HW2
'An Explicit In-Time Solver With Finite Difference Method For Linear Advection Problem'

Mehmet Şamil Dinçer (2236248)
Elvin Gültekinoğlu (2446169)

19/Nov/2023

1 Introduction

The purpose of this report is to demonstrate the implementation of explicit in-time solver by finite difference method. This implementation is based on C programming language and used to solve linear advection equation. Finite difference method is a numerical technique which approximates derivatives with finite differences to solve differential equations. It can be used to solve ordinary or partial differential equations by converting these into a group of linear equations. In other words, it transforms complex equations to easily solvable ones. In this project, to implement finite difference method to solve the aforementioned equation, a region in space is divided into sub-regions, which are nodes, so that a matrix containing all these with respective positions is created in a compact form. Then the solver algorithm is utilized and final results are presented on ParaView application.

2 The Theoretical Background

In this section, the methodology and its steps which are utilized to solve differential equations explicitly for linear advection equation are explained in detail.

2.1 Linear Advection Equation

Let us consider an advection equation, as shown in Equation 1:

$$\frac{\partial q}{\partial t} + \nabla(uq) = 0 \tag{1}$$

To gain a clearer understanding, let's examine the two-dimensional case represented by Equation 2:

$$\frac{\partial q}{\partial t} + \frac{\partial (uq)}{\partial x} + \frac{\partial (vq)}{\partial y} = 0 \tag{2}$$

In Dhawan, Kapoor and Kumar [2], this equation can be solved by using finite difference method. In order to solve this equation numerically in such manner, particularly with respect to time and position, it is crucial to determine the mesh system. An example of a mesh system is in figure1.

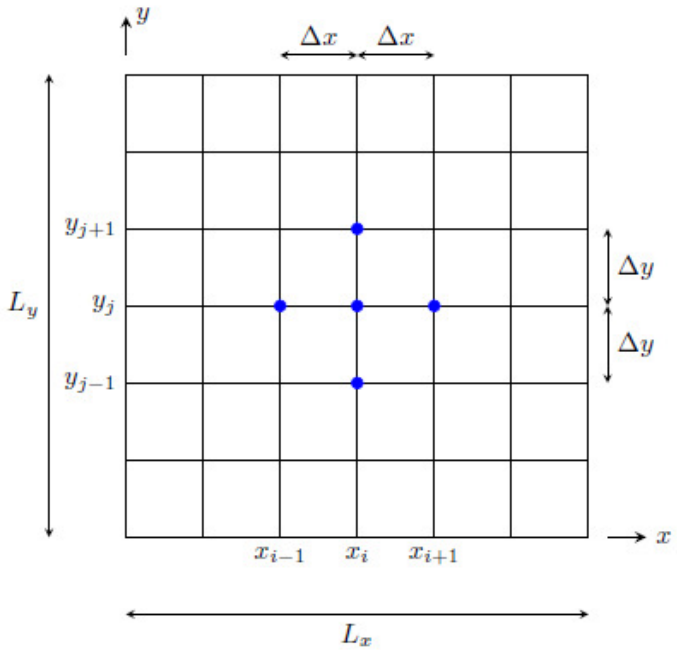


Figure 1: Finite Difference Mesh

After the defining mesh system, certain $\frac{\partial q}{\partial t}_{i,j}$ parameters are needed. There exists approximate solutions for $\frac{\partial (uq)}{\partial x}_{i,j}$ and $\frac{\partial (vq)}{\partial x}_{i,j}$, which are given by Equations 3 and 4 respectively:

$$\frac{\partial (uq)}{\partial x}_{i,j} \approx \begin{cases} \frac{u(i,j)p(i,j) - u(i-1,j)q(i-1,j)}{x(i,j) - x(i-1,j)}, & \text{for } u(i,j) \geq 0 \\ \frac{u(i+1,j)p(i+1,j) - u(i,j)q(i,j)}{x(i+1,j) - x(i,j)}, & \text{for } u(i,j) < 0 \end{cases} \tag{3}$$

$$\frac{\partial (vq)}{\partial x}_{i,j} \approx \begin{cases} \frac{v(i,j)p(i,j) - v(i-1,j)q(i-1,j)}{y(i,j) - y(i-1,j)}, & \text{for } v(i,j) \geq 0 \\ \frac{v(i+1,j)p(i+1,j) - v(i,j)q(i,j)}{y(i+1,j) - y(i,j)}, & \text{for } v(i,j) < 0 \end{cases} \tag{4}$$

Once $\frac{\partial (vq)}{\partial x}_{i,j}$ and $\frac{\partial (vq)}{\partial x}_{i,j}$ are defined, $\frac{\partial q}{\partial t}_{i,j}$ can be calculated using Equation 5:

$$\frac{\partial q}{\partial t} = \text{rhs}(q) = \left(-\frac{\partial (uq)}{\partial x} + \frac{\partial (vq)}{\partial y} \right) \tag{5}$$

Subsequently, a low-storage fourth-order five-stage Runge-Kutta(Runge-Kutta 45) method can be employed to integrate the resulting ordinary differential equation 5 in time (Burman and Ern, 2012, [1]) The Runge-Kutta 45 method is a numerical integration technique commonly used for solving ordinary differential equations (ODEs). It is an extension of the classic fourth-order Runge-Kutta (RK4) method. In the RK45 method, the solution to the ODE is advanced through a series of steps using a weighted average of function evaluations at different points within each step. Unlike RK4, which uses a fixed step size, RK45 dynamically adjusts the step size to balance accuracy and computational efficiency.

