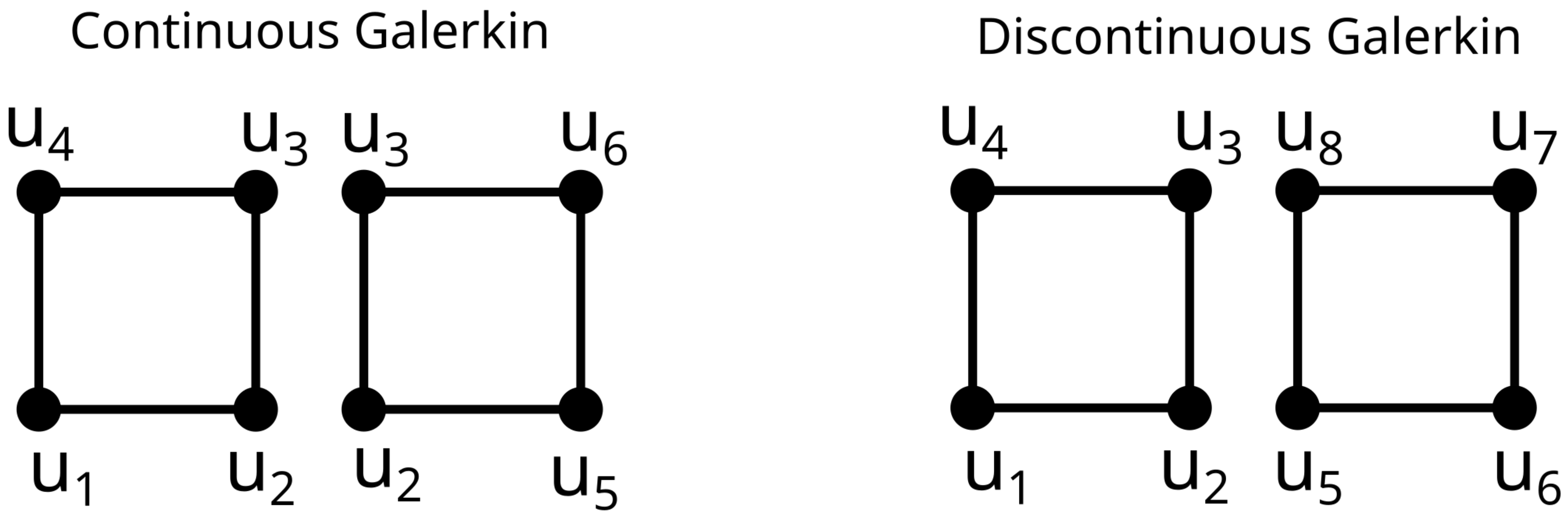
Ferrite.jl

Discontinuous Galerkin Methods Infrastructure: A GSoC Project

Abdulaziz Hamid - GSoC 2023 contributor @ Ferrite.jl - CompEng student

Background on DG

- Degrees of freedom on the interface are not shared



Background on DG

- Degrees of freedom on the interface are not shared
- Interface integral term exists in the weak form
    - Usually in form of $\sum_K \int_{\partial K} \nu \hat{\sigma} \cdot n ds$

Where ν is the test function, $\hat{\sigma}$ is the numerical flux, and n is the normal to the current side of the interface.

Background on DG

- Degrees of freedom on the interface are not shared
- Interface integral term exists in the weak form
  - Usually in form of $\sum_K \int_{\partial K} \nu\hat{\sigma} \cdot n ds$
  - Introduces jumps and averages

$$\sum_K \int_{\partial K} \nu\hat{\sigma} \cdot n ds = \int_\Gamma [\![\nu]\!] \cdot \{\hat{\sigma}\} ds + \int_{\Gamma^0} \{\nu\} [\![\hat{\sigma}]\!] ds$$

Where

$$\{u\} = \frac{1}{2}(u^+ + u^-), \quad [[u]] = u^+ \cdot n^+ + u^- \cdot n^-$$

Background on DG

- Degrees of freedom on the interface are not shared
- Interface integral term exists in the weak form
  - Usually in form of $\sum_K \int_{\partial K} \nu \hat{\sigma} \cdot n ds$
  - Introduces jumps and averages
  - Quadrature points on the interface must be synced

