Project 7: Discontinuous-Galerkin Solver

Changkun Li

1 Problem Description

1.1 Equation to solve

$$\frac{\partial}{\partial t}\begin{bmatrix} h \\ hu \\ hv \end{bmatrix} + \frac{\partial}{\partial x}\begin{bmatrix} hu \\ hu^2 + gh^2/2 \\ hvu \end{bmatrix} + \frac{\partial}{\partial y}\begin{bmatrix} hv \\ huv \\ hv^2 + gh^2/2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

1.2 DG weak form

$$\underbrace{\int_{\Omega_k} \phi_{k,i} \frac{\partial u}{\partial t} d\Omega}_{\textbf{Term 1}} \underbrace{- \int_{\Omega_k} \nabla \phi_{k,i} \cdot \vec{F} d\Omega}_{\textbf{Term 2}} + \underbrace{\int_{\partial\Omega_k} \phi_{k,i}^{+} \hat{F}(u^+, u^-, \vec{n}) dl}_{\textbf{Term 3}} = \mathbf{0}$$

1.3 Initial condition

$$h^0(x,y) = 1.0 + 0.3e^{-50(x-1.5)^2 - 50(y-0.7)^2}, \quad \vec{v}^0(x,y) = \vec{0}$$

1.4 Mesh

city0.gri, city1.gri, city2.gri from previous project.

1.5 Boundary Condition

Wall boundary condition should be used for all boundary groups, boundary flux is just continuous flux dotted with the normal vector, computed with the boundary state (zero normal velocity). For DG solver, boundary condition only contributes to **Term 3**.

1.6 Time Stepping

RK4 will be used.

$$\Delta t = \frac{\min_i(\sqrt{A_i}\,\mathrm{CFL})}{\max_i(\sqrt{gh_i})}$$

where A_i denotes area of element i, h_i denotes maximum water heigh in element i. Throughout this project, I am using CFL = 0.8

1.7 Output Calculation

We are interested in the forces on the building. In the shallow water equations, the force exerted on a building is

$$\vec{F} = F_x\hat{\mathbf{x}} + F_y\hat{\mathbf{y}} = \int_{\text{building}} \frac{1}{2}\rho g h^2 \vec{n}\, dl$$

but that's not the formula we use in code, instead we use

$$\vec{F} = \sum_{be=1}^{N_{be}} \sum_{q=1}^{nq} \frac{1}{2}\rho g h_{be,q}^2 w_q^{1d} \vec{n}_{be} \Delta l_{be}$$

in which N_{be} is the number of boundary edges of that building, nq is the number of 1D quadrature points, w_q is the quadrature weight.

2 Tasks

2.1 DG solver and Free-stream test/preservation test

2.1.1 Codes

① **precompute.cpp**: given approximation order p, precomputes

- C: lagrange basis coefficients
- M_proto: mass matrix
- Phi: basis evaluated on 2d quadrature points
- dxPhi, dyPhi: basis gradient on 2d quadrature points
- Phi_edge_c(CC): basis evaluated on edge quadrature points, both clockwise(c) and counter-clockwise(CC) ordering are stored
- wq_2d, wq_1d: quadrature weights

② Element Jacobian inverse is precomputed in another file, the reason for that is element Jacobian inverse requires extra knowledge of mesh which is not an input to **precompute.cpp**. Notice that I didn't calculate J^{-1}, instead I calculated $\det(J)J^{-1}$ which appears in **Term 2**.

③ **Res.cpp**: given state vector **U**, calculate residual. It includes three parts

- loop over elements: calculate **Term 2**
- loop over interior edges: calculate contribution of interior edges to **Term 3**
- loop over boundary edges: calculate contribution of boundary edges to **Term 3**

④ **Force.cpp**: given state vector **U**, calculate forces on each building.

⑤ **Submesh.cpp**: given state vector **U** and original mesh, output submesh for plotting.

2.1.2 Freestream test/ preservation test

I have set up two types of tests

- Full state boudnary: $h = 1, (u, v) =$ some nonzero constants.
- Wall boundary: $h = 1, (u, v) = 0$,
- p = 0, dt = 0.00336806
- p = 1, dt = 0.00112269
- p = 2, dt = 0.000673611

The results are shown in Figure [1] - [6]. We can see that residuals are near machine-precision, that means our solver has successfully passed free-stream test/ preservation test.

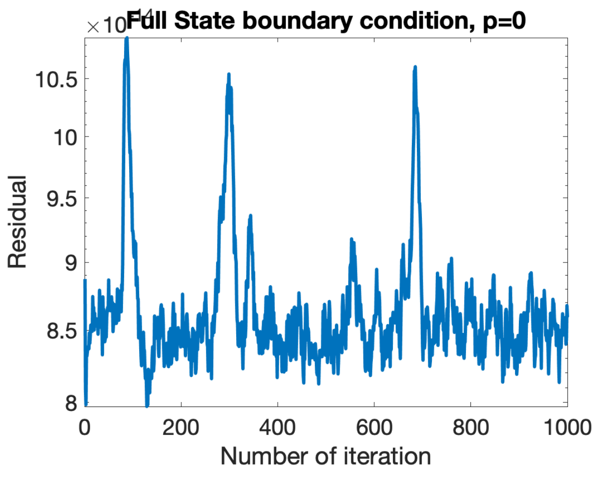

Figure 1:

