{PN}$, where $d_{PN} = ||\mathbf{PN}||_2$. The vector $\mathbf{e}_t$ is the unit tangential vector formed from $\mathbf{e}_t = \mathbf{e}_{PN} - \mathbf{n}$, with $\mathbf{n}$ being the face normal. The distance from **P** to $\mathbf{F}_i$, d_{PF_i}, is computed as $d_{PF_i} = \mathbf{PF} \cdot \mathbf{e}_{PN}$, and $\mathbf{F}_i = \mathbf{P} + d_{PF_i}\, \mathbf{e}_{PN}$.

2.3.2 Weighted harmonic mean of a quantity D at a face

Some quantities, like the diffusion coefficient, require the harmonic mean at the faces. For unstructured grids this requires a weighted harmonic mean where the weight requires the definition of

$$r_P = \frac{d_{PF_i}}{d_{PN}} \tag{2.8}$$

after which the harmonic mean is defined as

$$D_f = \left(\frac{r_P}{D_P} + \frac{1 - r_P}{D_P} \right)^{-1} \tag{2.9}$$

2.3.3 Flux of a gradient

We can now develop the computation of $\mathbf{A}_f \cdot (D_f \boldsymbol{\nabla}\phi)_f$. When we deal with skewed faces, this diffusion term needs some treatment. In general, since the face is skewed, the flux at the cell face develops additional components other than the orthogonal component along the direction of **PN**. These additional components are not intuitive to understand. Refer to Figure 3 below for the following discussion.

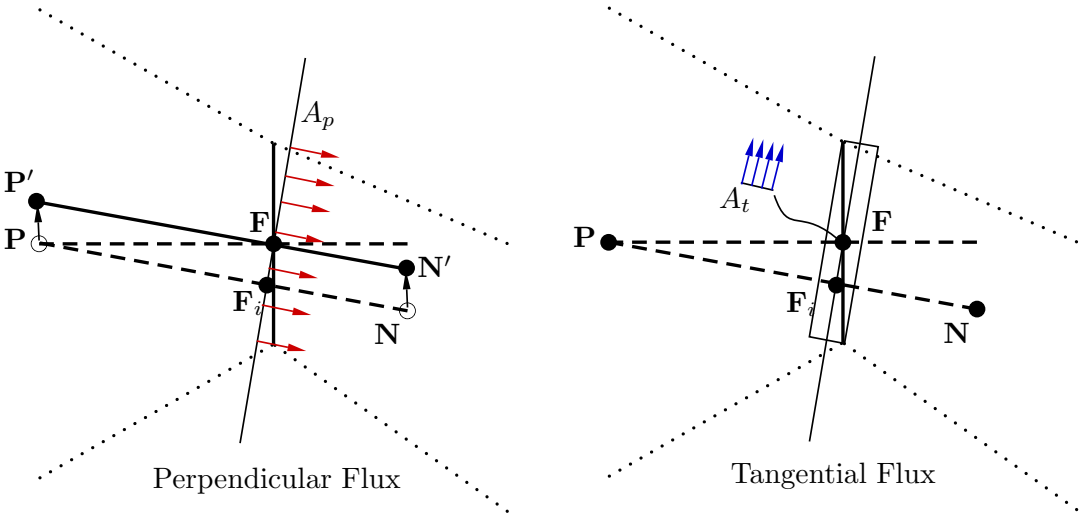

Figure 3: Decomposed areas for the interpolation of a gradient flux

Given that the correct way to approximate the flux across the face is to compute the gradient along its centroid, we require the gradient to be computed from $\phi_{P'}$ and $\phi_{N'}$ along the projected perpendicular area A_p. Additionally we need to compute the flux along the tangential area A_t. Including these terms within an implicit formulation is, however, cumbersome to say the least. Instead we include explicit corrections as described in [2].

Firstly, we formulate the fluxes at $\mathbf{P}'$ and $\mathbf{N}'$ (see the left schematic in Figure 3) as

$$\begin{aligned} \phi_{P'} &= \phi_P + (\boldsymbol{\nabla}\phi)_P^{(n-1)} \cdot \mathbf{F_i F} \\ \phi_{N'} &= \phi_N + (\boldsymbol{\nabla}\phi)_N^{(n-1)} \cdot \mathbf{F_i F} \end{aligned} \tag{2.10}$$

where $\mathbf{F_i F}$ is the vector from $\mathbf{F}_i$ to $\mathbf{F}$ and the gradients are computed from the previous iteration's flux field. The perpendicular gradient term can then be computed from

$$\begin{aligned} \Big(\mathbf{A}_f \cdot (\boldsymbol{\nabla}\phi)_f\Big)_p &= \frac{A_p}{d_{PN}}\Big(\phi_{N'} - \phi_{P'}\Big) \\ &= \frac{A_p}{d_{PN}}\Big(\phi_N - \phi_P\Big) + \frac{A_p}{d_{PN}}\Big((\boldsymbol{\nabla}\phi)_N - (\boldsymbol{\nabla}\phi)_P\Big)^{(n-1)} \cdot \mathbf{F_i F} \end{aligned}$$

where $A_p = A_f/(\mathbf{e}_{PN} \cdot \mathbf{n})$. Secondly, the tangential component can be computed from

$$\begin{aligned}\Big(\mathbf{A}_f \cdot (\boldsymbol{\nabla}\phi)_f\Big)_t &= (\boldsymbol{\nabla}\phi)_F \cdot \mathbf{A}_t \\ &\approx (\boldsymbol{\nabla}\phi)_{F_i}^{(n-1)} \cdot \mathbf{A}_t\end{aligned} \tag{2.11}$$

where $\mathbf{A}_t = \mathbf{A}_f - A_p \mathbf{e}_{PN}$ and the gradient at the face centroid is approximated by the gradient interpolated at $\mathbf{F}_i$ which is not an unreasonable approximation considering that cell skewness is normally minimized. Putting it all together we have

$$\mathbf{A}_f \cdot (\boldsymbol{\nabla}\phi)_f = \frac{A_p}{d_{PN}}\Big(\phi_N - \phi_P\Big) + \frac{A_p}{d_{PN}}\Big((\boldsymbol{\nabla}\phi)_N - (\boldsymbol{\nabla}\phi)_P\Big)^{(n-1)} \cdot \mathbf{F_i F} + (\boldsymbol{\nabla}\phi)_{F_i}^{(n-1)} \cdot \mathbf{A}_t \tag{2.12}$$

In these equations we require the explicit value of $(\boldsymbol{\nabla}\phi)^{n-1}$ which can be computed using different methods, i.e., Green-Gauss and Weighted least-squares. These two methods will now be detailed.

2.3.4 Computing gradients using Green-Gauss

This formulation is identical to "Option 3" of chapter 9.2 in the book by Moukalled et al. [1]. The first order Green-Gauss approximation reduces essentially to a summation over faces.

$$\boldsymbol{\nabla}\phi = \frac{1}{V}\sum_f \mathbf{A}_f \phi_f \tag{2.13}$$

where ϕ_f is taken as a linear interpolation between the adjoining face's cells. This formulation poses some challenges when undetermined faces fluxes are encountered, as is the case with boundaries and unstructured meshes, and therefore we require an algorithm to compensate. Therefore the following algorithm is applied:

1. Compute $\boldsymbol{\nabla}\phi^{(n)}$ over the entire domain using $\frac{1}{V}\sum_f \mathbf{A}_f\phi_f$ where ϕ_f is computed from (**??**) as

   $$\phi_f = (1 - r_P)\phi_P + (r_P)\phi_N + \Big[(1 - r_P)\boldsymbol{\nabla}\phi_P + (r_P)\boldsymbol{\nabla}\phi_N\Big]^{(n-1)} \cdot \mathbf{F_i F}$$

   and boundary faces are computed as

   $$\phi_b = \phi_P + \boldsymbol{\nabla}\phi_P^{(n-1)} \cdot \mathbf{PF}$$

2. Compute $||\boldsymbol{\nabla}\phi^{(n)} - \boldsymbol{\nabla}\phi^{(n-1)}||_2$. If $||\boldsymbol{\nabla}\phi^{(n)} - \boldsymbol{\nabla}\phi^{(n-1)}||_2 < \epsilon$ then terminate.

3. Swap $\boldsymbol{\nabla}\phi^{(n)}$ and $\boldsymbol{\nabla}\phi^{(n-1)}$.

4. Repeat 2 to 4.

In most cases the iteration process adds little to the iterative performance and it can be sufficient to just terminate after a single iteration.

2.3.5 Computing gradients using weighted Least-Squares

Section 9.3 in [1] provides a very good description of how the weighted Least-Squares gradients are determined. In that formulation

$$\left\{\sum_f w_f \mathbf{PN} \otimes \mathbf{PN}\right\} \boldsymbol{\nabla}\phi_P = \sum_f \left[w_f \mathbf{PN}(\phi_N - \phi_P)\right]$$

where

$$w_f = \frac{1}{(||\mathbf{PN}||_2)^2}$$

and the term in {} is a tensor. On boundaries, **PN** and ϕ_N is replaced by **PF** and ϕ_b.