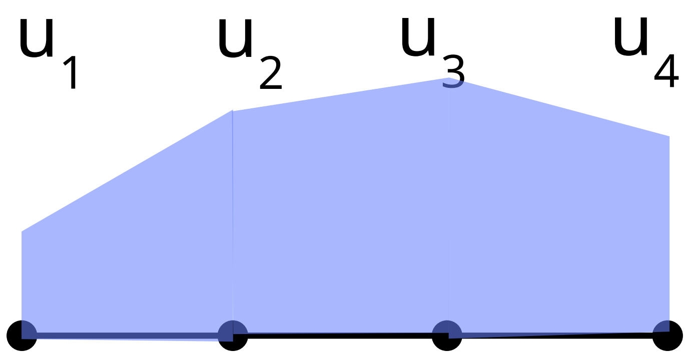
Required modifications

- Sparsity Patterns
- Constraints
- Assembly
  - Iterators
  - Jumps and Averages
  - Quadrature points

Sparsity patterns

Elements are coupled using shared dofs in Continuous Galerkin

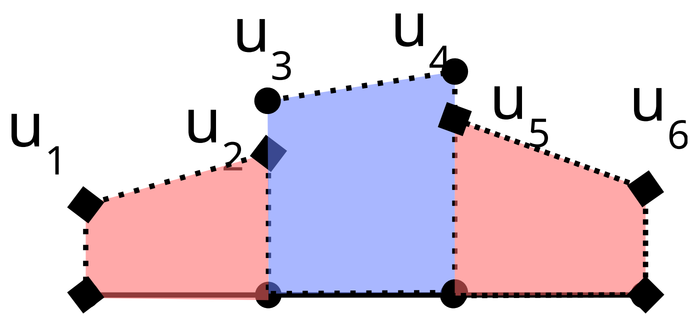
```
julia> K = create_sparsity_pattern(dh)
4×4 SparseArrays.SparseMatrixCSC{Float64,
 Int64} with 10 stored entries:
 0.0  0.0   .    .
 0.0  0.0  0.0   .
  .   0.0  0.0  0.0
  .    .   0.0  0.0
```



Sparsity patterns

Elements don't share dofs in DG, thus are coupled using numerical flux in the interface integral term.

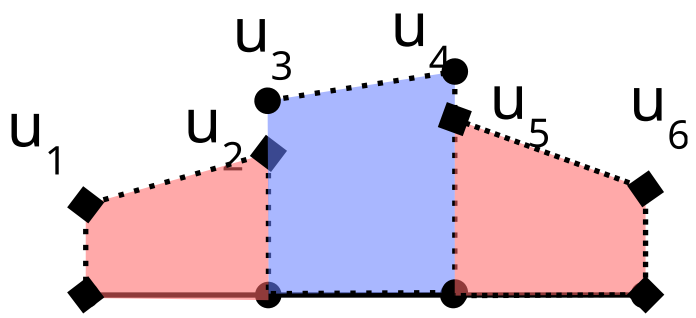
```
julia> K = create_sparsity_pattern(dh)
6×6 SparseArrays.SparseMatrixCSC{Float64,
 Int64} with 12 stored entries:
 0.0  0.0   ⋅    ⋅    ⋅    ⋅
 0.0  0.0   ⋅    ⋅    ⋅    ⋅
  ⋅    ⋅   0.0  0.0   ⋅    ⋅
  ⋅    ⋅   0.0  0.0   ⋅    ⋅
  ⋅    ⋅    ⋅    ⋅   0.0  0.0
  ⋅    ⋅    ⋅    ⋅   0.0  0.0
```