Runge-Kutta 4th Order, 5-Stage formulation is below:

$$\begin{aligned} k_1 &= h \cdot f(t_n, y_n), \\ k_2 &= h \cdot f(t_n + c_2 h, y_n + a_{21} k_1), \\ k_3 &= h \cdot f(t_n + c_3 h, y_n + a_{31} k_1 + a_{32} k_2), \\ k_4 &= h \cdot f(t_n + c_4 h, y_n + a_{41} k_1 + a_{42} k_2 + a_{43} k_3), \\ k_5 &= h \cdot f(t_n + c_5 h, y_n + a_{51} k_1 + a_{52} k_2 + a_{53} k_3 + a_{54} k_4), \\ y_{n+1} &= y_n + b_1 k_1 + b_2 k_2 + b_3 k_3 + b_4 k_4 + b_5 k_5, \end{aligned} \tag{6}$$

The solver of linear advection code (rhsQ) is given below:

```
void RhsQ(solver_t *solver, tstep_t *tstep, int stage){
mesh_t *msh = &solver->msh;
int n; //variable of n to get rid of (i,j) representation
double dux, dvy, dqt, u, v;
for(int j=0; j<msh->NY; j++){
    for(int i=0; i<msh->NX; i++){
      n=j*msh->NX+i; //calculation of n to get rid of (i,j) representation

      //Calculating u(n) and v(n)
      if (solver->u[n] < 0){
      dux=(solver->q[msh->N2N[4 * n + 1]]*solver->u[msh->N2N[4 * n +
      ↪  1]]-solver->q[n]*solver->u[n])/(msh->x[1]-msh->x[0]);
      }
      else{
      dux=(solver->q[n]*solver->u[n]-solver->q[msh->N2N[4 * n +
      ↪  3]]*solver->u[msh->N2N[4 * n + 3]])/(msh->x[1]-msh->x[0]);
      }
      if (solver->u[n + msh->Nnodes] < 0){
      dvy=(solver->q[msh->N2N[4 * n]]*solver->u[msh->N2N[4 * n]+
      ↪  msh->Nnodes]-solver->q[n]*solver->u[n+
      ↪  msh->Nnodes])/(msh->y[msh->NX]-msh->y[0]);
      }
      else{
      dvy=(solver->q[n]*solver->u[n+ msh->Nnodes]-solver->q[msh->N2N[4 * n +
      ↪  2]]*solver->u[msh->N2N[4 * n + 2]+
      ↪  msh->Nnodes])/(msh->y[msh->NX]-msh->y[0]);
      }
      tstep->rhsq[n]=-(dux+dvy); // dq/dt is calculated.

      //Time integration in 2 steps
      //Step 1: Update residual*/
      //resq = rk4a(stage)* resq + dt*rhsq
      tstep->resq[n] = tstep->rk4a[stage] * tstep->resq[n] + tstep->dt *
      ↪  tstep->rhsq[n];
      //Step:2 Update solution and store
       //q = q + rk4b(stage)*resq
      solver->q[n] = solver->q[n] + tstep->rk4b[stage] * tstep->resq[n];

    }
  }
}
```

2.2 Initial Conditions

To solve the linear advection equation, initial condition and velocity fields have to be defined at the initial state. The initial values of q, which are transport quantity, are calculated with Equation 7 below:

$$q = \sqrt{(x - x_c)^2 + (y - y_c)^2} - r \tag{7}$$

The velocity of x direction is u and the velocity of y direction is v. They are calculated with Equations 8, 9 below:

$$u = sin(4\pi(x + 0.5))sin(4\pi(y + 0.5)) \tag{8}$$

$$v = cos(4\pi(x + 0.5))cos(4\pi(y + 0.5)) \tag{9}$$

The initial canditions of linear advection code(initialCondition) is below:

```
void initialCondition(solver_t *solver){
  mesh_t *msh = &(solver->msh);
  solver->q = (double *)malloc(msh->Nnodes*sizeof(double));
  solver->u = (double *)malloc(2*msh->Nnodes*sizeof(double));
  int n;
  double x_c=0.5;
  double y_c=0.75;
  double r=0.15;
  for(int j=0; j<msh->NY; j++){
    for(int i=0; i<msh->NX; i++){
     n=j*msh->NX+i; //calculation of n to get rid of (i,j) representation
     solver->q[n] = sqrt(pow(msh->x[n] - x_c, 2) + pow(msh->y[n] - y_c, 2)) -
     ↪  r;
     solver->u[n]=sin(4 * M_PI * (msh->x[n] + 0.5)) * sin(4 * M_PI * (msh->y[n]
     ↪  + 0.5) );
     solver->u[n+msh->Nnodes]=cos(4 * M_PI * (msh->x[n] + 0.5) ) * cos(4 * M_PI
     ↪  * (msh->y[n] + 0.5) );
    }
  }
}
```

2.3 Boundaries and Neighbors

The periodic boundary condition is applied for this equation. At the periodic boundary condition, the node (i; j) is connected to nodes (i+1; j), (i; j +1), (i - 1; j) (i; j - 1) from east, north, west and south directions. The periodic boundary conditions imply that the nodes for (i = 0; j) are connected to (i = NX - 1; j) for j = 0, 1, ..., NY - 1. And similarly, the nodes (j = 0; i) are connected to (j = NY - 1; i) for i = 0, 1, ..., NX - 1. However, for coding, there are 9 different mesh boundary states. The states are shown in Figure 2.

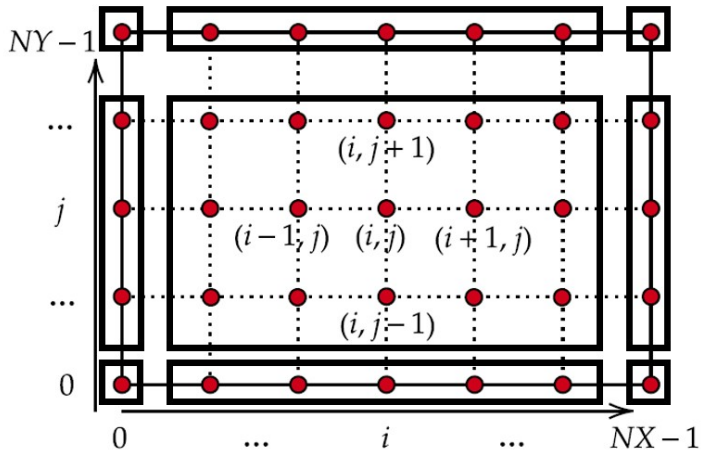


Figure 2: Finite Difference Method

So, to define the neighbors and boundaries, the node positions have to be checked.

For corner nodes, they don't have 2 neighbors. To define these neighbors, the rule explained before is used twice: once for the x-direction and second for the y-direction.

For edges nodes, they don't have 1 neighbor. To define these neighbors, the rule explained before is used once for missing directions.

For inside nodes, they have all neighbors.