Note that this results in a system that needs to solved for each cell.

2.3.6 Assembling the linear system

With all the individual pieces of functionality established we can now determine how to assemble the linear system. We start with eq. (2.3) where now we insert eq. (2.12), and mark values at iteration (n) to be implicit and values at $(n-1)$ as explicit, after which we get

$$-\sum_f D_f \left[\frac{A_p}{d_{PN}}\left(\phi_N - \phi_P\right)^{(n)} + \frac{A_p}{d_{PN}}\left((\boldsymbol{\nabla}\phi)_N - (\boldsymbol{\nabla}\phi)_P\right)^{(n-1)} \cdot \mathbf{F_i F} + (\boldsymbol{\nabla}\phi)_{F_i}^{(n-1)} \cdot \mathbf{A}_t\right] + V\sigma\phi_P^{(n)} = V\ q_P^{(n-1)} \tag{2.14}$$

which we can arrange to have all the implicit values to the left

$$-\sum_f D_f \left[\frac{A_p}{d_{PN}}\left(\phi_N - \phi_P\right)^{(n)}\right] + V\sigma\phi_P^{(n)} = V\ q_P^{(n-1)} + \sum_f D_f \left[\frac{A_p}{d_{PN}}\left((\boldsymbol{\nabla}\phi)_N - (\boldsymbol{\nabla}\phi)_P\right)^{(n-1)} \cdot \mathbf{F_i F} + (\boldsymbol{\nabla}\phi)_{F_i}^{(n-1)} \cdot \mathbf{A}_t\right]. \tag{2.15}$$

This can be simplified into a coefficient formulation

$$a_P\phi_P + \sum_f a_{Nf}\phi_N = b_N \tag{2.16}$$

where

$$\begin{aligned} a_P &= \sum_f D_f \frac{A_p}{d_{PN}} + V\sigma \\ a_{Nf} &= -D_f \frac{A_p}{d_{PN}} \\ b_N &= V\ q_P^{(n-1)} + \sum_f b_{Nf} \\ b_{Nf} &= D_f \left[\frac{A_p}{d_{PN}} \left((\boldsymbol{\nabla}\phi)_N - (\boldsymbol{\nabla}\phi)_P \right)^{(n-1)} \cdot \mathbf{F_i F} + (\boldsymbol{\nabla}\phi)_{F_i}^{(n-1)} \cdot \mathbf{A}_t \right]. \end{aligned} \tag{2.17}$$

The final process is to assemble this coefficient formulation for each cell P where each index N and P maps to a given i, j matrix or i vector index. The methodology to obtain these i, j indices requires some machinery to be developed relating to sparsity patterns which will be discussed next.

2.4 Boundary conditions

2.4.1 Dirichlet boundary conditions $\phi_N = \phi_B$

A Dirichlet boundary condition manifests in the $\mathbf{A}_f \cdot (D\boldsymbol{\nabla}\phi)_f$ term where we start with

$$\mathbf{A}_f \cdot (\boldsymbol{\nabla}\phi)_f = \frac{A_p}{d_{PN}} \left(\phi_N - \phi_P \right)^{(n)} + \frac{A_p}{d_{PN}} \left((\boldsymbol{\nabla}\phi)_N - (\boldsymbol{\nabla}\phi)_P \right)^{(n-1)} \cdot \mathbf{F_i F} + (\boldsymbol{\nabla}\phi)_{F_i}^{(n-1)} \cdot \mathbf{A}_t$$

and note that the definition of $\mathbf{F_i F}$ requires the projection of $\mathbf{PF}$ onto $\mathbf{PN}$, however, in this case the vectors $\mathbf{PF}$ and $\mathbf{PN}$ are the same and therefore $\mathbf{F_i}$ equals $\mathbf{F}$ from which it follows that $\mathbf{F_i F}$ is zero in magnitude. We also know that $(\boldsymbol{\nabla}\phi)_N = 0$ and therefore $(\boldsymbol{\nabla}\phi)_{F_i}^{(n-1)}$ just becomes the projection of $(\boldsymbol{\nabla}\phi)_P$ to $\mathbf{F}$. Since $(\boldsymbol{\nabla}\phi)_P$ is constant over the whole cell, $(\boldsymbol{\nabla}\phi)_{F_i}^{(n-1)}$ is simply taken to be equal to $(\boldsymbol{\nabla}\phi)_P$ from which we now have

$$\mathbf{A}_f \cdot (D\boldsymbol{\nabla}\phi)_f = D_f \left[\frac{A_p}{d_{PN}} \left(\phi_B - \phi_P^{(n)} \right) + (\boldsymbol{\nabla}\phi)_P^{(n-1)} \cdot \mathbf{A}_t \right]$$

Note the substitution of $\phi_N = \phi_B$, $D_f = D_P$, and the change to the iteration indices. This term then alters, for face f^*, the a_{Nf} and b_{Nf} coefficients in eq. (2.17) as

$$\begin{aligned} a_{Nf^*} &= 0 \\ b_{Nf^*} &= D_{f^*} \left[\frac{A_p}{d_{PN}} \phi_B + (\boldsymbol{\nabla}\phi)_P^{(n-1)} \cdot \mathbf{A}_t \right] \end{aligned}$$

2.4.2 Neumann boundary conditions $\mathbf{A}_f \cdot (D\boldsymbol{\nabla}\phi)_f = F$

This type of a boundary condition specifies an area flux on face f^* and reduces simply to the modifications

$$\begin{aligned} a_P &= \sum_{\substack{f \\ f \neq f^*}} D_f \frac{A_p}{d_{PN}} + V\sigma \\ a_{Nf^*} &= 0 \\ b_{Nf^*} &= F \end{aligned}$$

2.4.3 Robin boundary conditions $a\phi + b\mathbf{A}_f \cdot (D\boldsymbol{\nabla}\phi)_f = F$

A Robin-type boundary condition is a combination of a Dirichlet boundary condition and a Neumann boundary condition. It can be incorporated in the form

$$\mathbf{A}_f \cdot (D\boldsymbol{\nabla}\phi)_f = \frac{F}{b} - \frac{a}{b}\phi_P$$

which requires the coefficients in (2.17) to be altered, for face f^*, as

$$\begin{aligned} a_P &= \sum_{\substack{f \\ f \neq f^*}} D_f \frac{A_p}{d_{PN}} + V\sigma + \frac{a}{b} \\ a_{Nf} &= 0 \\ b_{Nf} &= \frac{F}{b} \end{aligned}$$

2.5 Sparsity pattern

2.5.1 Base block

The parallel sparsity pattern for matrix assembly requires **node ordering** in a single block-sense (base block) as depicted in Figure 4 below. The nodes are cell centered and the base blocks are dependent on the number of cells. The local base block size (i.e., number of entries) is the number of local cells and the global base block size is the total number of cells (globally).

```
auto fv = new SpatialDiscretization_FV;
spatial_discretization = fv;

fv->AddViewOfLocalContinuum(grid);
fv->AddViewOfNeighborContinuums(grid);
fv->ReOrderNodes(grid);
```

After nodal reordering all partition locations will have two populated arrays, `locJ_block_address` and `locJ_block_size` , indexed by partition-id (i.e., processor 15: `locJ_block_address[15]`).

The `locJ_block_address` array contains the global address where each partition's base block starts and the `locJ_block_size` array contains each partitions base block size.

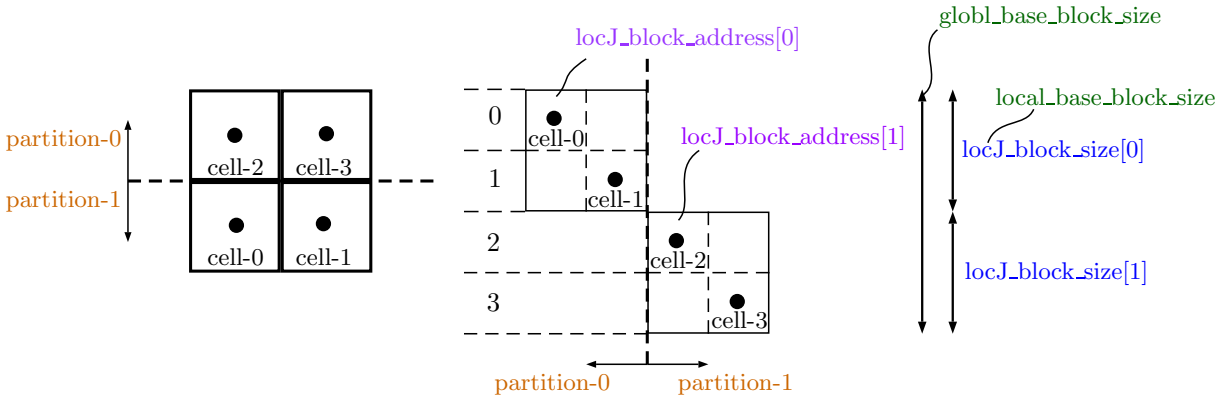

Figure 4: Base block node ordering for the Finite Volume discretization method.