Sparsity patterns

Elements don't share dofs in DG, thus are coupled using numerical flux in the interface integral term.

```
julia> K = create_sparsity_pattern(dh;
 topology = topology, cross_coupling=trues(1,1))
6×6 SparseArrays.SparseMatrixCSC{Float64,
 Int64} with 28 stored entries:
 0.0  0.0  0.0  0.0   .    .
 0.0  0.0  0.0  0.0   .    .
 0.0  0.0  0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0  0.0
  .    .   0.0  0.0  0.0  0.0
  .    .   0.0  0.0  0.0  0.0
```



Sparsity patterns

Implementation

- `cross_element_coupling!`

Issues faced (solved)

- Type instablilities (i.e., `getnbasefunctions(fi::Interpolation)::Any`).
- Allocations

Constraints

- DG elements can have their dofs in the interior of the cell, thus dirichlet boundary conditions enforced using penalty terms.
- For elements with dofs on the boundary, strong enforcement is done using `DofHandler`

Constraints

Implementation

- `dirichlet_boundarydof_indices`
    - `dirichlet_(face|vertex|edge)dof_indices`
- `(face|vertex|edge)dof_indices` are empty for `DiscontinuousLagrange`.

Iterators

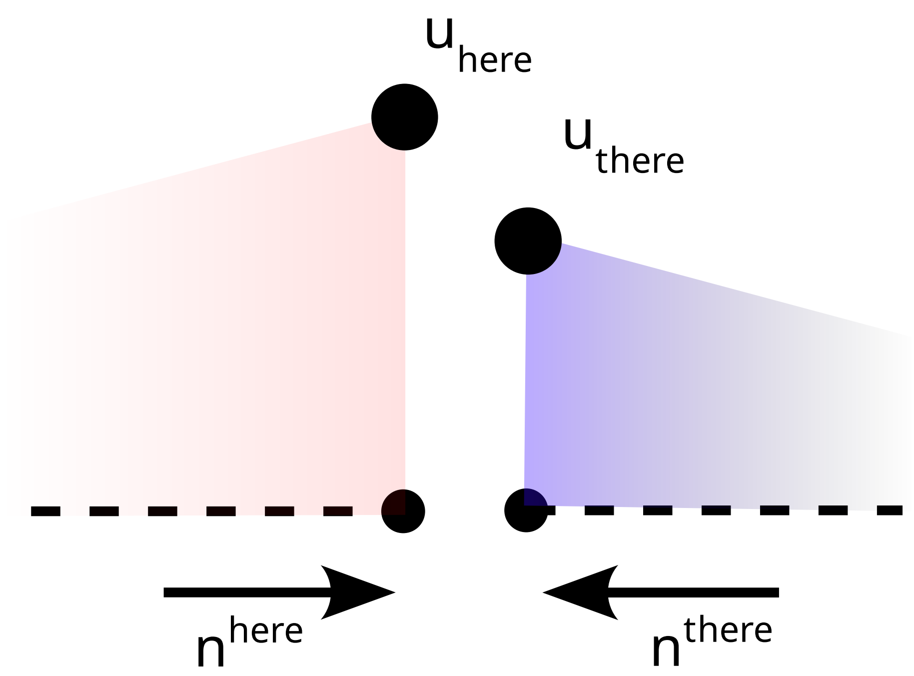
Implementation

- `InterfaceCache`
  - Two `FaceCache`s
  - dofs
- `InterfaceIterator`

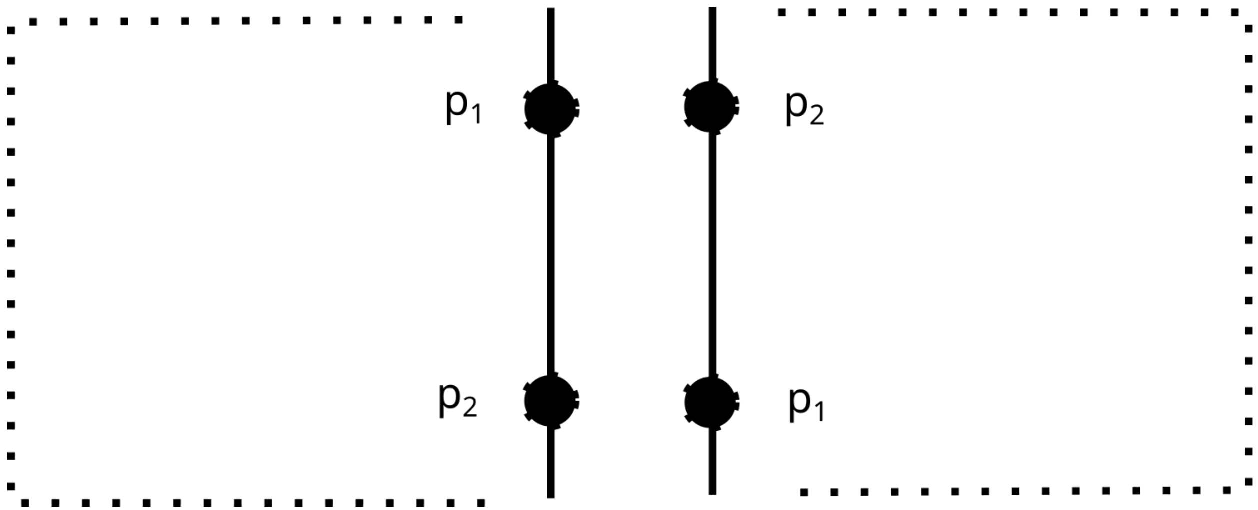
Jumps and averages

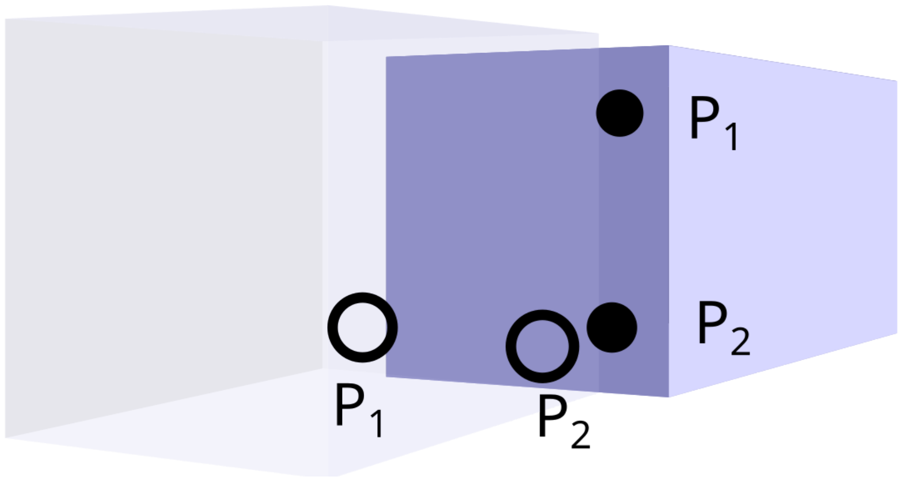
Implementation

- `InterfaceValues`
  - Two `FaceValue` s
- Jumps use $[[u]] = u^{\text{there}} - u^{\text{here}}$
- `(shape|function)_(value|gradient)_(jump|average)`



Syncing quadrature points





Syncing quadrature points

options:

- Transform using a transformation matrix.
- Permute the existing values using cached permutations.
- Cache values for each interface case.

Chosen:

- Transforming using a transformation matrix as other options can be too much caching.