To store neighbor positions of each node, linear indices n is used instead of double indices (i, j). The formulation of linear indices is given in Equation 10:

$$n = jXNX + i \tag{10}$$

To create mesh and store neighbor positions of each node code(createMesh) is below:

```
mesh_t createMesh(char* inputFile){
  mesh_t msh;
  // Read required fields i.e. NX, NY, XMIN, XMAX, YMIN, YMAX
  msh.NX   = readInputFile(inputFile, "NX");
  msh.NY   = readInputFile(inputFile, "NY");
  msh.xmin   = readInputFile(inputFile, "XMIN");
  msh.xmax   = readInputFile(inputFile, "XMAX");
  msh.ymin   = readInputFile(inputFile, "YMIN");
  msh.ymax   = readInputFile(inputFile, "YMAX");
  msh.Nnodes = msh.NX*msh.NY;
  msh.x = (double *) malloc(msh.Nnodes*sizeof(double));
  msh.y = (double *) malloc(msh.Nnodes*sizeof(double));
  //Compute Coordinates of the nodes
  double x_interval=(msh.xmax-msh.xmin)/(msh.NX-1); //horizontal interval
  ↪  between two nodes
  double y_interval=(msh.ymax-msh.ymin)/(msh.NY-1); //vertical interval between
  ↪  two nodes
  //double grid_array[msh.NX*msh.NY][2]; //creating array for calculating
  ↪  positions of nodes , x-->row , y--> column
  int n;
  for(int j=0; j<msh.NY; j++){
    for(int i=0; i<msh.NX; i++){
    n=j*msh.NX+i; //calculation of n to get rid of (i,j) representation
    msh.x[n]=i*x_interval; //x coordinates of nodes
    msh.y[n]=j*y_interval; //y coordinates of nodes
    } }
 //For every node 4 connections East north west and South-North that periodic
 ↪  connections required specific treatment
  msh.N2N = (int *)malloc(4*msh.Nnodes*sizeof(int)); // yukarı-->0, sağ-->1,
  ↪  aşağı-->2, sol-->3
  for(int j=0; j<msh.NY; j++){
    for(int i=0; i<msh.NX; i++){
    n=j*msh.NX+i;
      if(i==0 && j!=0 && j!=msh.NY-1){
        //left side
        msh.N2N[4*n]=n+msh.NX;
        msh.N2N[4*n+1]=n+1;
        msh.N2N[4*n+2]=n-msh.NX;
        msh.N2N[4*n+3]=n+msh.NX-1;
      }else if (j==0 && i!=0 && i!=msh.NX-1){
        //bottom side
        msh.N2N[4*n]=n+msh.NX;
        msh.N2N[4*n+1]=n+1;
        msh.N2N[4*n+2]=((msh.NY-1)*msh.NX)+i;
```

```
        msh.N2N[4*n+3]=n-1;
      }else if (i==msh.NX-1 && j!=0 && j!=msh.NY-1){
        //right side
        msh.N2N[4*n]=n+msh.NX;
        msh.N2N[4*n+1]=(n-msh.NX)+1;
        msh.N2N[4*n+2]=n-msh.NX;
        msh.N2N[4*n+3]=n-1;
      }else if (j==msh.NY-1 && i!=0 && i!=msh.NX-1){
        //upper side
        msh.N2N[4*n]=i;
        msh.N2N[4*n+1]=n+1;
        msh.N2N[4*n+2]=n-msh.NX;
        msh.N2N[4*n+3]=n-1;
      }else if(i==0&&j==0){
        //bottom left corner
        msh.N2N[4*n]=n+msh.NX;
        msh.N2N[4*n+1]=n+1;
        msh.N2N[4*n+2]=((msh.NY-1)*msh.NX);
        msh.N2N[4*n+3]=(n+(msh.NX))-1;
      }else if (i==msh.NX-1 && j==0){
        //bottom right corner
        msh.N2N[4*n]=n+msh.NX;
        msh.N2N[4*n+1]=(n-msh.NX)+1;
        msh.N2N[4*n+2]=((msh.NY-1)*msh.NX)+(msh.NX-1);
        msh.N2N[4*n+3]=n-1;
      }else if (j==msh.NY-1 && i==0){
        //top left corner
        msh.N2N[4*n]=0;
        msh.N2N[4*n+1]=n+1;
        msh.N2N[4*n+2]=n-msh.NX;
        msh.N2N[4*n+3]=(msh.NX*msh.NY)-1;
      }else if (j==msh.NY-1 && i==msh.NX-1){
        //top right corner
        msh.N2N[4*n]=i;
        msh.N2N[4*n+1]=(n-msh.NX)+1;
        msh.N2N[4*n+2]=n-msh.NX;
        msh.N2N[4*n+3]=n-1;
      } else {
        //inner
        msh.N2N[4*n]=n+msh.NX;
        msh.N2N[4*n+1]=n+1;
        msh.N2N[4*n+2]=n-(msh.NX);
        msh.N2N[4*n+3]=n-1;
      } } }
return msh;
}
```

3 Results

The analysis is conducted with 3 different time steps and 6 different mesh sizes. These analyses are presented in Table 1 below:

Table 1: List of Analysis

Time step	Mesh resolution
1e-4	20*20
	40*40
	80*80
	160*160
	320*320
	400*400
5e-5	20*20
	40*40
	80*80
	160*160
	320*320
	400*400
1e-5	20*20
	40*40
	80*80
	160*160
	320*320
	400*400

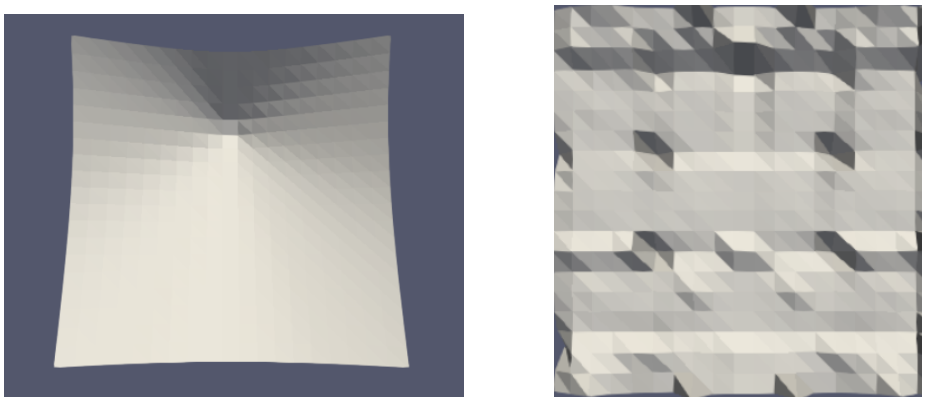
The procedure for selecting the time step and mesh resolution involves initially employing a larger time step and lower mesh resolution. Subsequently, the mesh resolution is incrementally increased while maintaining the same time step. Once all desired mesh resolutions have been evaluated, the time step is reduced, and the entire analysis is repeated.