2.5.2 Multiblock

For mutliple degrees-of-freedom per node Chi-Tech supports two storage modes: Nodal and Block. Nodal-storage creates a block of unknowns per node, i.e., given a node address, the DOF-addresses will follow sequentially for each unknown. In contrast, Block-storage creates a block for each unknown, i.e., each block contains all the nodes but only a single unknown. The addressing scheme is determined as shown in Figure 5 below which depicts 3 unknown (DOFs) per node. The color coding blue, green and red denotes the particular unknown, i.e., *Velocity-x,y,z*, or *Groups 0,1,2*.

Global IDs	Local IDs
cell-0	**0**
cell-1	**1**
cell-2	**0**
cell-3	**1**

N_{LC} Number of local cells (locJ_block_size)
N Number of unknowns
Unknown $block_{id}$ 0
Unknown $block_{id}$ 1
Unknown $block_{id}$ 2

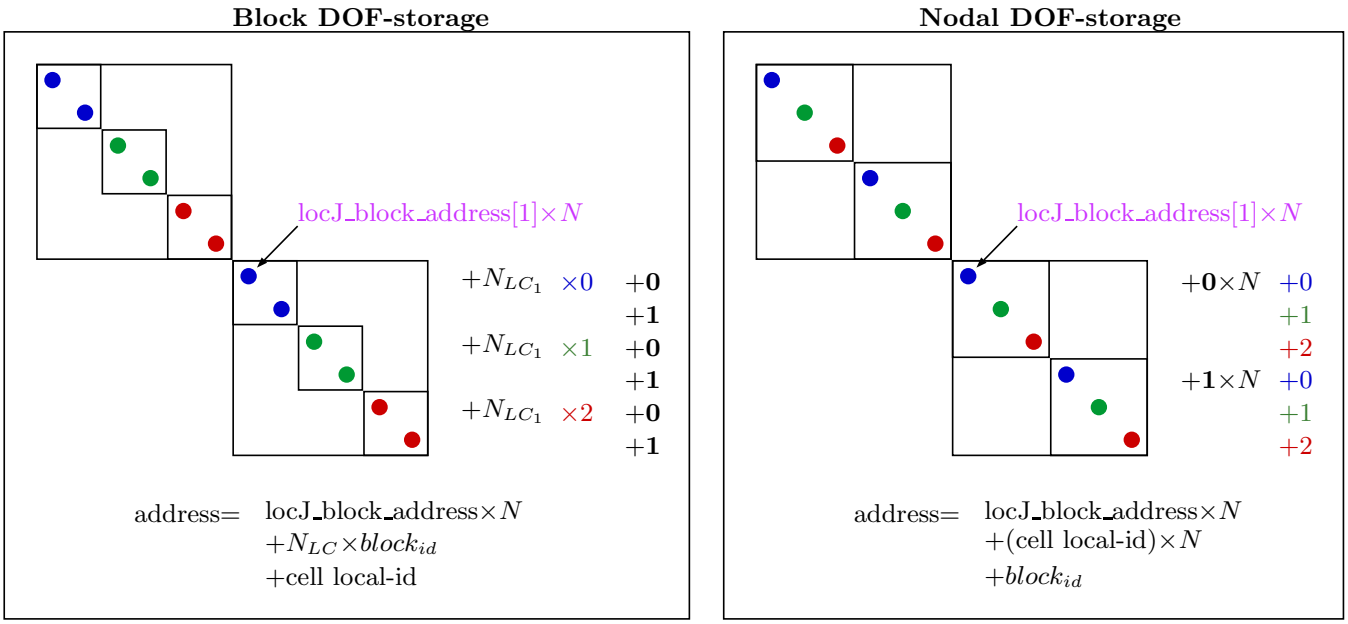

Figure 5: Addressing scheme for multiblock unknowns with the Finite Volume discretization method.

3 Continuous Finite Element Method

3.1 Discretization

3.1.1 Weak form

As a preface to the spatial discretization using CFEM and DFEM we first introduce the general weak form of the diffusion equation. We proceed with multiplying with φ_i, a function mapping eq. (1.1) to a trial space $\mathcal{D}_i$ for which we require that

$$\int_{\mathcal{D}_i}\Big(-\varphi_i\boldsymbol{\nabla}\cdot(D\boldsymbol{\nabla}\phi)\Big).dV+\int_{\mathcal{D}_i}(\varphi_i\sigma_a\phi).dV=\int_{\mathcal{D}_i}(\varphi_i q).dV. \tag{3.1}$$

In this equation we note that, from the product rule we have

$$\begin{aligned}\frac{d(fg)}{dx}&=\frac{df}{dx}\cdot g+f\cdot\frac{dg}{dx}\\ \therefore\int\frac{d(fg)}{dx}.dx&=\int\frac{df}{dx}\cdot g.dx+\int f\cdot\frac{dg}{dx}.dx\end{aligned}$$

Applying this as an analogy with $f=\varphi$ and $g=D\boldsymbol{\nabla}\phi$ we get the "integration by parts" formulation

$$\begin{aligned}\int_{\mathcal{D}_i}\boldsymbol{\nabla}\cdot(\varphi_i D\boldsymbol{\nabla}\phi).dV&=\int_{\mathcal{D}_i}\boldsymbol{\nabla}\varphi_i\cdot D\boldsymbol{\nabla}\phi.dV+\int_{\mathcal{D}_i}\varphi_i\boldsymbol{\nabla}\cdot(D\boldsymbol{\nabla}\phi).dV\\ \therefore-\int_{\mathcal{D}_i}\Big(\varphi_i\boldsymbol{\nabla}\cdot(D\boldsymbol{\nabla}\phi).dV\Big)&=\int_{\mathcal{D}_i}\boldsymbol{\nabla}\varphi_i\cdot D\boldsymbol{\nabla}\phi.dV-\int_{\mathcal{D}_i}\boldsymbol{\nabla}\cdot(\varphi_i D\boldsymbol{\nabla}\phi).dV\end{aligned} \tag{3.2}$$

Now applying Gauss's Divergence theorem on the last term we have

$$\int_{\mathcal{D}_i}\boldsymbol{\nabla}\cdot(\varphi_i D\boldsymbol{\nabla}\phi).dV=\int_{\partial\mathcal{D}}\mathbf{n}\cdot\varphi_i D\boldsymbol{\nabla}\phi.dA \tag{3.3}$$

which we can place in equation 3.2 and then subsequently into equation 3.1 which leads to the weak form

$$\boxed{\int_{\mathcal{D}_i}\Big(\boldsymbol{\nabla}\varphi_i\cdot D\boldsymbol{\nabla}\phi+\varphi_i\sigma_a\phi\Big).dV-\int_{\partial\mathcal{D}}\Big(\mathbf{n}\cdot\varphi_i D\boldsymbol{\nabla}\phi\Big).dA=\int_{\mathcal{D}_i}(\varphi_i q).dV}. \tag{3.4}$$

3.1.2 Application of basis functions

Consider ϕ approximated by the contributions of basis functions, N_j, and associated coefficients ϕ_j, i.e.

$$\phi \approx \phi_h = \sum_{j=0}^{N} \phi_j N_j. \tag{3.5}$$

Also consider the source q as being a combination of: a constant-per-element component, q_c, and a non-constant-per-element component, q_{nc}. The non-constant component can then also be expandedusing basis functions and coefficients

$$q_{nc} \approx q_{nc,h} = \sum_{j=0}^{N} q_{nc,j} N_j. \tag{3.6}$$

Equation 3.4 now becomes

$$\int_{\mathcal{D}_i} \left(\boldsymbol{\nabla}\varphi_i \cdot D\boldsymbol{\nabla}(\sum_{j=0}^{N} \phi_j N_j) + \varphi_i \sigma_a (\sum_{j=0}^{N} \phi_j N_j) \right).dV - \int_{\partial\mathcal{D}} \left(\mathbf{n} \cdot \varphi_i D\boldsymbol{\nabla}(\sum_{j=0}^{N} \phi_j N_j) \right).dA$$
$$= \int_{\mathcal{D}_i} \varphi_i q_c.dV + \int_{\mathcal{D}_i} \varphi_i (\sum_{j=0}^{N} q_{nc,j} N_j).dV$$

after which we can move the ∇ operator such that

$$\int_{\mathcal{D}_i} \left(\boldsymbol{\nabla}\varphi_i \cdot D(\sum_{j=0}^{N} \phi_j \boldsymbol{\nabla} N_j) + \varphi_i \sigma_a (\sum_{j=0}^{N} \phi_j N_j) \right).dV - \int_{\partial\mathcal{D}} \left(\mathbf{n} \cdot \varphi_i D(\sum_{j=0}^{N} \phi_j \boldsymbol{\nabla} N_j) \right).dA$$
$$= \int_{\mathcal{D}_i} \varphi_i q_c.dV + \int_{\mathcal{D}_i} \varphi_i (\sum_{j=0}^{N} q_{nc,j} N_j).dV$$

We now take into account that each integral over a trial space $\mathcal{D}_i$ is a summation over all the elements $\mathcal{K}$ that fall within this space. I.e.

