This is a short report for 16930 final project – an extension of HDG method to solve 2d unsteady burgers' equation. Please refer to the detailed writeup for a detailed derivation of every equation. As my slides are very throughout, I will copy them here but add descriptions to each.

Following equations are solved using HDG method

$$\frac{\partial u}{\partial t} + \nabla \cdot (\boldsymbol{f}(\boldsymbol{u}) - \kappa \cdot \nabla u) = s \qquad in\ K$$
$$u = g_D \qquad on\ \partial K$$

There are many options for the nonlinear advection term **f(u):**
Standard form:

$$\boldsymbol{f}(\boldsymbol{u}) = [0.5u^2, 0.5u^2] \quad \& \quad \frac{d\boldsymbol{f}(\boldsymbol{u})}{du} = [u, u]$$

Cubic form:

$$\boldsymbol{f}(\boldsymbol{u}) = [1/3u^3, 1/3u^3] \ \& \frac{d\boldsymbol{f}(\boldsymbol{u})}{du} = [u^2, u^2]$$

Exponential form:

$$\boldsymbol{f}(\boldsymbol{u}) = [e^u, e^u] \ \& \frac{d\boldsymbol{f}(\boldsymbol{u})}{du} = [e^u, e^u\ ]$$

Anisotropic form:

$$\boldsymbol{f}(\boldsymbol{u}) = [0, 1/2u^2] \ \& \frac{d\boldsymbol{f}(\boldsymbol{u})}{du} = [0, u\ ]$$

Linear form:

$$\boldsymbol{f}(\boldsymbol{u}) = [10u, 10u] \ \& \frac{d\boldsymbol{f}(\boldsymbol{u})}{du} = [10,10\ ]$$

The corresponding simulations are shown in slides animation.
Here is the breakdown to 1st order derivatives:

$$\frac{1}{\kappa}\boldsymbol{q} + \nabla u = 0 \qquad in\ K$$
$$\frac{\partial u}{\partial t} + \nabla \cdot (\boldsymbol{f}(\boldsymbol{u}) + \boldsymbol{q}) = f \ \ in\ K$$
$$u = g_D \qquad on\ \partial K$$

And the weak form in the residual form

$$R_1 = \left(\frac{1}{\kappa}\boldsymbol{q_h^t}, \boldsymbol{v}\right)_K - (u_h^t, \nabla \cdot \boldsymbol{v})_K + \left\langle \widehat{u_h^t}, \boldsymbol{v} \cdot \boldsymbol{n} \right\rangle_{\partial K} = 0 = f_1$$

$$R_2 = (\nabla \cdot \boldsymbol{q_h^t}, w)_K + \left(\frac{u_h^t}{\Delta t}, w\right)_K - (\boldsymbol{f}(\boldsymbol{u_h})^t, \nabla w)_K + \langle \tau \cdot u_h^t, w \rangle_{\partial K} + \left\langle \left(\boldsymbol{f}(\widehat{\boldsymbol{u_h^t}}) \cdot \boldsymbol{n} - \tau \cdot \widehat{u_h^t}\right), w \right\rangle_{\partial K}$$
$$= (s, w)_K + \left(\frac{u_h^{t-1}}{\Delta t}, w\right)_K = f_2$$

$$R_3 = \langle \boldsymbol{q_h^t} \cdot \boldsymbol{n}, \mu \rangle_{\partial K_{in}} + \langle \tau \cdot u_h^t, \mu \rangle_{\partial K_{in}} + \left\langle \left(f\left(\widehat{u_h^t}\right) \cdot n - \tau \cdot \widehat{u_h^t}\right), \mu \right\rangle_{\partial K_{in}} = 0 = f_3$$

When compute Jacobian of residuals, just replaces the following terms

- $-\left(\frac{df^t}{du} du_h^t, \nabla w\right)_K \rightarrow -(\boldsymbol{f}(\boldsymbol{u_h})^t, \nabla w)_K \rightarrow E_{non}$
- $\left\langle \left(\frac{df^t}{du} d\widehat{u_h^t} \cdot \boldsymbol{n} - \tau \cdot d\widehat{u_h^t}\right), w \right\rangle_{\partial K} \rightarrow \left\langle \left(\boldsymbol{f}(\widehat{\boldsymbol{u_h^t}}) \cdot \boldsymbol{n} - \tau \cdot \widehat{u_h^t}\right), w \right\rangle_{\partial K} \rightarrow I_{non}$
- $\left\langle \left(\frac{df^t}{du} d\widehat{u_h^t} \cdot \boldsymbol{n} - \tau \cdot d\widehat{u_h^t}\right), \mu \right\rangle_{\partial K_{in}} \rightarrow \left\langle \left(\boldsymbol{f}(\widehat{\boldsymbol{u_h^t}}) \cdot \boldsymbol{n} - \tau \cdot \widehat{u_h^t}\right), \mu \right\rangle_{\partial K_{in}} \rightarrow L_{non}$

Tau in this case is set to constant 1 for simplicity, so I don't have to deal with nonlinearity. But this has been causing stability problems. Namely, when advection speed increases, I also need to increase tau to enhance stability. But due to the lack of c*n, I don't really know how much I should increase tau. So, most of the time, it is through trial and error.