The results are below for each time step and mesh resolution from Figure 3 to Figure 38.

3.1 Time step:1e-4

The chosen mesh resolutions are below:

3.1.1 Mesh 20*20



(a) Initial field (b) Final field

Figure 3: Time step is 1e-4

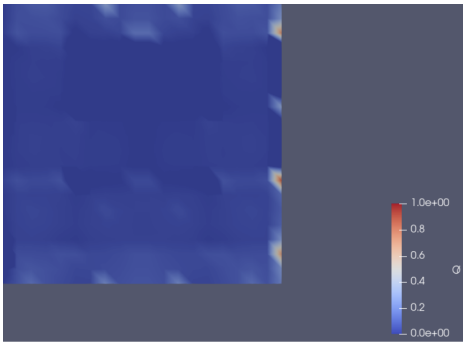
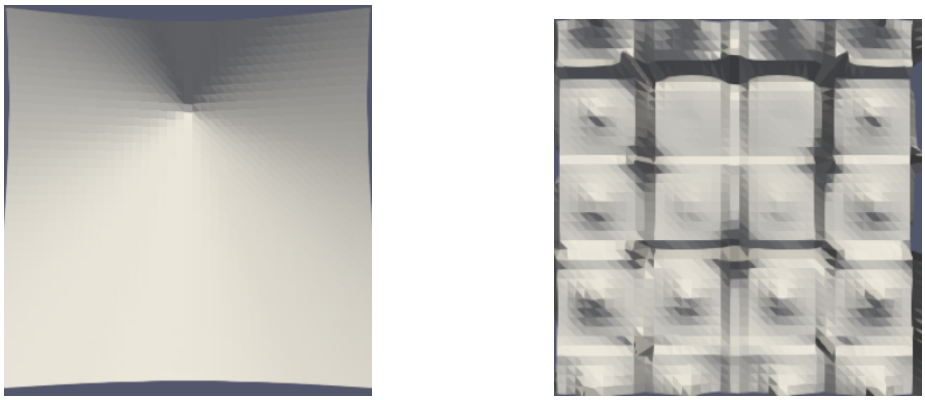


Figure 4: The contour final field for time step 1e-4

As seen from Figure 3 to Figure 4, resolution is not enough.

3.1.2 Mesh 40*40



(a) Initial field (b) Final field

Figure 5: Time step is 1e-4

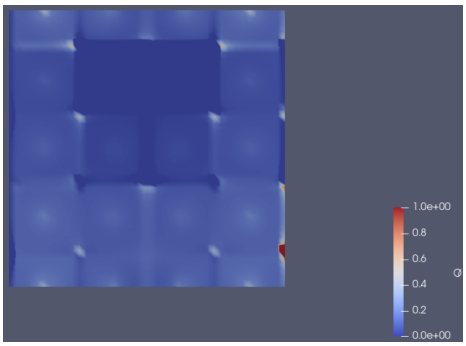
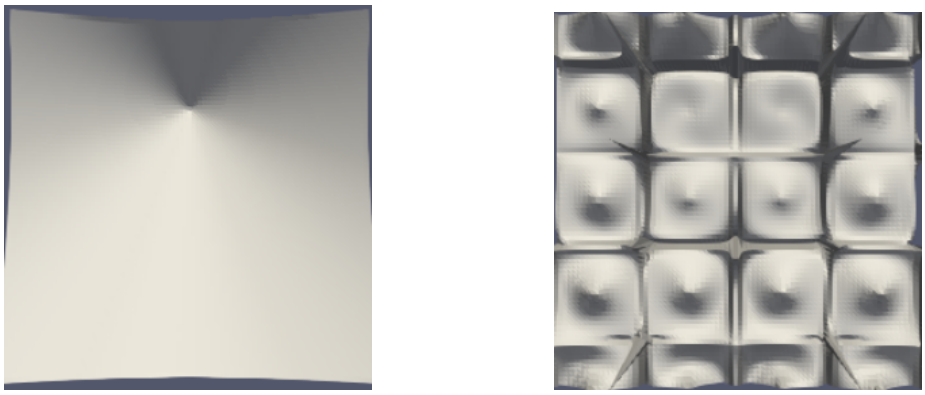


Figure 6: The contour final field for time step 1e-4

As seen from Figure 5 to Figure 6, resolution is not enough.

3.1.3 Mesh 80*80



(a) Initial field (b) Final field

Figure 7: Time step is 1e-4

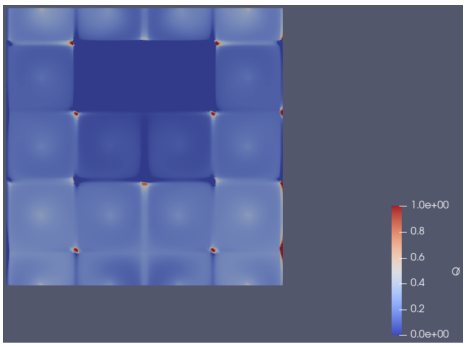
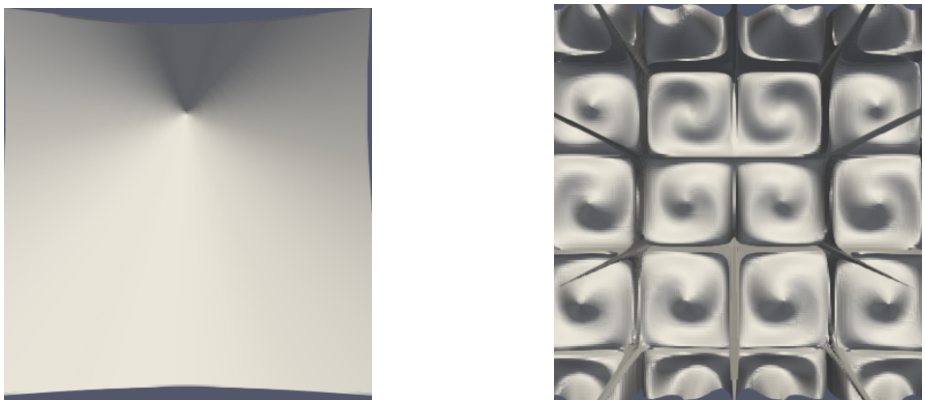


Figure 8: The contour final field for time step 1e-4

As seen from Figure 7 to Figure 8, resolution is better but still not enough.