Trial space i

$$\sum^{K} \Big[\int_{\mathcal{D}_i} \left(\boldsymbol{\nabla}\varphi_i \cdot D(\sum_{j=0}^{N} \phi_j \boldsymbol{\nabla} N_j) + \varphi_i \sigma_a (\sum_{j=0}^{N} \phi_j N_j) \right).dV - \int_{\partial\mathcal{D}} \left(\mathbf{n} \cdot \varphi_i D(\sum_{j=0}^{N} \phi_j \boldsymbol{\nabla} N_j) \right).dA \Big]$$
$$= \sum^{K} \Big[\int_{\mathcal{D}_i} (\varphi_i q_c).dV + \int_{\mathcal{D}_i} \varphi_i (\sum_{j=0}^{N} q_{nc,j} N_j).dV \Big]$$

Rearranging

$$\sum_{k=0}^{K}\sum_{j=0}^{N}\Big[\int_{\mathcal{D}_i}\Big(\boldsymbol{\nabla}\varphi_i \bullet D(\phi_j\boldsymbol{\nabla}N_j)+\varphi_i\sigma_a(\phi_j N_j)\Big).dV-\int_{\partial\mathcal{D}}\Big(\mathbf{n}\cdot\varphi_i D(\phi_j\boldsymbol{\nabla}N_j)\Big).dA\Big]$$
$$=\sum_{k=0}^{K}\Big[\int_{\mathcal{D}_i}(\varphi_i q_c).dV\Big]+\sum_{k=0}^{K}\sum_{j=0}^{N}\Big[\int_{\mathcal{D}_i}\varphi_i(q_{nc,j}N_j).dV\Big]$$

$$\therefore\sum_{k=0}^{K}\sum_{j=0}^{N}\phi_j\Big[D\int_{\mathcal{D}_i}\boldsymbol{\nabla}\varphi_i\bullet\boldsymbol{\nabla}N_j.dV+\sigma_a\int_{\mathcal{D}_i}\varphi_i N_j.dV-D\ \mathbf{n}\bullet\int_{\partial\mathcal{D}}\varphi_i\boldsymbol{\nabla}N_j.dA\Big]$$
$$=\sum_{k=0}^{K}\Big[q_c\int_{\mathcal{D}_i}\varphi_i.dV\Big]+\sum_{k=0}^{K}\sum_{j=0}^{N}\Big[q_{nc,j}\int_{\mathcal{D}_i}\varphi_i N_j.dV\Big]$$

By using the same shape functions for the test functions as was used for the basis functions, $\varphi_i = N_i$, we have the following, Galerkin-form of the equations

$$\begin{aligned}\therefore&\sum_{k=0}^{K}\sum_{j=0}^{N}\phi_j\Big[D\int_{\mathcal{D}_i}\boldsymbol{\nabla}N_i\bullet\boldsymbol{\nabla}N_j.dV+\sigma_a\int_{\mathcal{D}_i}N_iN_j.dV-D\ \mathbf{n}\bullet\int_{\partial\mathcal{D}}N_i\boldsymbol{\nabla}N_j.dA\Big]\\&=\sum_{k=0}^{K}\Big[q_c\int_{\mathcal{D}_i}N_i.dV\Big]+\sum_{k=0}^{K}\sum_{j=0}^{N}\Big[q_{nc,j}\int_{\mathcal{D}_i}N_iN_j.dV\Big]\end{aligned}\tag{3.7}$$

Computing the integrals of different combinations of the shape functions is specific to the type of element used, i.e. 1D slab, 2D triangle, 2D polygon, 3D tetrahedron, 3D polyhedron, etc. This information is contained in relevant whitepapers.

3.1.3 Assembling the linear system of equations

With each trial space representing an equation, we can assemble a row of a matrix A and an associated entry in the vector $\mathbf{b}$ as

For each element k, for each DOF-i, for each DOF-j

$$a_{ij}=a_{ij}+D\int_{\mathcal{D}_i}\boldsymbol{\nabla}N_i\cdot\boldsymbol{\nabla}N_j.dV+\sigma_a\int_{\mathcal{D}_i}N_iN_j.dV-D\ \mathbf{n}\cdot\int_{\partial\mathcal{D}}N_i\boldsymbol{\nabla}N_j.dA\tag{3.8}$$

$$b_i=b_i+q_{nc,j}\int_{\mathcal{D}_i}N_iN_j.dV\tag{3.9}$$

For each element k, for each DOF-i

$$b_i=b_i+q_c\int_{\mathcal{D}_i}N_i.dV\tag{3.10}$$

3.2 Boundary conditions

There are 2 primary types of boundary conditions implemented in ChiTech; **Dirichlet** type boundary conditions and **Robin** type boundary conditions.

3.2.1 Robin and Neumann type boundary conditions

A **Neumann** boundary condition takes the form

$$-D\mathbf{n}\cdot\boldsymbol{\nabla}\phi = f \qquad \text{on } \partial\mathcal{D}$$

where f represents a function. This representation is trivial to implement in the equation for a_{ij} since it essentially means the integral on the boundary is a known and can hence be moved to the right hand side. Hence the equations to do so simply become

$$a_{ij} = a_{ij} + D\int_{\mathcal{D}_i}\boldsymbol{\nabla}N_i\cdot\boldsymbol{\nabla}N_j.dV + \sigma_a\int_{\mathcal{D}_i}N_iN_j.dV - \cancel{D\,\mathbf{n}\cdot\int_{\partial\mathcal{D}}N_i\boldsymbol{\nabla}N_j.dA}$$
$$b_i = b_i + q_{nc,j}\int_{\mathcal{D}_i}N_iN_j.dV - f_j\int_{\partial\mathcal{D}}N_iN_j.dA$$

and the rest remaining untouched.

In the case of a **Robin** boundary condition the form is similar;

$$\alpha\phi + \beta D\mathbf{n}\cdot\boldsymbol{\nabla}\phi = f \qquad \text{on } \partial\mathcal{D}$$

however, this time around there is a component still dependent on ϕ which must remain on the left hand side. The equations are

$$a_{ij} = a_{ij} + D\int_{\mathcal{D}_i}\boldsymbol{\nabla}N_i\cdot\boldsymbol{\nabla}N_j.dV + \sigma_a\int_{\mathcal{D}_i}N_iN_j.dV - \cancel{D\,\mathbf{n}\cdot\int_{\partial\mathcal{D}}N_i\boldsymbol{\nabla}N_j.dA} + \frac{\alpha}{\beta}\int_{\partial\mathcal{D}}N_iN_j.dA$$
$$b_i = b_i + q_{nc,j}\int_{\mathcal{D}_i}N_iN_j.dV + \frac{f_j}{\beta}\int_{\partial\mathcal{D}}N_ib_j.dV$$

With this notation we can see that by using a Robin boundary condition with $\alpha = 0$ and $\beta = -1$ we can essentially specify a Neumann boundary condition.

The versatility of this boundary condition can also be extended to **Vacuum** boundary conditions in neutron diffusion which take the form

$$\frac{1}{4}\phi + \frac{1}{2}D\mathbf{n}\cdot\boldsymbol{\nabla}\phi = 0 \qquad \text{on } \partial\mathcal{D}$$

representing a zero incoming current. It is simple to see that using the values $\alpha = \frac{1}{4}$, $\beta = \frac{1}{2}$ and $f = 0$ one can specify a vacuum boundary condition using a Robin boundary condition.

3.2.2 Dirichlet type boundary conditions

The Robin type boundary conditions does not have the potential to destroy the symmetry of the matrix. **Dirichlet** boundary conditions on the other hand do have this potential. The Dirichlet boundary conditions takes the simple form

$$\phi = c \qquad \text{on } \partial\mathcal{D}.$$

Since none of the weak form equations have components of this form there is only one possible contribution to a_{ij} and that is

$$a_{ii}{=}1 \qquad a_{ij} = 0$$
$$b_i = c$$

Now, it is fairly trivial in theory to apply this process, however, with the finite element method normally assembling the matrix element-by-element the dirichlet boundary conditions will have to be applied after the element-by-element assembly. Additionally, the dirichlet process zeros out the non-diagonal columns of the given row. This is a problem because the rest of the element-by-element assembly connected other DOFs to the dirichlet DOF via the columns of their respective rows and hence if we do but zero out the non-diagonal components of the matrix row then we are left with a **non-symmetric** matrix.

In ChiTech the whole mess of Dirichlet boundary conditions is handled by modifying the element-by-element assembly process to never connect DOFs to the known dirichlet nodes. For a better understanding of how this is done please consult the coding implementation section.