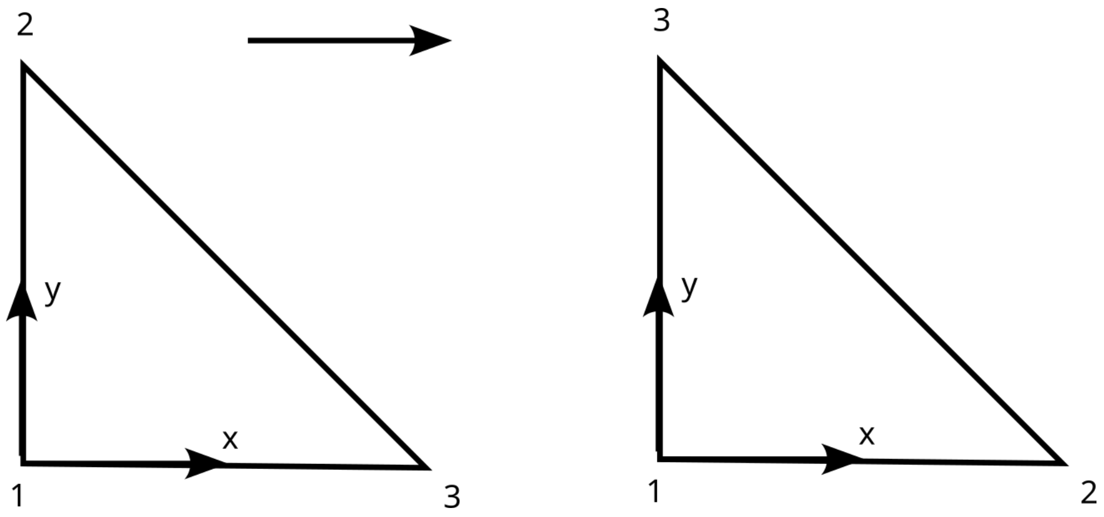
Syncing quadrature points

Implementation

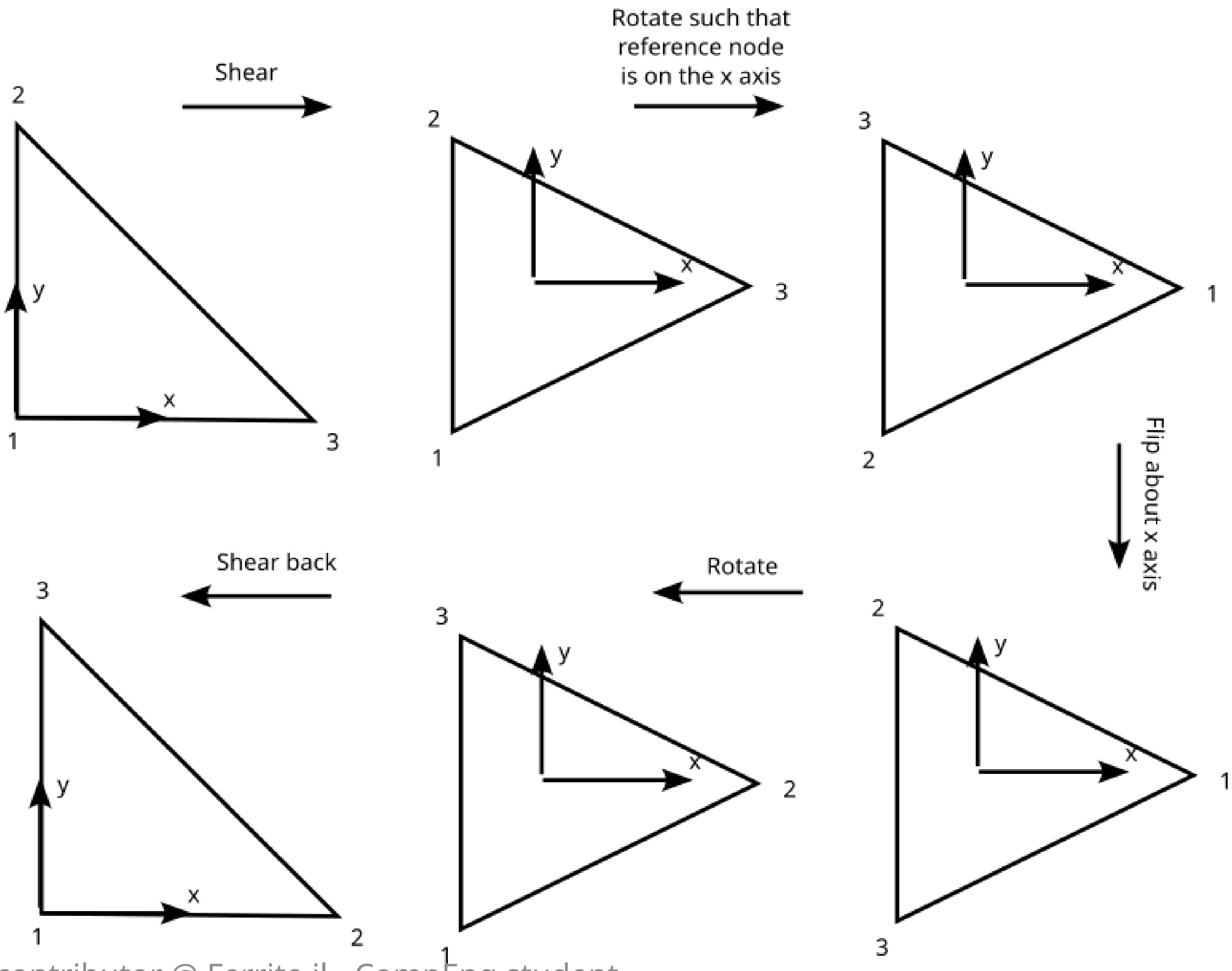
- `InterfaceTransformation` struct
- `get_transformation_matrix(::InterfaceTransformation)`
- `transform_interface_points!`
- quadrature points are transformed on each `reinit!`