3.1.4 Mesh 160*160



(a) Initial field (b) Final field

Figure 9: Time step is 1e-4

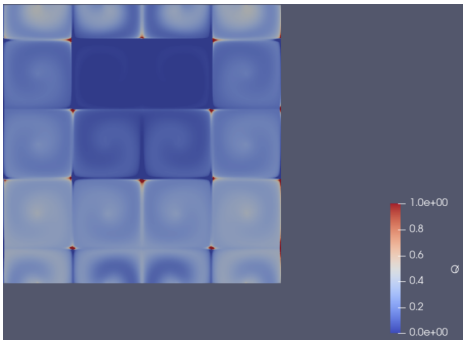
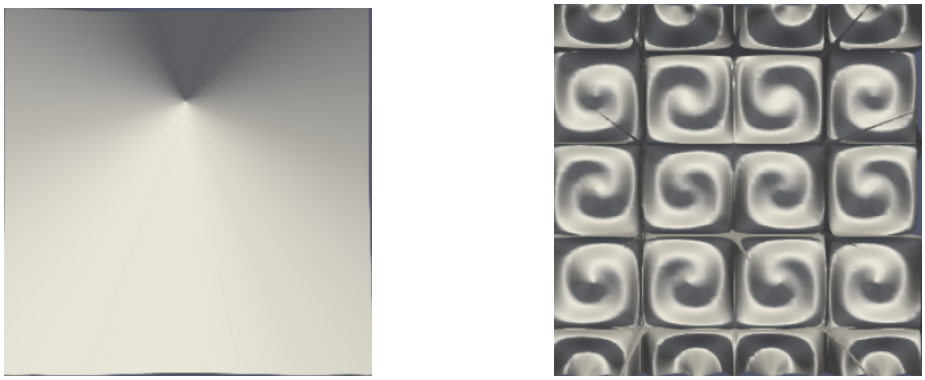


Figure 10: The contour final field for time step 1e-4

As seen from Figure 9 to Figure 10, resolution is better.

3.1.5 Mesh 320*320



(a) Initial field (b) Final field

Figure 11: Time step is 1e-4

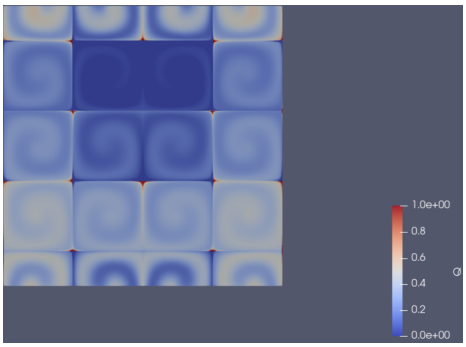
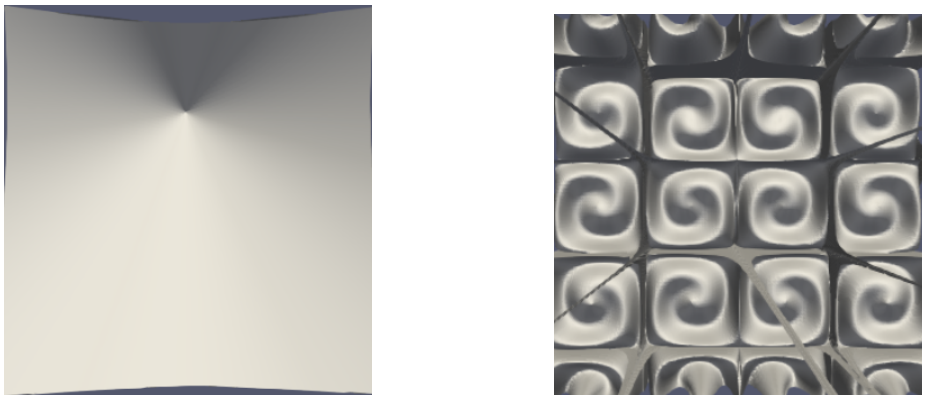


Figure 12: The contour final field for time step 1e-4

As seen from Figure 11 to Figure 12, resolution is good.

3.1.6 Mesh 400*400



(a) Initial field (b) Final field

Figure 13: Time step is 1e-4

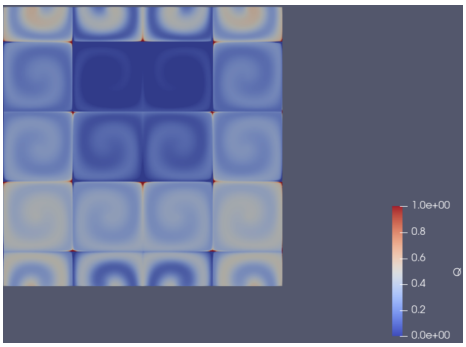


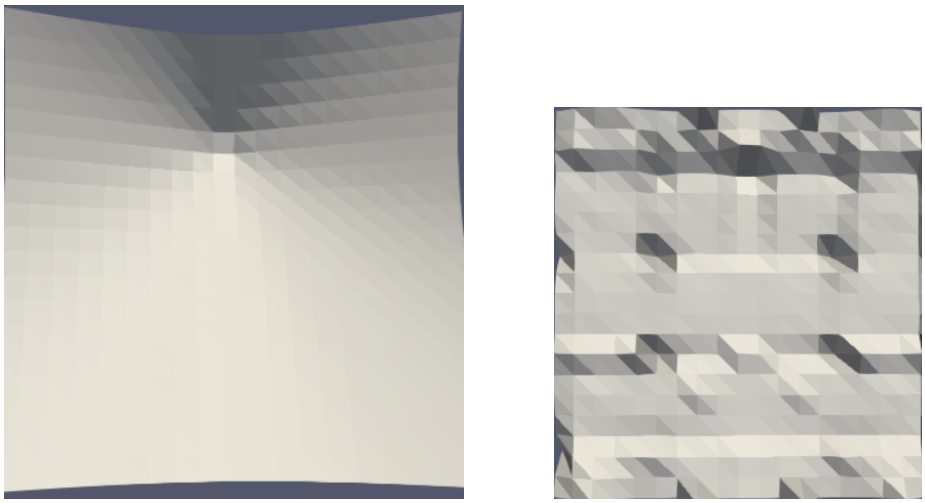
Figure 14: The contour final field for time step 1e-4

As seen from Figure 13 to Figure 14, resolution is good and adequate.