3.3 Sparsity pattern

3.3.1 Base block

Let us now consider a simple 2D arrangement of 4×4 cells as shown in Figure 6 below. For a continuous finite element method the solution is defined on the nodes of the mesh whereas the cells are distributed on processors depicted with colors. This brings some choice on which processor will own a given node when shared.

```
auto pwl = new SpatialDiscretization_PWL(0,
        chi_math::SpatialDiscretizationType::PIECEWISE_LINEAR_CONTINUOUS);
spatial_discretization = pwl;

pwl->AddViewOfLocalContinuum(grid);
pwl->OrderNodesCFEM(grid);
```

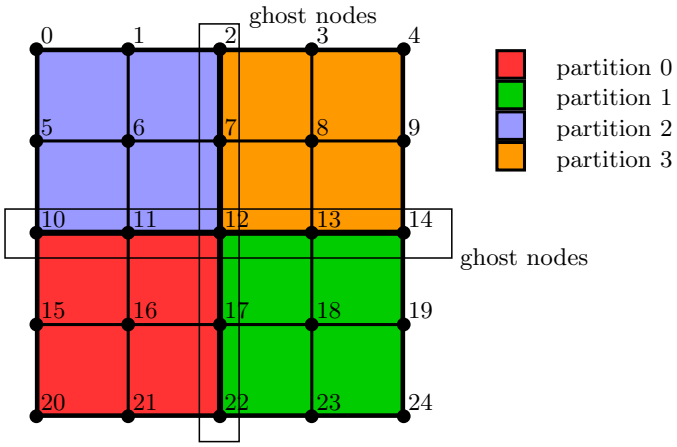

Figure 6: Simple CFEM arrangement of cells colored by processor ownership.

Nodal re-ordering is applied to the nodes of the mesh with a call to (ReorderNodesPWLC). This process is a two stage process:

Stage 1:

- First all the nodes relevant to a process is collected into a set of exclusive and non-exclusive nodes. This involves a loop over all the nodes associated with a local cell.
- Another loop is executed over local cells, however, this time we loop over faces and then face nodes. If a face is on a process boundary then node indices associated with all the face nodes are flagged as being ghosted.
- Using the ghost flags, a list of exclusive nodes is creater as well as a list of non-exclusive nodes.
- A ring communication is then used to communicate all the ghost nodes from location i to location $i+1$ with the last location ending up with the complete list of ghost nodes.

- The last location then broadcasts the completed ghost node set to all other locations.

Stage 2:

- The ghost nodes are broken into $2P-2$ pieces (2 pieces per location, first and last locations get only 1 piece). If the amount of ghost nodes are not divisible they are stored as the remainder which will get subdivided between the first and last location.
- With this information in hand each processor can determine the portion of the matrix it owns provided it knows the starting row. At this point only the first processor knows its ownership start ... its row 0. Therefore we initiate another ring communication. Each location takes its starting location, adds the local exclusive nodes, then the portion of the ghost nodes its been given and then sends the next location its starting location.
- Perform the mapping of original node indexes to distributed node indices.

This process can be visualized as depicted in Figure 7 below. The ordering allows for minimal communication between processes and overall low bandwidth when the amount of rows per processor is small. If further bandwidth reduction is required then a suitable reordering is required per process to reorder the exclusive nodes.

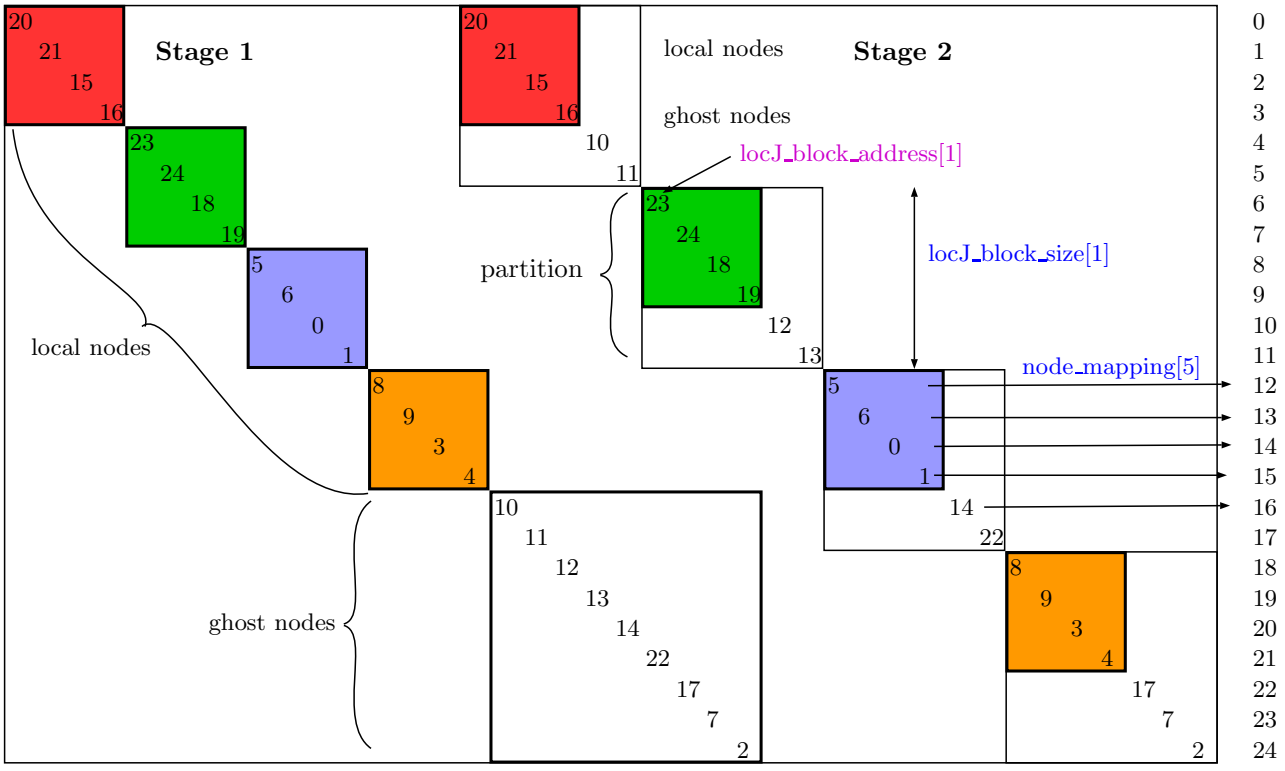

Figure 7: Stages of node ordering for CFEM.

After nodal reordering all partition locations will have three populated arrays, `node_mapping` , `locJ_block_address` and `locJ_block_size` .

The `node_mapping` array maps a global node to the new parallel ordering and is indexed by node global id whereas the latter two are indexed by partition-id (i.e., processor 15: `locJ_block_address[15]`).

The `locJ_block_address` array contains the global address where each partition's base block starts and the `locJ_block_size` array contains each partitions base block size.

3.3.2 Multiblock

For multiple degrees-of-freedom per node Chi-Tech support two storage modes: Nodal and Block. Nodal-storage creates a block of unknowns per node, i.e., given a node address, the DOF-addresses will follow sequentially for each unknown. In contrast, Block-storage creates a block for each unknown, i.e., each block contains all the nodes but only a single unknown. Multiblock addressing is applied as shown