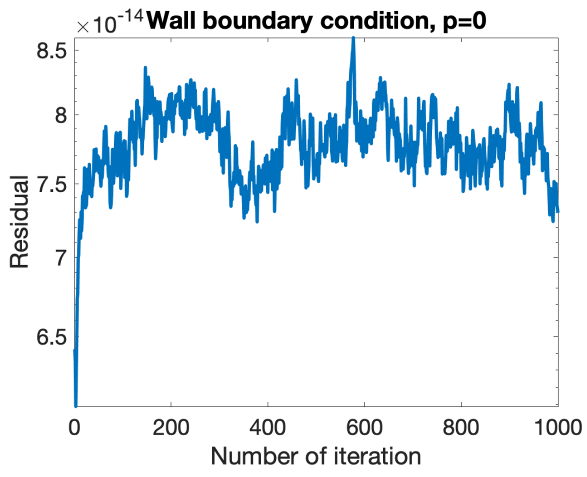

Figure 2:

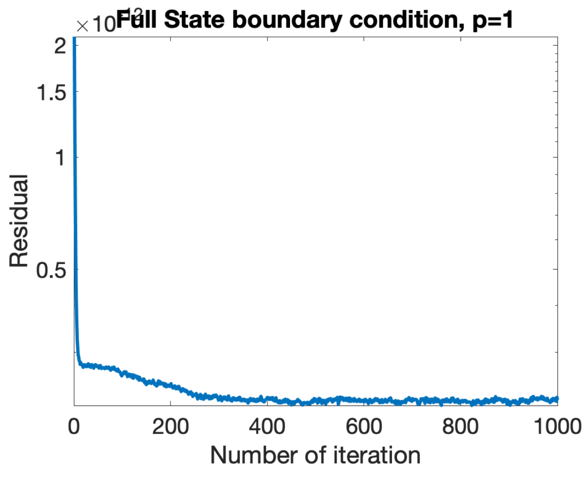

Figure 3:

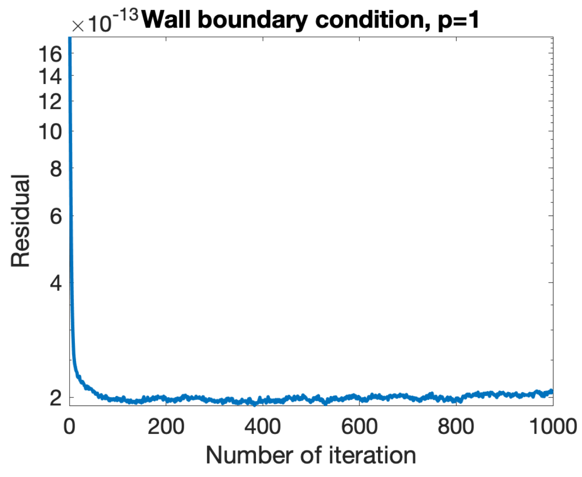

Figure 4:

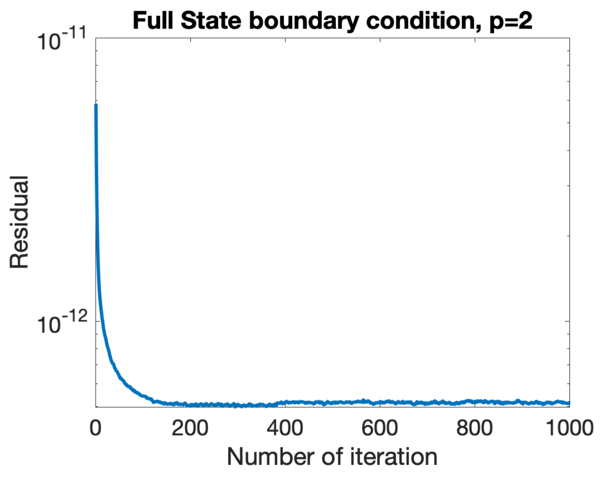

Figure 5:

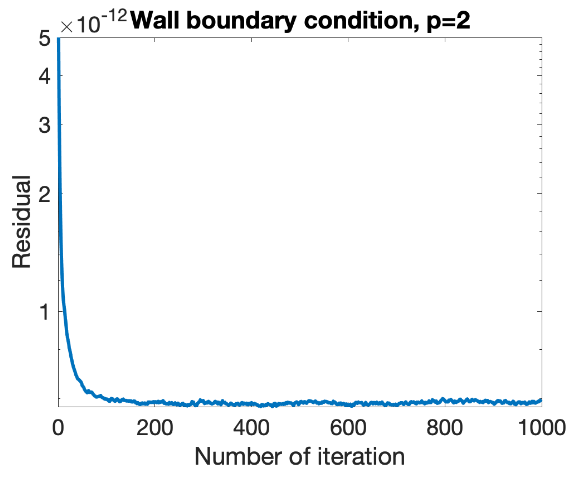

Figure 6:

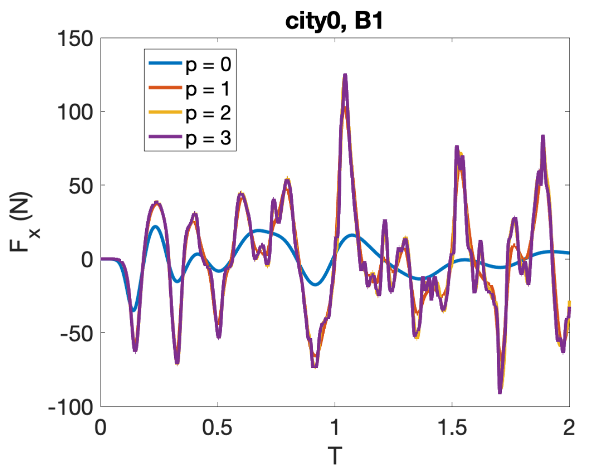

Figure 7:

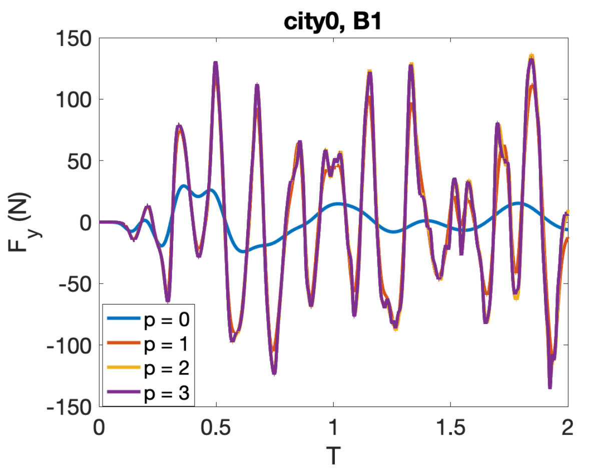

Figure 8:

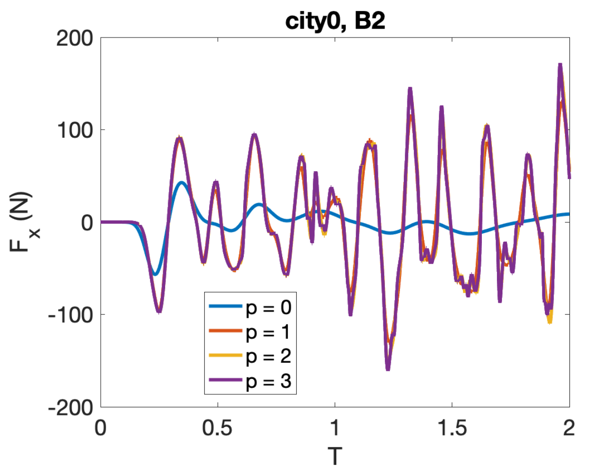

Figure 9:

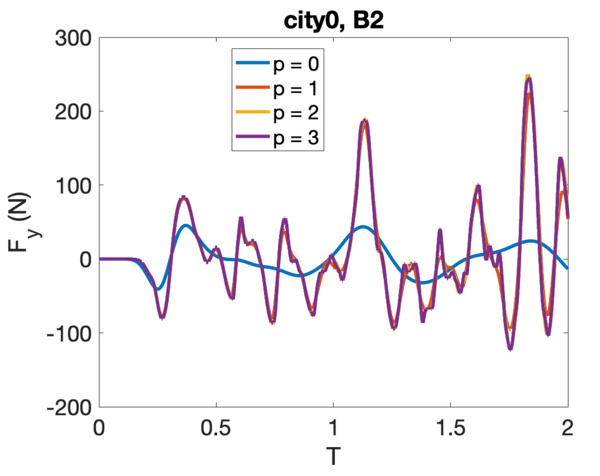

Figure 10:

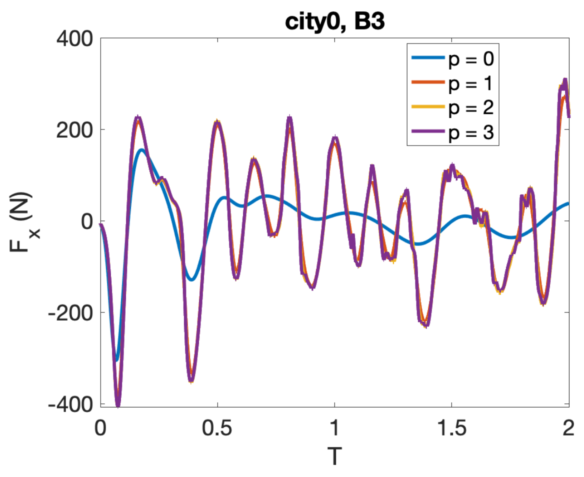

Figure 11:

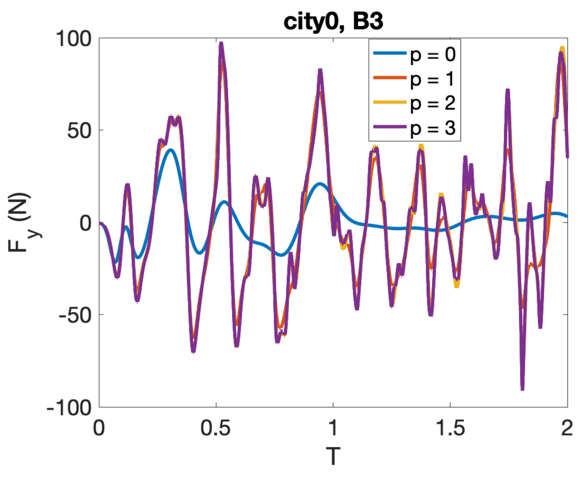

Figure 12:

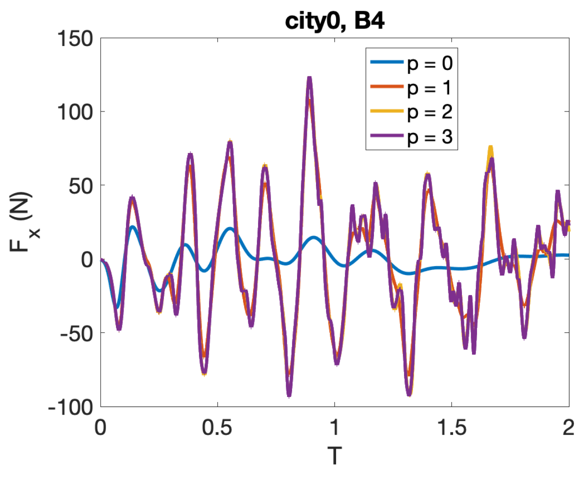

Figure 13:

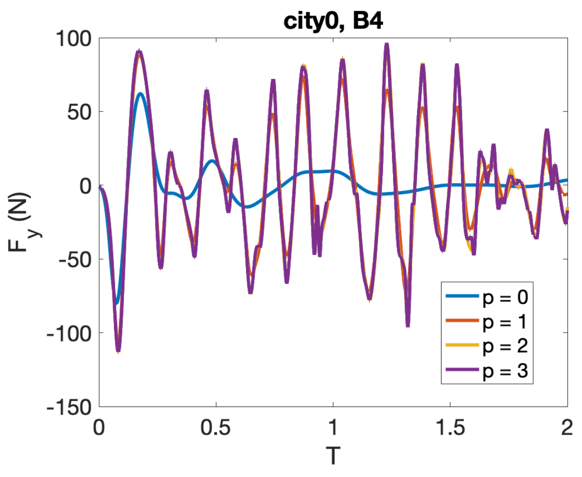

Figure 14:

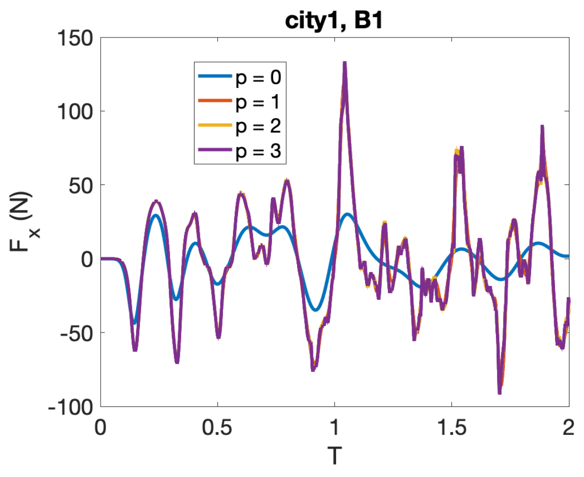

Figure 15:

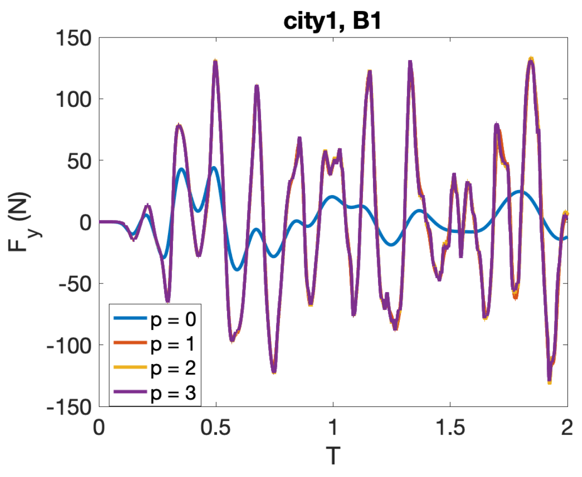

Figure 16:

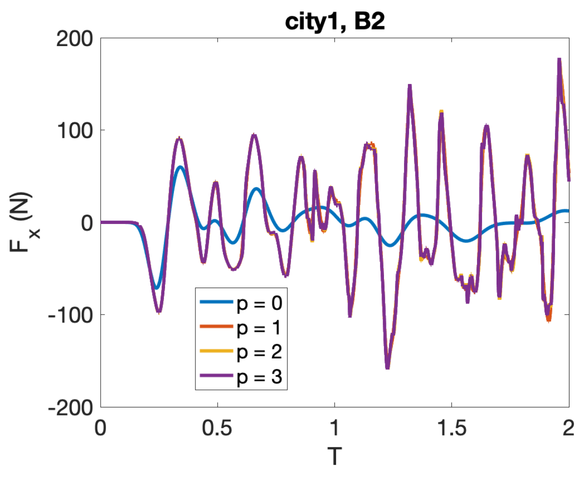

Figure 17:

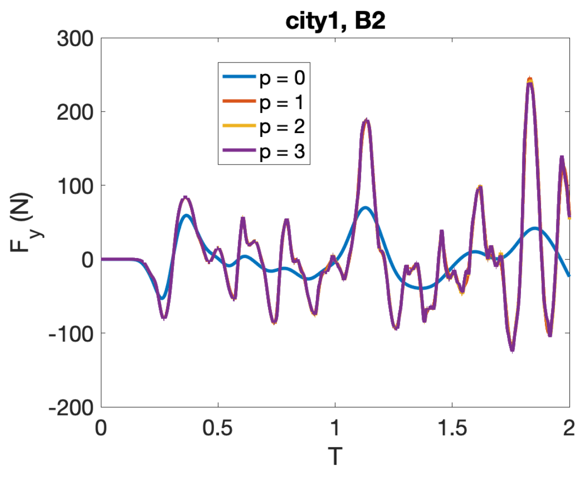

Figure 18:

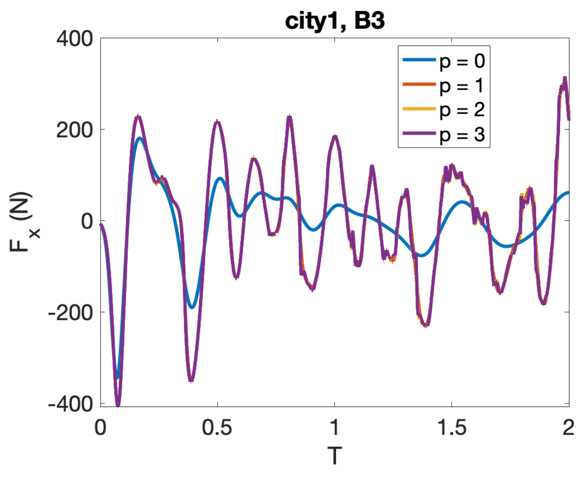

Figure 19:

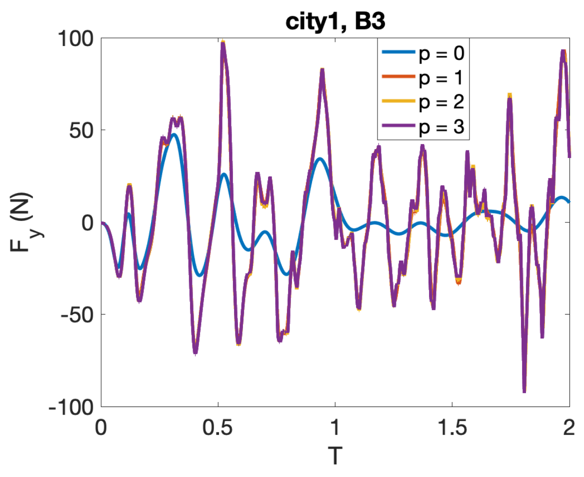

Figure 20:

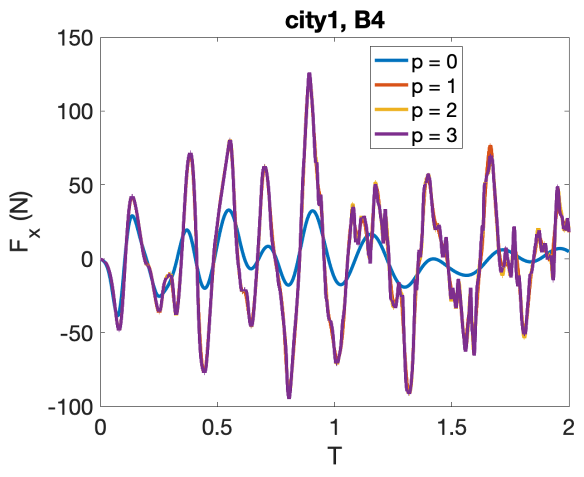

Figure 21:

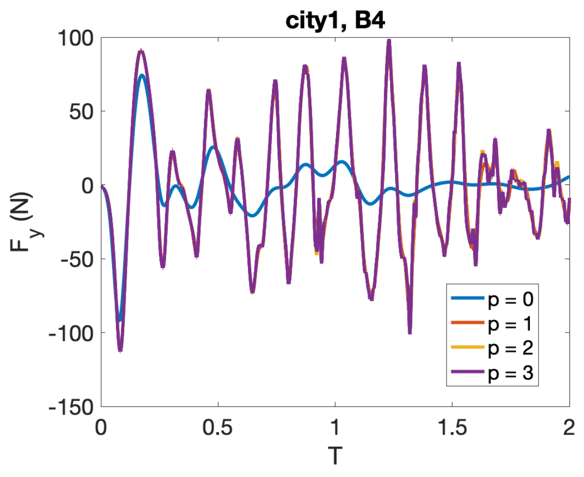

Figure 22:

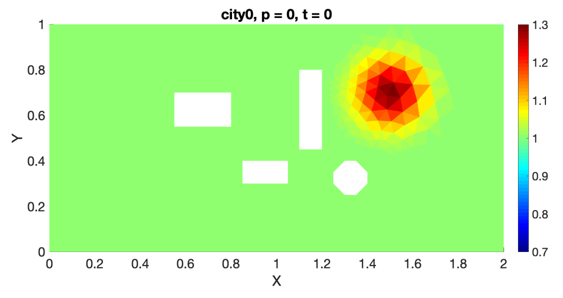

Figure 23:

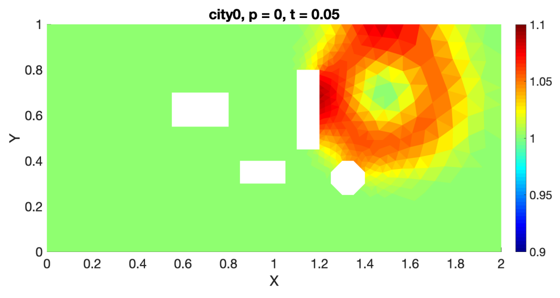

Figure 24:

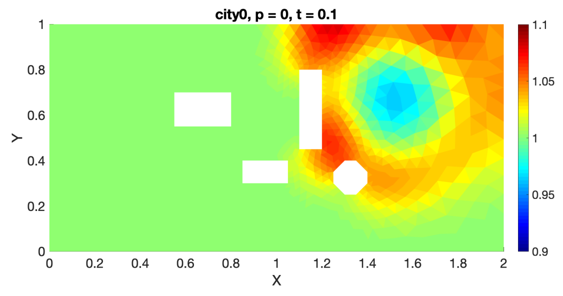

Figure 25:

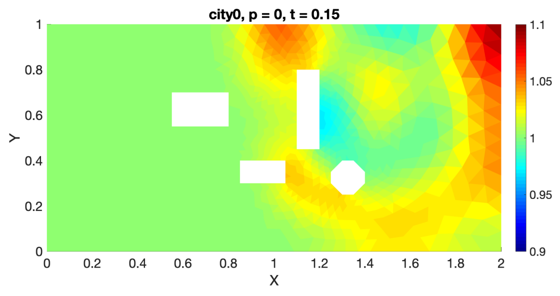

Figure 26:

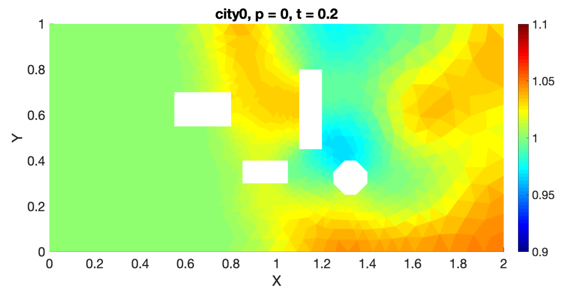

Figure 27:

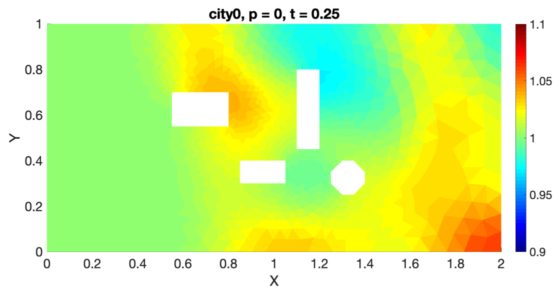

Figure 28:

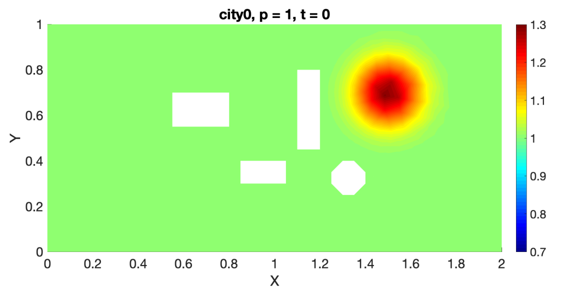

Figure 29:

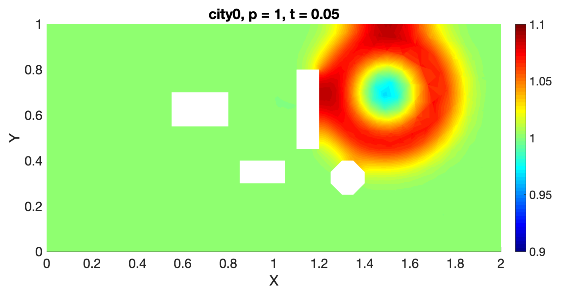

Figure 30:

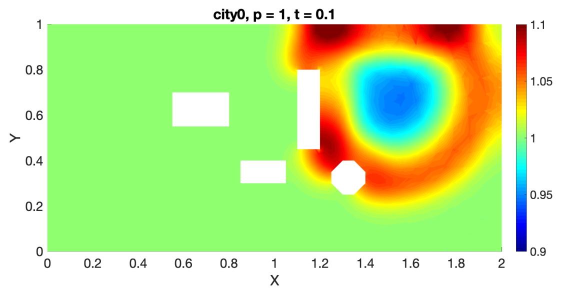

Figure 31:

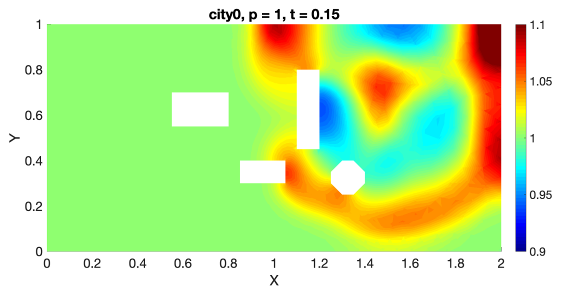

Figure 32:

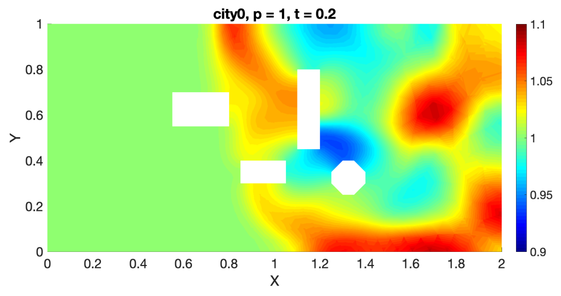

Figure 33:

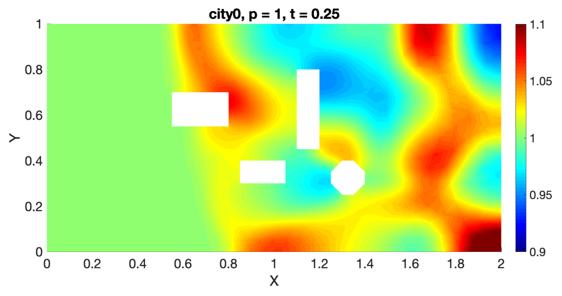

Figure 34:

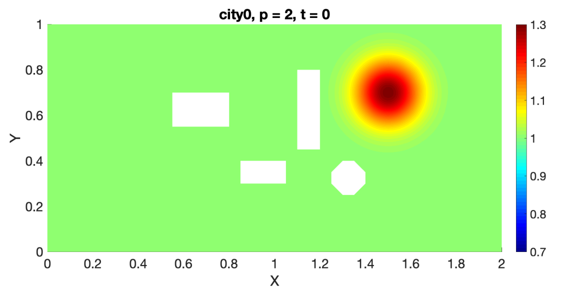

Figure 35:

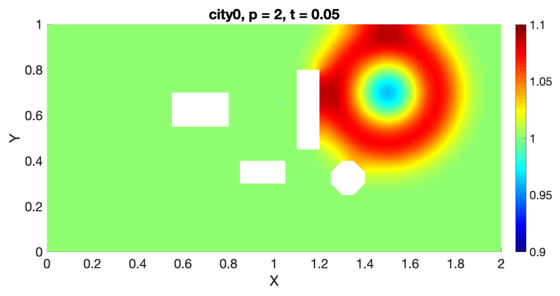

Figure 36:

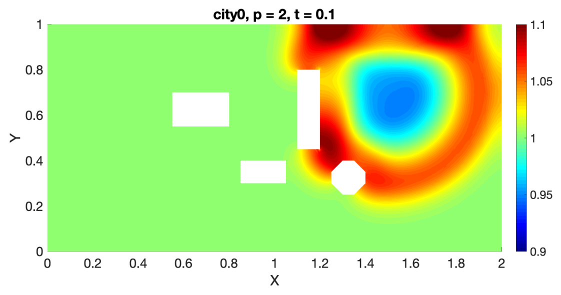

Figure 37:

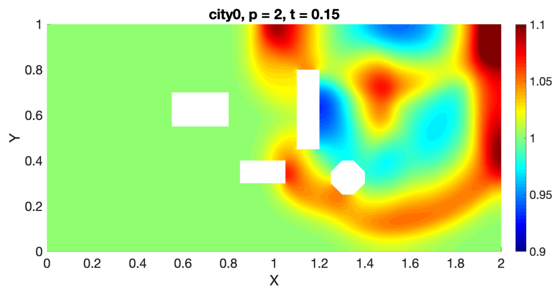

Figure 38:

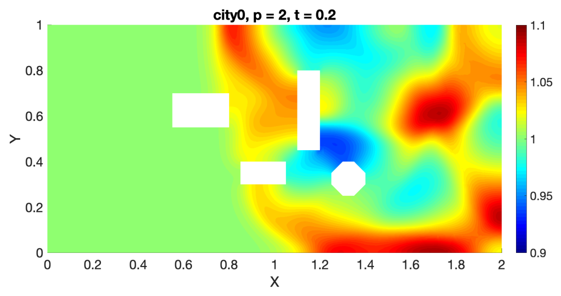

Figure 39:

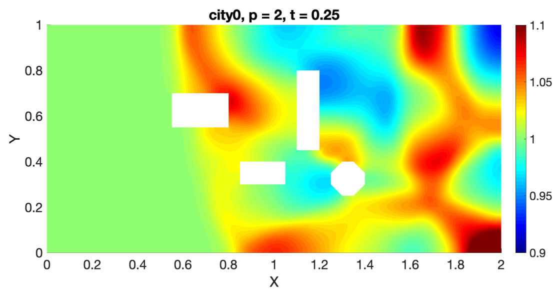

Figure 40:

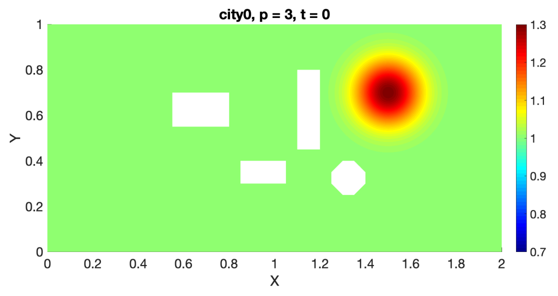

Figure 41:

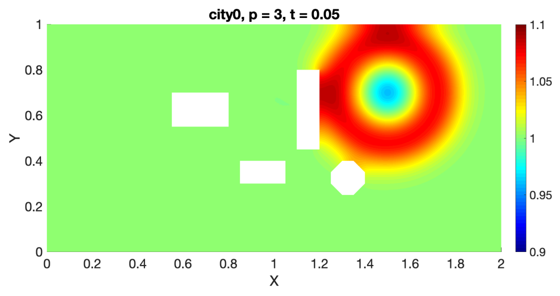

Figure 42:

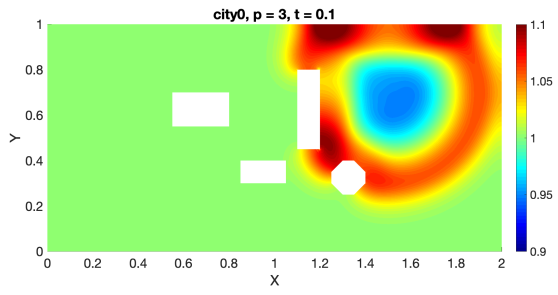

Figure 43:

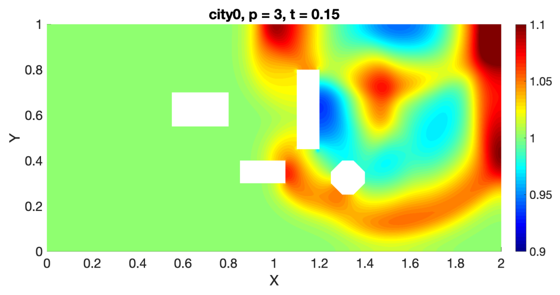

Figure 44:

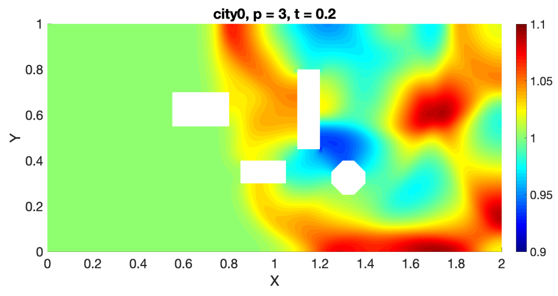

Figure 45:

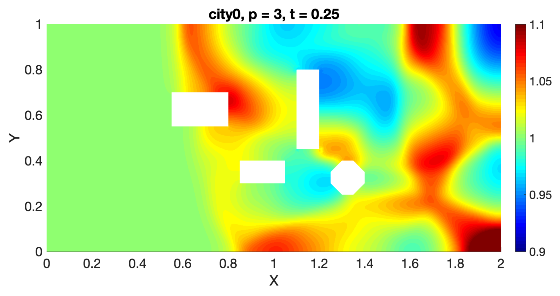

Figure 46:

2.2 Forces on buildings

The results are shown in Figure [7] - [22].

- High order solutions are roughly following the trend of finite volume solution (p=0), but they have much more fluctuations, this suggests that FV method significantly under-resolves the problem on coarse mesh.
- For this specific problem, we see that although results of p=1 and p=3 are very close to each other, p=3 does have more ’spikes’ than p=1, this phenomenon demonstrates that p=3 has larger(better) approximation space.

2.3 Water height contours

The results are shown in Figure [23] - [46].

- The solution fidelity improvement we gain going from FV to DG is substantial, DG with p=1 on coarsest mesh ’city0.gri’ already looks better than FV on finest mesh ’city2.gri’.
- Although the physics in DG and FV are same (SWE and Roe flux), DG allows us to capture physical phenomena which is missing in FV method due to under-resolving, e.g., when p=3 we can see vortexes start to developing around buildings’ sharp corners.

3 Appendix: Time step information from code output

```
Free-stream full state boundary condition test (p=0):
dt = 0.00336806
Free-stream full state boundary condition test (p=1):
dt = 0.00112269
Free-stream full state boundary condition test (p=2):
dt = 0.000673611
Free-stream wall boundary condition test (p=0):
dt = 0.00336806
Free-stream wall boundary condition test (p=1):
dt = 0.00112269
Free-stream wall boundary condition test (p=2):
dt = 0.000673611
city0.gri p=0 starts
dt = 0.00294118
city0.gri p=0 ends
city0.gri p=1 starts
dt = 0.000980392
city0.gri p=1 ends
city0.gri p=2 starts
dt = 0.000588235
city0.gri p=2 ends
city0.gri p=3 starts
dt = 0.000420168
city0.gri p=3 ends
city1.gri p=0 starts
dt = 0.00147059
city1.gri p=0 ends
city1.gri p=1 starts
dt = 0.000490196
city1.gri p=1 ends
city1.gri p=2 starts
dt = 0.000294118
city1.gri p=2 ends
city1.gri p=3 starts
dt = 0.00021097
city1.gri p=3 ends
```

4 Appendix: Debugging notes

For C++ Eigen library, suppose you want to reshape a MatrixXd object **tmp**, you can do
Map<MatrixXd> **b**(tmp.data(), rows, cols);
But an important thing to notice is that you can not do **inplace-transpose** on new matrix **b**, that is **b**.transposeInPlace() is not allowed, because the shape of **b** is somehow fixed.

My first instinct is that **b** is a dynamic-size matrix object, so its size should be flexible (changeable), but that's not true, the real type of **b** is Map<MatrixXd>, it is just a map onto **tmp**, also **b** itself does not require extra storage, that is **b** is a different view on **tmp**, changes in **b** will lead to changes in **tmp** (**b** is a reshaped 'reference' of **tmp**). For Class Map, transposeInPlace() is not provided, which makes sense, because I think changing the structure of **tmp** (**tmp**.data()) on memory by just operating on **b** is very dangerous.