3.2 Time step:5e-5

The chosen mesh resolutions are below:

3.2.1 Mesh 20*20



(a) Initial field (b) Final field

Figure 15: Time step is 5e-5

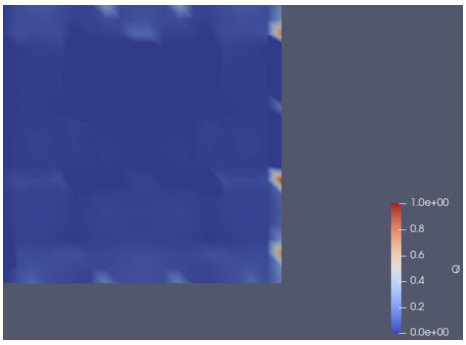
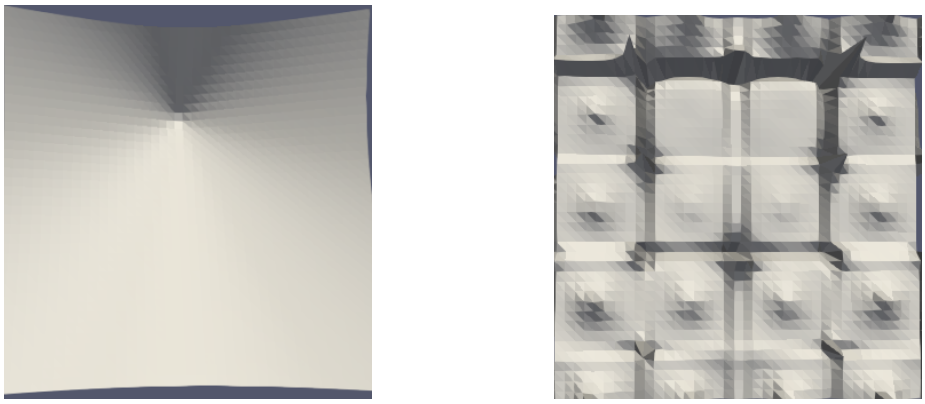


Figure 16: The contour final field for time step 5e-5

As seen from Figure 15 to Figure 16, resolution is not enough.

3.2.2 Mesh 40*40



(a) Initial field (b) Final field

Figure 17: Time step is 5e-5

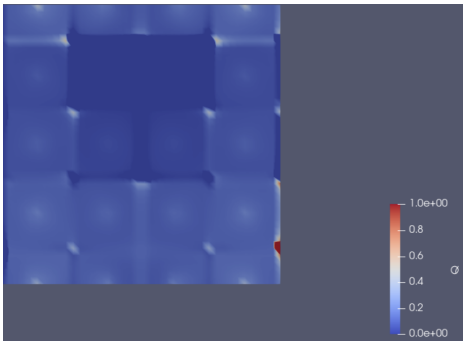
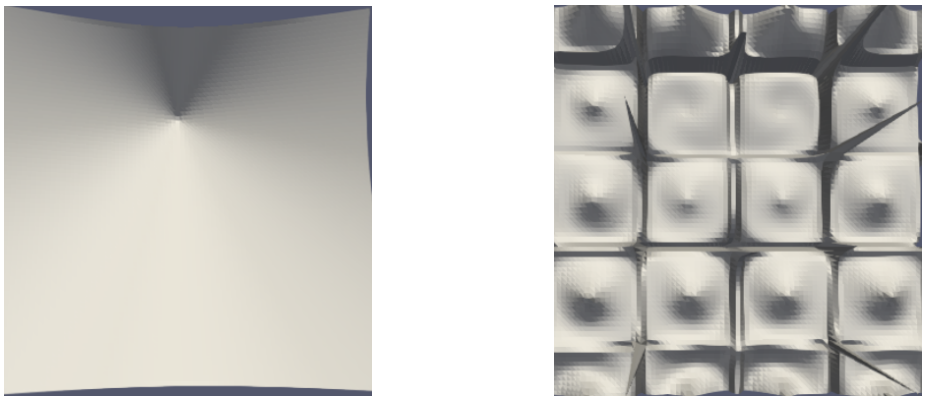


Figure 18: The contour final field for time step 1e-4

As seen from Figure 17 to Figure 18, resolution is not enough.

3.2.3 Mesh 80*80



(a) Initial field (b) Final field

Figure 19: Time step is 5e-5

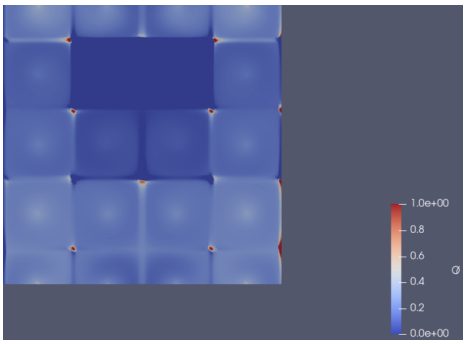
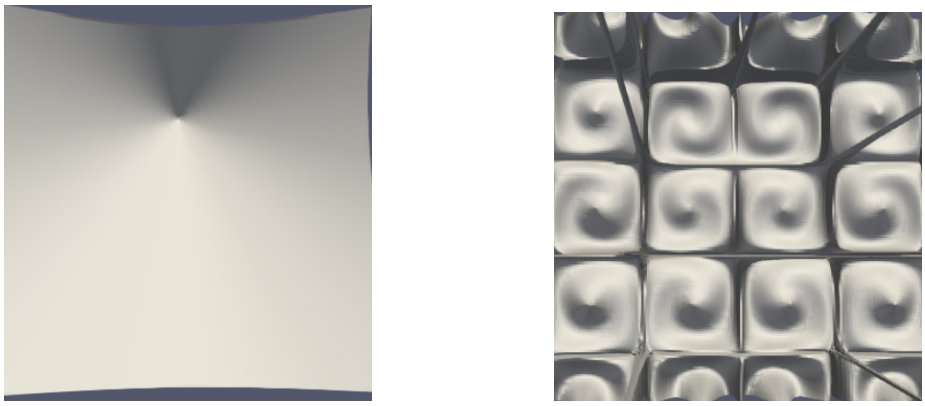


Figure 20: The contour final field for time step 5e-5

As seen from Figure 19 to Figure 20, resolution is not enough.