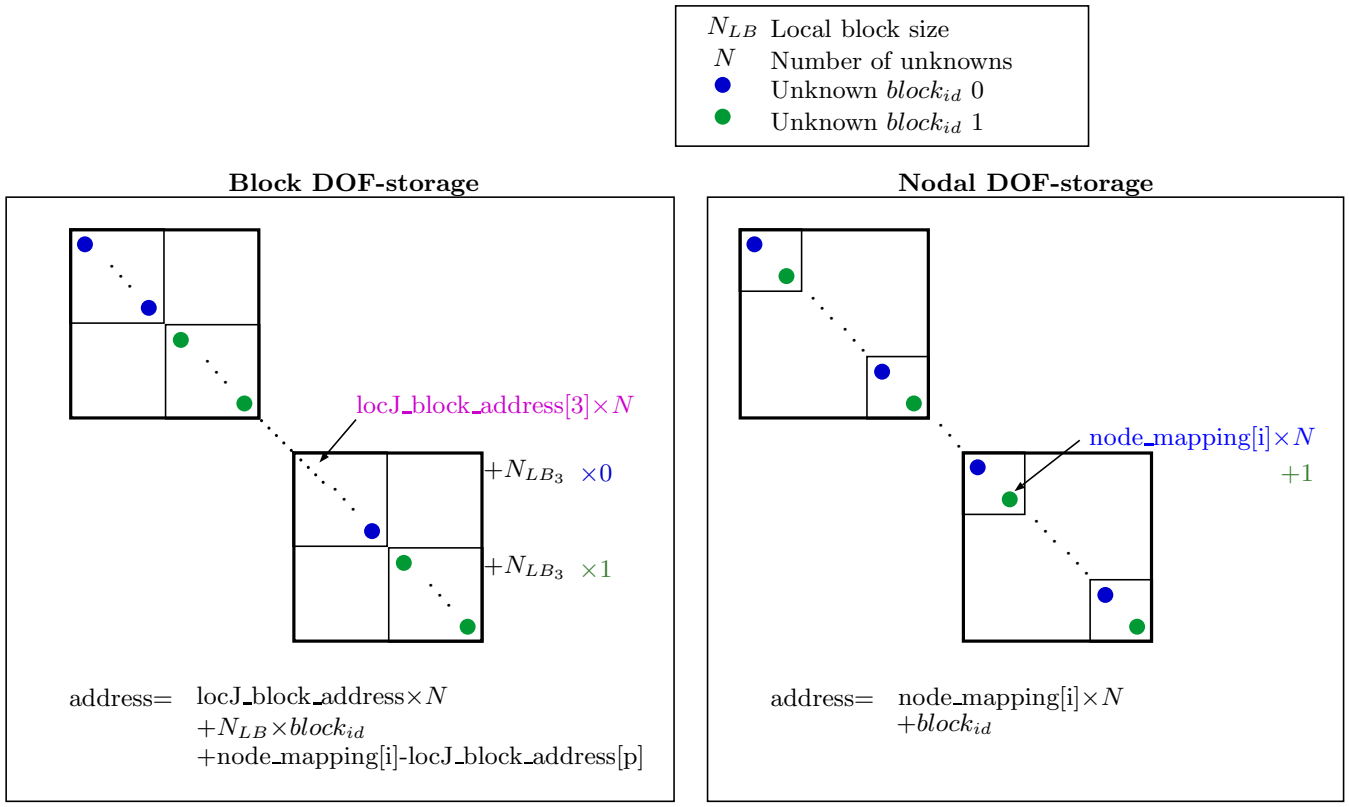

Figure 8: Addressing scheme for multiblock unknowns with the CFEM discretization method.

4 CFEM verification

This section establishes a suite of test cases to verify the accurate implementation of the CFEM method. For each dimension we will restrict the domain $r \in [-1, 1]$.

4.1 Slab 1D

Consider the one dimensional problem

$$-\frac{\partial}{\partial x}\frac{\partial \phi}{\partial x} = 1 \qquad \forall x \in [-1, 1]$$

We will apply different boundary conditions to test the implementation.

4.1.1 Dirichlet boundary conditions

We now consider the problem

$$\begin{aligned} -\frac{\partial}{\partial x}\frac{\partial \phi}{\partial x} &= 1 \qquad \forall x \in [-1, 1] \\ \phi(-1) &= 1 \qquad \text{on } \partial\mathcal{D} \\ \phi(1) &= 1 \qquad \text{on } \partial\mathcal{D} \end{aligned}$$

which has the analytical solution

$$\phi(x) = -\frac{1}{2}x^2 + \frac{3}{2}$$

A comparison of the solutions is shown in Figure 9. The comparison looks good.

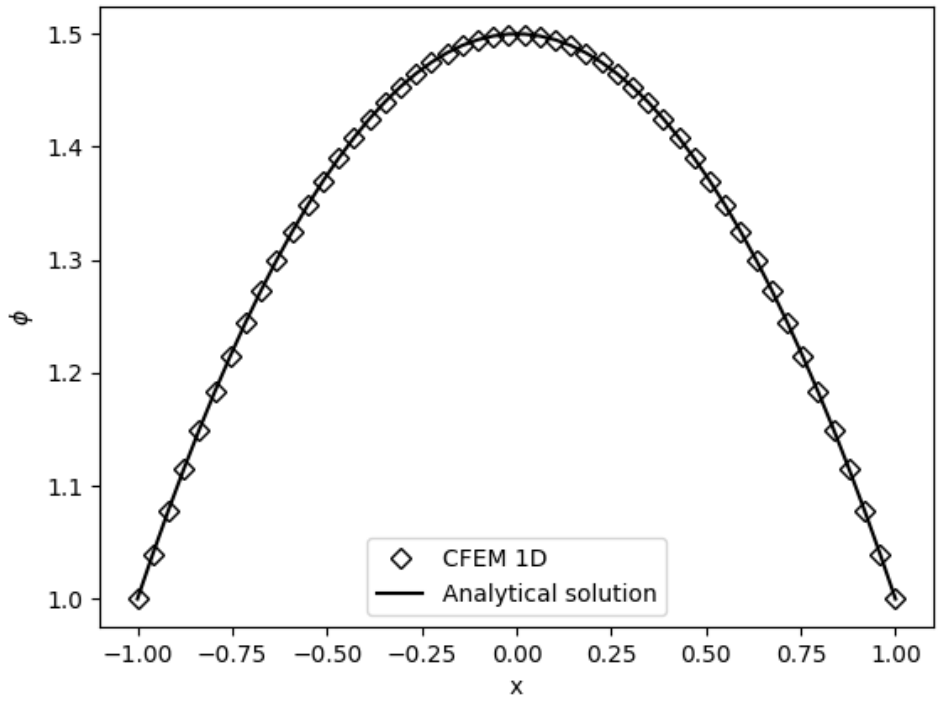

Figure 9: Comparison of CFEM solution to analytical solution.

4.1.2 Reflecting boundary condition

Implementing a reflecting boundary condition cannot involve both boundaries being reflecting or else the system will be indeterminite. Therefore we will take the previous analytical solution and attempt to simulate the right half.

$$
\begin{aligned}
-\frac{\partial}{\partial x}\frac{\partial \phi}{\partial x} &= 1 \qquad \forall x \in [-1, 1] \\
\left.\frac{\partial \phi}{\partial x}\right|_{x=0} &= 0 \qquad \text{on } \partial\mathcal{D} \\
\phi(1) &= 1 \qquad \text{on } \partial\mathcal{D}
\end{aligned}
$$

The analytical solution is still

$$\phi(x) = -\frac{1}{2}x^2 + \frac{3}{2}$$

The results are shown in Figure 10 below.

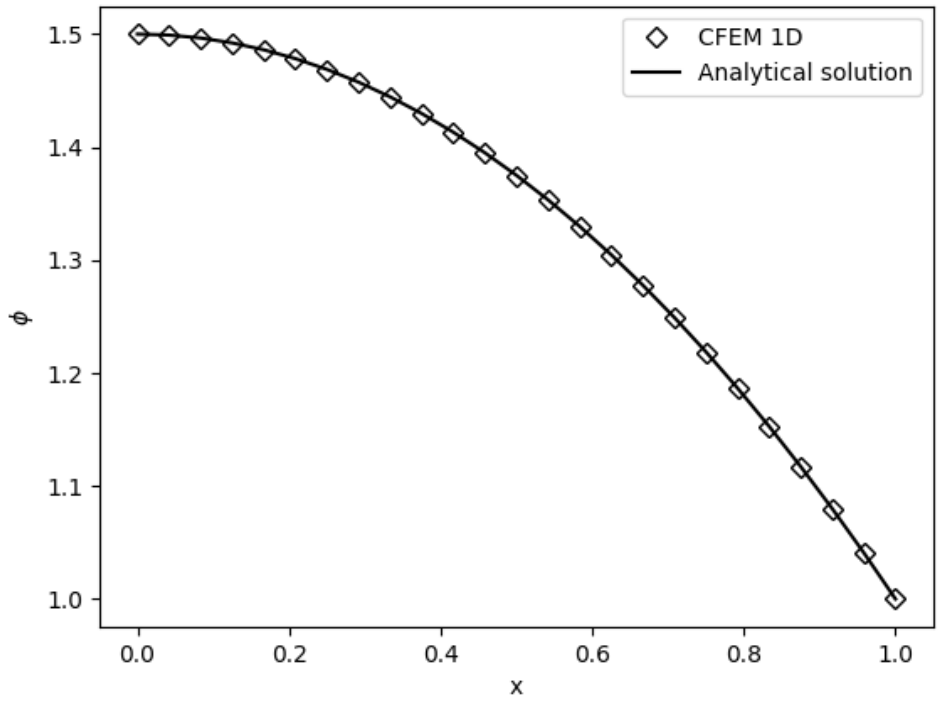

Figure 10: Comparison of CFEM solution to analytical solution using a reflecting boundary.

4.1.3 Vacuum boundary conditions

For vacuum boundary conditions we deal with the following

$$-\frac{\partial}{\partial x}\frac{\partial \phi}{\partial x} = 1 \qquad \forall x \in [-1, 1]$$
$$\frac{1}{4}\phi + \frac{1}{2}D\hat{n} \cdot \frac{\partial \phi}{\partial x} = 0 \qquad \text{on } \partial\mathcal{D}$$

The results are shown in Figure 11.