Syncing quadrature points

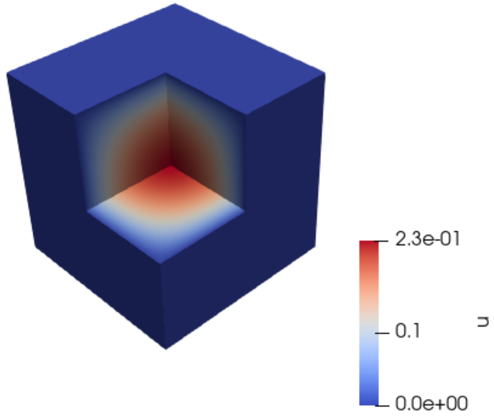
```
flipping = SMatrix{3,3}(1.0, 0.0, 0.0, 0.0, -1.0, 0.0, 0.0, 0.0, 1.0)

translate_1 = SMatrix{3,3}(1.0, 0.0, 0.0, 0.0, 1.0, 0.0, -sinpi(2/3)/3, -0.5, 1.0)
stretch_1 = SMatrix{3,3}(sinpi(2/3), 0.5, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 1.0)

translate_2 = SMatrix{3,3}(1.0, 0.0, 0.0, 0.0, 1.0, 0.0, sinpi(2/3)/3, 0.5, 1.0)
stretch_2 = SMatrix{3,3}(1/sinpi(2/3), -1/2/sinpi(2/3), 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 1.0)

return stretch_2 * translate_2 * rotation_matrix_pi(-θpre) * flipping * rotation_matrix_pi(θ + θpre) * translate_1 * stretch_1
```



Syncing quadrature points



Heat equation tutorial*



Interior penalty formulation

$$\int_{\Omega} \nabla u \cdot \nabla \delta u d\Omega - \int_{\Gamma} [[u]] \cdot \{\nabla \delta u\} + [[\delta u]] \cdot \{\nabla u\} d\Gamma + \int_{\Gamma} \mu [[u]].[[\delta u]] d\Gamma = \int_{\Omega} \delta u d\Omega,$$

> *based on "Unified Analysis of Discontinuous Galerkin Methods for Elliptic Problems" by Douglas N. Arnold, F. Brezzi, B. Cockburn, and L. Donatella Marini

Heat equation tutorial

Convergence test results:

```
[ Info: order = 1
[ Info: mean order of convergence for L2 = 1.996
[ Info: mean order of convergence for H1 = 0.999

[ Info: order = 3
[ Info: mean order of convergence for L2 = 3.986
[ Info: mean order of convergence for H1 = 2.997
```

- $\Delta Log_2(L2) \approx P + 1, \quad \Delta Log_2(H1) \approx P$

Future Work

- Arbitrary order interpolations (Done for `Lagrange` with hypercubes).
- Better method to work with mixed grids.
- Interface with AMR.