3.2.4 Mesh 160*160



(a) Initial field (b) Final field

Figure 21: Time step is 5e-5

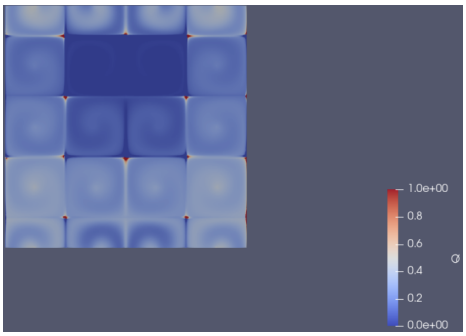
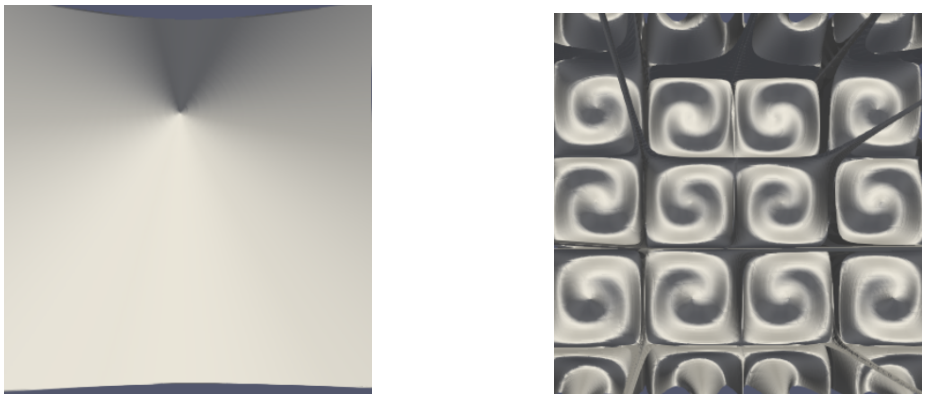


Figure 22: The contour final field for time step 5e-5

As seen from Figure 21 to Figure 22, resolution is better but there are still regions in which it is not adequate.

3.2.5 Mesh 320*320



(a) Initial field (b) Final field

Figure 23: Time step is 5e-5

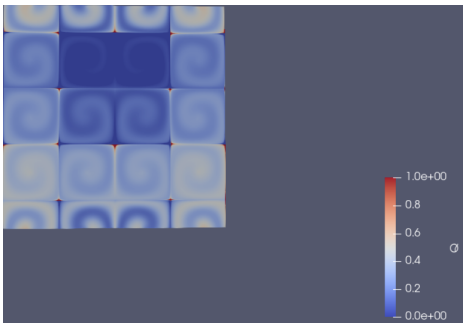
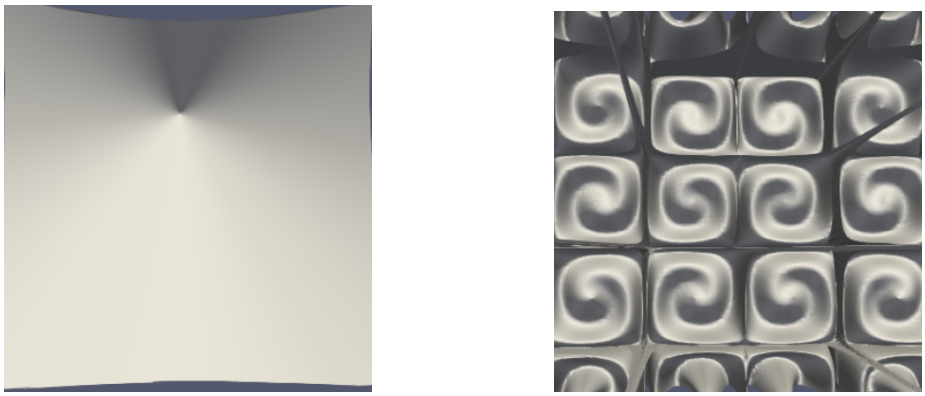


Figure 24: The contour final field for time step 5e-5

As seen from Figure 23 to Figure 24, resolution is good.

3.2.6 Mesh 400*400



(a) Initial field (b) Final field

Figure 25: Time step is 5e-5

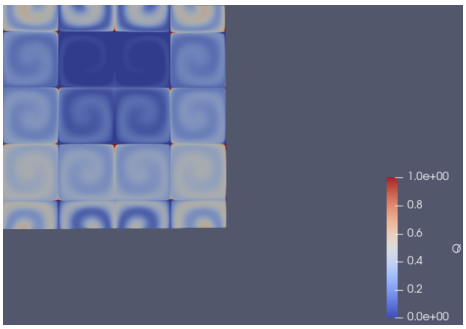


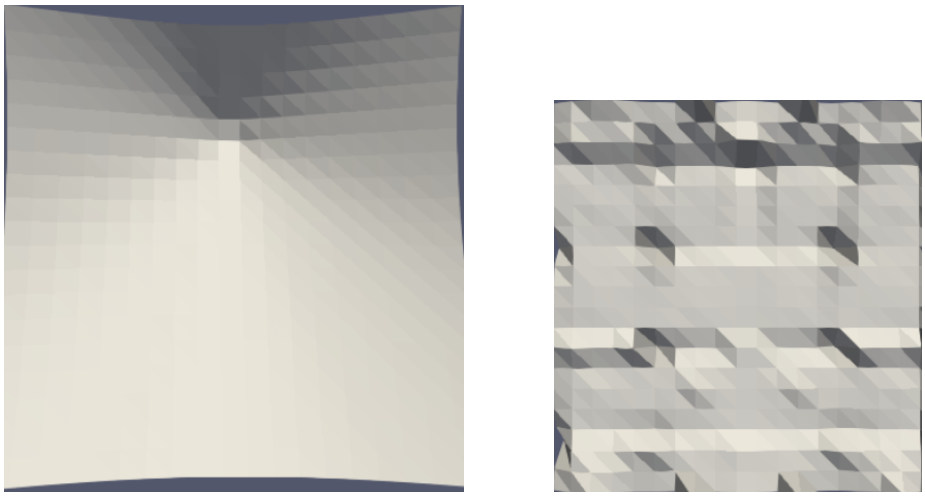
Figure 26: The contour final field for time step 5e-5

As seen from Figure 25 to Figure 26, resolution is very good.

3.3 Time step:1e-5

The chosen mesh resolutions are below:

3.3.1 Mesh 20*20



(a) Initial field (b) Final field

Figure 27: Time step is 1e-5

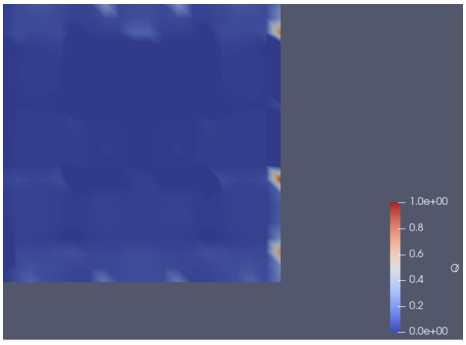


Figure 28: The contour final field for time step 1e-5

As seen from Figure 27 to Figure 28, resolution is not enough.