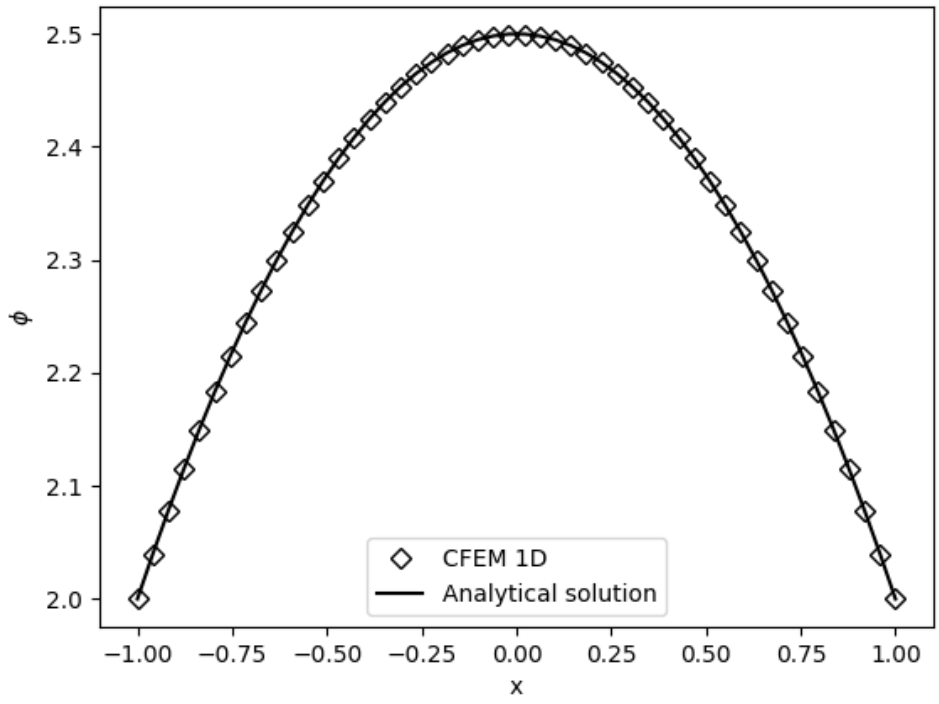

Figure 11: Comparison of CFEM solution to analytical solution using vacuum boundary conditions.

5 Modified Interior Penalty

Instead of a Continuous Finite Element Method (CFEM) discretization, where the solution is defined on nodes, the Discontinuous Finite Element Method (DFEM) stores values per cell. More specifically, using Piecewise Linear (PWL) basis functions and trial functions, defines the solution per cell node. The complication with using this formulation is then that the connectivity between cells is not as easy to determine as it was for CFEM discretization.

The weak form of the diffusion equation is extended as can be observed in the paper by Ragusa, repeated here, the interior penalty method matrix takes on the form

$$\begin{aligned} a(\delta\phi, b_i) = &(\sigma_a \delta\phi, b_i)_{\mathcal{D}} + (D\nabla\delta\phi, \nabla b_i)_{\mathcal{D}} \\ &+ (\kappa^{MIP}[\![\delta\phi]\!], [\![b_i]\!])_f + (\{\!\{D\partial_n b_i\}\!\}, [\![\delta\phi]\!])_f + (\{\!\{D\partial_n\delta\phi\}\!\}, [\![b_i]\!])_f \\ &+ (\kappa^{MIP}\delta\phi, b_i)_{\partial\mathcal{D}} - \frac{1}{2}(\delta\phi, \mathrm{D}\partial_n b_i)_{\partial\mathcal{D}} - \frac{1}{2}(\mathrm{D}\partial_n\delta\phi, b_i)_{\partial\mathcal{D}} \end{aligned}$$

where the jump and average operators are defined across an interface as

$$\begin{aligned} [\![u]\!] &= u^+ - u^- \\ \{\!\{u\}\!\} &= \frac{1}{2}(u^+ + u^-) \end{aligned}$$

The definition of κ^{MIP} is discussed in a later section. The $\pm$ is associated with the sense the given cell has with the given face, i.e. if the face has the righthand-rule convention and consistency with the normal then the cell that has a negative sense to it is the cell to which this convention is consistent. For our purposes we will denote $cell^-$ as the "current"-cell and $cell^+$ as the "adjacent"-cell.

Other notations used here are the volume integrals, $(F)_{\mathcal{D}}$, integration over interior faces $(F)_f$, and integration over face exterior faces $(F)_{\partial\mathcal{D}}$ on the boundary of the domain.

To assemble the matrix entries using this formulation we can replace $\delta\phi$ with b_j to find

$$\begin{aligned} a(b_j, b_i) = &(\sigma_a b_j, b_i)_{\mathcal{D}} + (D\nabla b_j, \nabla b_i)_{\mathcal{D}} \\ &+ (\kappa^{MIP}[\![b_j]\!], [\![b_i]\!])_f + (\{\!\{D\partial_n b_i\}\!\}, [\![b_j]\!])_f + (\{\!\{D\partial_n b_j\}\!\}, [\![b_i]\!])_f \\ &+ (\kappa^{MIP}b_j, b_i)_{\partial\mathcal{D}} - \frac{1}{2}(b_j, \mathrm{D}\partial_n b_i)_{\partial\mathcal{D}} - \frac{1}{2}(\mathrm{D}\partial_n b_j, b_i)_{\partial\mathcal{D}} \end{aligned}$$

Let us now develop these terms part-by-part. Imagine 3 terms, corresponding to the terms that have either $[\![]\!]$ or $\{\!\{\}\!\}$, which will be termed part A, B and C.

5.1 Part A

Part A then becomes

$$(\kappa^{MIP}[\![b_j]\!],[\![b_i]\!])_f = \int_f \kappa^{MIP}[\![b_j]\!],[\![b_i]\!].dA$$

Now, for simplicity let us replace κ^{MIP} with K and only focus on the terms inside the integral, part A now becomes

$$\begin{aligned}
&\kappa^{MIP}[\![b_j]\!],[\![b_i]\!] \\
&= K(b_j^+ - b_j^-)(b_i^+ - b_i^-) \\
&= K(b_i^+ - b_i^-)b_j^+ - K(b_i^+ - b_i^-)b_j^- \\
&= K(b_i^+ - b_i^-)b_j^+ \quad + \quad \boxed{K(b_i^- - b_i^+)b_j^-}.
\end{aligned} \tag{5.1}$$

The assembly of the interface terms into the matrix is a very complicated process. Each interface has to update each cell belonging to the interface. This is further complicated by the nominal way in which the matrix is normally assembled, i.e. cell-by-cell. Fortunately, some symmetry exists within the different parts as can be seen in part A above. The boxed portion in the above code is symmetric to the unboxed portion with respect to the cell with a negative sense to the face. In other words if we loop cell by cell and only execute the boxed portion, this will be equivalent to looping over the interfaces and updating both sides.

Given we computed K we can now calculate the following on the interface based on our current cell location ($cell^-$)

$$\begin{aligned}
&(\kappa^{MIP}[\![b_j]\!],[\![b_i]\!])_{E_h^i} \\
&\text{for } i,j \text{ on current cell} \\
&\text{for } i^* \text{ on adjacent cell} \\
a_{i_r j_r} &= \left[\kappa^{MIP}\int_{S_f} b_i.b_j.dA\right]_{cur-cell} \\
a_{i_r^* j_r} &= \left[-\kappa^{MIP}\int_{S_f} b_{i^*}.b_j.dA\right]_{adj-cell},
\end{aligned} \tag{5.2}$$

where i_r and j_r refers to the global matrix row and columns as mapped from the local matrices. The coding implementation for this is shown on the next page.

The implementation features a primary loop over trial space i followed by a secondary loop over basis functions j. Since we are dealing only with the shape functions we loop over face dofs and map the face DOFs to cell DOFs using the data **edge_dof_mappings**, developed during FE initialization. We also use the "interior penalty"-view of the cell to determine the matrix row (ir) corresponding to this cell's DOF-i. Since we are dealing with b_i^* we also have to determine i_r^*, and hence we determine i_{map} which is the adjacent cell's DOF-i index that corresponds to the current cell's DOF-i. The mapping of i_{map} is then

used with the adjacent cell's "interior penalty"-view to map to i_r^*. A similar procedure is applied to the b_j components but this time we are not dealing with indices on the adjacent cell and therefore only j_r is mapped. The values are inserted into global matrix using PETSc's **MatSetValue** which need not refer to local values only.

```
//============================= Assemble penalty terms
for (int fi=0; fi<num_face_dofs; fi++)
{
  int i  = fe_view->edge_dof_mappings[f][fi];
  int ir = cell_ip_view->MapDof(i);

  //Mapping face index to adj-cell
  int imap = MapCellDof(adj_cell,poly_cell->edges[f][fi]);
  int irstar = adj_ip_view->MapDof(imap);

  for (int fj=0; fj<2; fj++)
  {
    int j  = fe_view->edge_dof_mappings[f][fj];
    int jr = cell_ip_view->MapDof(j);

    double aij = kappa*fe_view->IntS_shapeI_shapeJ[f][i][j];

    MatSetValue(A,ir    ,jr, aij,ADD_VALUES);
    MatSetValue(A,irstar,jr,-aij,ADD_VALUES);
  }//for fj

}//for fi
```

5.2 Part B

Part B is

$$([\![b_j]\!],\{\!\{D\partial_n b_i\}\!\})_f = \int_f [\![b_j]\!],\{\!\{D\partial_n b_i\}\!\}.dA$$