Newton iteration is implemented in the following format

$$[Jacobian]^{n-1} \cdot [dU] = [Source]^{t-1} - [Residual]^{n-1} \cdot [U]^{n-1}$$

Jacobian is the Jacobean of residual matrix.
Here is how nonlinear terms are represented in matrix form

- *Matrix Form*

$$R = \begin{bmatrix} A_x & 0 & B_x & C_x \\ 0 & A_y & B_y & C_y \\ D_x & D_y & E & I \\ N_x & N_y & K & L \end{bmatrix} \cdot \begin{bmatrix} q_{hx}^{n-1} \\ q_{hy}^{n-1} \\ u_h^{n-1} \\ \widehat{u_h^{n-1}} \end{bmatrix}$$

$$= \begin{bmatrix} A_x & 0 & B_x & C_x \\ 0 & A_y & B_y & C_y \\ D_x & D_y & E_{lin} & I_{lin} \\ N_x & N_y & K & L_{lin} \end{bmatrix} \cdot \begin{bmatrix} q_{hx}^{n-1} \\ q_{hy}^{n-1} \\ u_h^{n-1} \\ \widehat{u_h^{n-1}} \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & E_{non} & I_{non} \\ 0 & 0 & 0 & L_{non} \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} f_{1x} \\ f_{1y} \\ f_2 \\ f_3 \end{bmatrix}$$

- Linear Based Matrix
  - Never updated
- Nonlinear Addon Matrix
  - Updated every **Newton iteration (inner loop)**
- Source vector that has the t-1 term inside
  - Updated every **time iteration (outer loop)**

The benefit of doing that is it avoids recomputing the based matrix for each Newton iteration. And also, for the Jacobian matrix, only the addon matrix is different.
As the source term contains last time step information, it is updated every time step.
Written in HDG form is

$$([dL] - [dN_x \quad dN_y \quad dK]\begin{bmatrix} dA_x & 0 & dB_x \\ 0 & dA_y & dB_y \\ dD_x & dD_y & dE \end{bmatrix}^{-1}\begin{bmatrix} dC_x \\ dC_y \\ dI \end{bmatrix})[d\hat{u}_h]$$

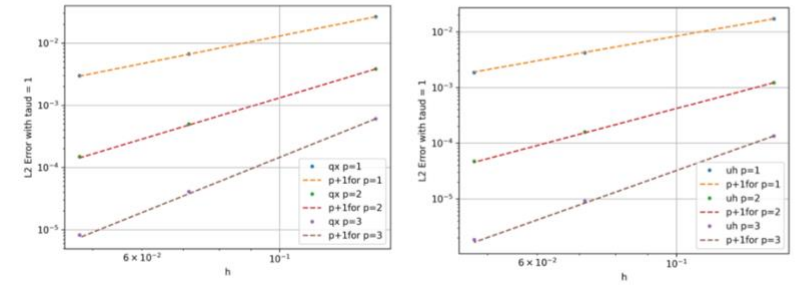
$$= [F_{tot\widehat{u_h}}] - [dN_x \quad dN_y \quad dK]\begin{bmatrix} dA_x & 0 & dB_x \\ 0 & dA_y & dB_y \\ dD_x & dD_y & dE \end{bmatrix}^{-1}\begin{bmatrix} F_{totq_x} \\ F_{totq_y} \\ F_{totu_h} \end{bmatrix}$$

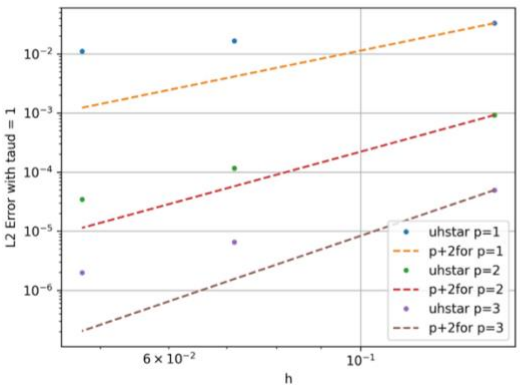
$$\begin{bmatrix} F_{totq_x} \\ F_{totq_y} \\ F_{totu_h} \\ F_{tot\widehat{u_h}} \end{bmatrix} = \begin{bmatrix} f_{1x} \\ f_{1y} \\ f_2 \\ f_3 \end{bmatrix} - \begin{bmatrix} A_x & 0 & B_x & C_x \\ 0 & A_y & B_y & C_y \\ D_x & D_y & E & I \\ N_x & N_y & K & L \end{bmatrix} \cdot \begin{bmatrix} q_{hx}^{n-1} \\ q_{hy}^{n-1} \\ u_h^{n-1} \\ \widehat{u_h^{n-1}} \end{bmatrix}$$

Here is how the convergence rate is tested:

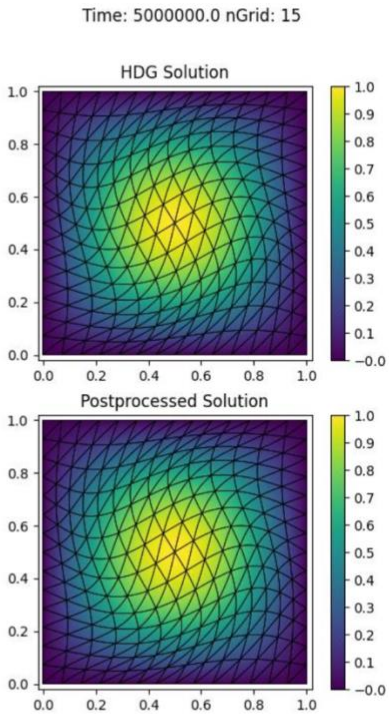
Convergence

- *Test Solution*: $u = \sin(\pi x)\sin(\pi y)$





Don't know what happened to post-processing



For both uniform and nonuniform grids, uh and qh are converging optimally. It is just the post-processed one acting weird. The post-processed solution for p=2 converged optimally for a uniform grid, and others didn't. I don't know the cause yet. Also it is noticed how for nonuniform grid, tau required is much higher.