Let us now expand the terms inside the integral

$$\begin{aligned}
&[\![b_j]\!],\{\!\{D\partial_n b_i\}\!\} \\
&= \frac{1}{2}\left(b_j^+ - b_j^-\right)\left(D^+\hat{n}\cdot\nabla b_i^+ + D^-\hat{n}\cdot\nabla b_i^-\right) \\
&= \frac{1}{2}\left(D^+\hat{n}\cdot\nabla b_i^+ + D^-\hat{n}\cdot\nabla b_i^-\right)b_j^+ - \frac{1}{2}\left(D^+\hat{n}\cdot\nabla b_i^+ + D^-\hat{n}\cdot\nabla b_i^-\right)b_j^- \\
&= \frac{1}{2}\left(D^+\hat{n}\cdot\nabla b_i^+ + D^-\hat{n}\cdot\nabla b_i^-\right)b_j^+ \quad \boxed{-\frac{1}{2}\left(D^+\hat{n}\cdot\nabla b_i^+ + D^-\hat{n}\cdot\nabla b_i^-\right)b_j^-}
\end{aligned}$$

The blocked term in this equation is symmetric to the non-blocked terms with respect to the current cell and the adjacent cell. In other words, the normal ($\hat{n}$) in the blocked portion is with respect to the current cell ($cell^-$). When we flip the sign of all the $\pm$ denotations and set $\hat{n} = -\hat{n}$ then we obtain the non-blocked

terms. Some of the terms in the blocked portions can be combined with others so let us defer showing the code for that for now and proceed to part C.

5.3 Part C

Part C is

$$(\{\{D\partial_n b_j\}\},[\![b_i]\!])_f = \int_f \{\{D\partial_n b_j\}\},[\![b_i]\!].dA$$

Let us now expand the terms inside the integral

$$\begin{aligned}
&\{\{D\partial_n b_j\}\},[\![b_i]\!] \\
&= \frac{1}{2}\Big(b_i^+ - b_i^-\Big)\Big(D^+\hat{n}\cdot\nabla b_j^+ + D^-\hat{n}\cdot\nabla b_j^-\Big) \\
&= \frac{1}{2}\Big(D^+\hat{n}\cdot\nabla b_j^+ + D^-\hat{n}\cdot\nabla b_j^-\Big)b_i^+ - \frac{1}{2}\Big(D^+\hat{n}\cdot\nabla b_j^+ + D^-\hat{n}\cdot\nabla b_j^-\Big)b_i^- \\
&= \frac{1}{2}\Big(D^+\hat{n}\cdot\nabla b_j^+ + D^-\hat{n}\cdot\nabla b_j^-\Big)b_i^+ \quad \boxed{-\frac{1}{2}\Big(D^+\hat{n}\cdot\nabla b_j^+ + D^-\hat{n}\cdot\nabla b_j^-\Big)b_i^-}
\end{aligned}$$

Again the same symmetry applies as with part B (i.e. flipping the denotations and the signs on the normals).

5.4 Assembling B and C

We first look at the blocked parts of B and C together

$$\begin{aligned}
&\boxed{-\frac{1}{2}\Big(D^+\hat{n}\cdot\nabla b_i^+ + D^-\hat{n}\cdot\nabla b_i^-\Big)b_j^-} \quad \boxed{-\frac{1}{2}\Big(D^+\hat{n}\cdot\nabla b_j^+ + D^-\hat{n}\cdot\nabla b_j^-\Big)b_i^-} \\
&= -\frac{1}{2}b_j^- D^+\hat{n}\cdot\nabla b_i^+ \quad -\frac{1}{2}b_j^- D^-\hat{n}\cdot\nabla b_i^- \quad -\frac{1}{2}b_i^- D^+\hat{n}\cdot\nabla b_j^+ \quad -\frac{1}{2}b_i^- D^-\hat{n}\cdot\nabla b_j^-
\end{aligned}$$

We now reshuffle the terms here after first noting that the second and last terms are the transpose of each other as well as the first and third.

$$= -\frac{1}{2}b_j^- D^-\hat{n}\cdot\nabla b_i^- \quad -\frac{1}{2}b_i^- D^-\hat{n}\cdot\nabla b_j^- \quad -\frac{1}{2}b_j^- D^+\hat{n}\cdot\nabla b_i^+ \quad -\frac{1}{2}b_i^- D^+\hat{n}\cdot\nabla b_j^+$$

We now reintroduce the surface integrals

$$\begin{aligned}
&\int_f\Big[-\frac{1}{2}b_j^- D^-\hat{n}\cdot\nabla b_i^- \quad -\frac{1}{2}b_i^- D^-\hat{n}\cdot\nabla b_j^- \quad -\frac{1}{2}b_j^- D^+\hat{n}\cdot\nabla b_i^+ \quad -\frac{1}{2}b_i^- D^+\hat{n}\cdot\nabla b_j^+\Big].dA \\
&= -\frac{1}{2}D^-\hat{n}\cdot\int_f\Big(b_j^-\cdot\nabla b_i^- + b_i^-\cdot\nabla b_j^-\Big).dA \quad -\frac{1}{2}D^+\hat{n}\cdot\int_f\Big(b_j^-\cdot\nabla b_i^+\Big).dA \quad -\frac{1}{2}D^+\hat{n}\cdot\int_f\Big(b_i^-\cdot\nabla b_j^+\Big).dA
\end{aligned}$$

Theoretically the two right hand integrals could have been lumped in a similar fashion to the left-most integral, however, this segregation proved useful for efficiency in the looping structure with minimal mapping. We therefore have the following three equations to implement

$$a_{i_r,j_r} = -\frac{1}{2}D^- \hat{n} \cdot \int_f \left(b_j^- \cdot \nabla b_i^- + b_i^- \cdot \nabla b_j^- \right) .dA \tag{5.3}$$

$$a_{i_r^*,j_r} = -\frac{1}{2}D^+ \hat{n} \cdot \int_f \left(b_j^- \cdot \nabla b_i^+ \right) .dA \tag{5.4}$$

$$a_{i_r,j_r^*} = -\frac{1}{2}D^+ \hat{n} \cdot \int_f \left(b_i^- \cdot \nabla b_j^+ \right) .dA \tag{5.5}$$

The coding implementation is shown below:

```
// -Di^- bj^- and
// -Dj^- bi^-
for (int i=0; i<fe_view->dofs; i++)
{
  int ir = cell_ip_view->MapDof(i);

  for (int j=0; j<fe_view->dofs; j++)
  {
    int jr = cell_ip_view->MapDof(j);

    double gij =
      n.Dot(fe_view->IntS_shapeI_gradshapeJ[f][i][j] +
            fe_view->IntS_shapeI_gradshapeJ[f][j][i]);
    double aij = -0.5*diffCoeff*gij;

    MatSetValue(A,ir,jr,aij,ADD_VALUES);
  }//for j
}//for i
```

```
// - Di^+ bj^-
for (int imap=0; imap<adj_fe_view->dofs; imap++)
{
  int irmap = adj_ip_view->MapDof(imap);

  for (int fj=0; fj<num_face_dofs; fj++)
  {
    int jmap  = MapCellDof(adj_cell,poly_cell->edges[f][fj]);
    int j     = MapCellDof(poly_cell,poly_cell->edges[f][fj]);
    int jr    = cell_ip_view->MapDof(j);

    double gij =
      n.Dot(adj_fe_view->IntS_shapeI_gradshapeJ[fmap][jmap][imap]);
    double aij = -0.5*diffCoeff*gij;

    MatSetValue(A,irmap,jr,aij,ADD_VALUES);
  }//for j
}//for i
```

```
// - Dj^+ bi^-
for (int jmap=0; jmap<adj_fe_view->dofs; jmap++)
{
  int jrmap = adj_ip_view->MapDof(jmap);

  for (int fi=0; fi<num_face_dofs; fi++)
  {
    int imap  = MapCellDof(adj_cell,poly_cell->edges[f][fi]);
    int i     = MapCellDof(poly_cell,poly_cell->edges[f][fi]);
    int ir    = cell_ip_view->MapDof(i);

    double gij =
      n.Dot(adj_fe_view->IntS_shapeI_gradshapeJ[fmap][imap][jmap]);
    double aij = -0.5*diffCoeff*gij;

    MatSetValue(A,ir,jrmap,aij,ADD_VALUES);
  }//for j
}//for i
```

References

[1] Moukalled F., Mangani L., Darwish M., *The Finite Volume Method in Computational Fluid Dynamics - An Advanced Introduction with OpenFOAM and Matlab*, Springer, 2016.

[2] Sezai I., *Implementation of boundary conditions in pressure-based finite volume methods on unstructured grids*, Numerical Heat Transfer, Part B: Fundamentals, 2017

[3] *Blender - a 3D modelling and rendering package*, Blender Online Community, Blender Foundation, Blender Institute, Amsterdam, 2018

[4] Cheng et al, *Delaunay Mesh Generation*, Chapman & Hall/CRC Computer & Information Science Series, 2013