3.3.2 Mesh 40*40

(a) Initial field (b) Final field

Figure 29: Time step is 1e-5

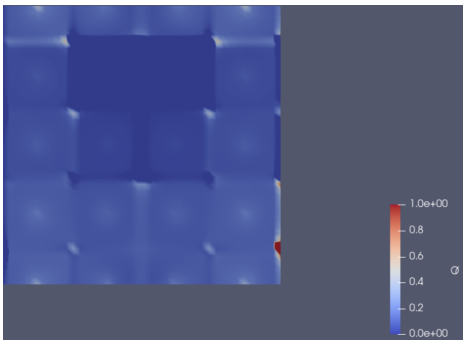
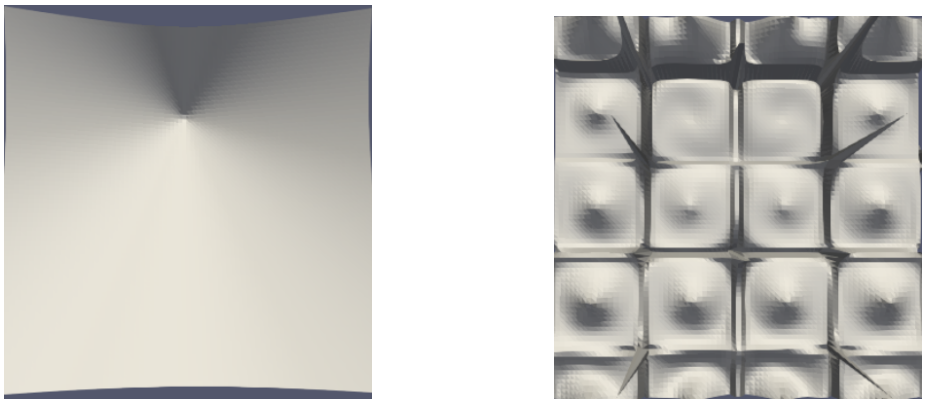


Figure 30: The contour final field for time step 1e-5

As seen from Figure 29 to Figure 30, resolution is not enough.

3.3.3 Mesh 80*80



(a) Initial field (b) Final field

Figure 31: Time step is 1e-5

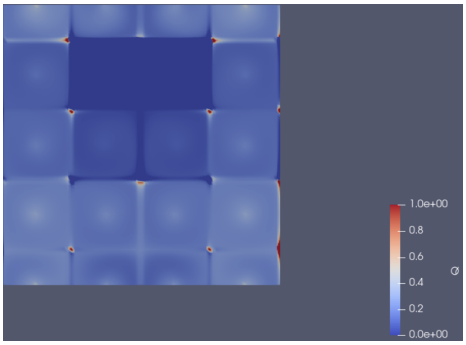
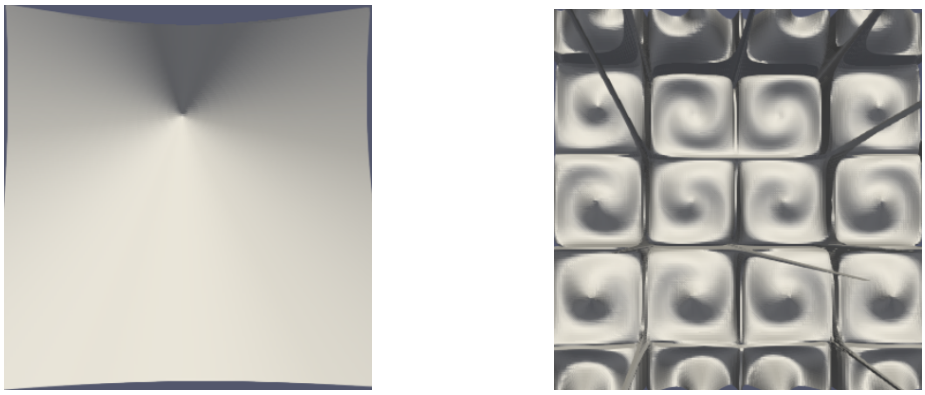


Figure 32: The contour final field for time step 1e-5

As seen from Figure 31 to Figure 32, resolution is not enough.

3.3.4 Mesh 160*160



(a) Initial field (b) Final field

Figure 33: Time step is 1e-5

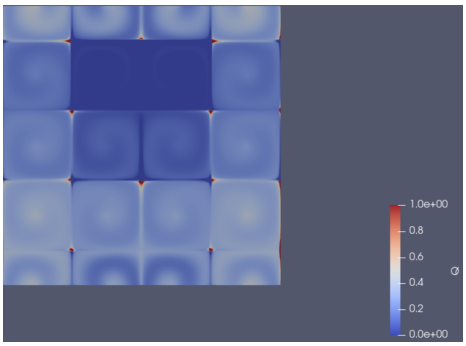
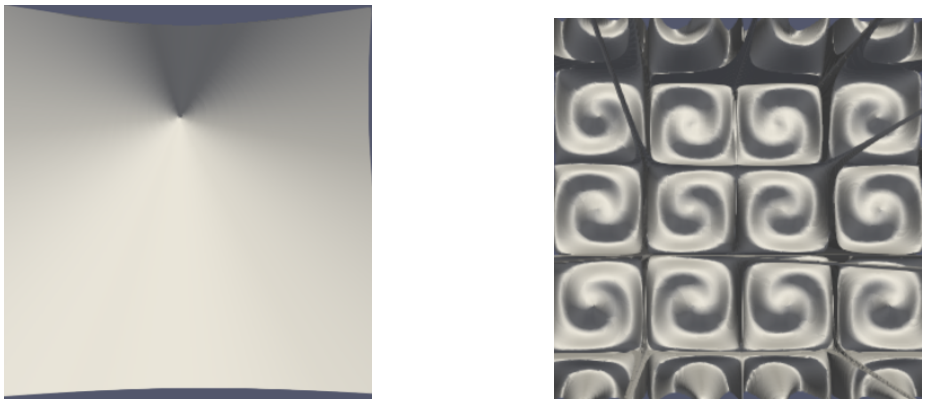


Figure 34: The contour final field for time step 1e-5

As seen from Figure 33 to Figure 34, resolution is better.

3.3.5 Mesh 320*320



(a) Initial field (b) Final field

Figure 35: Time step is 1e-5

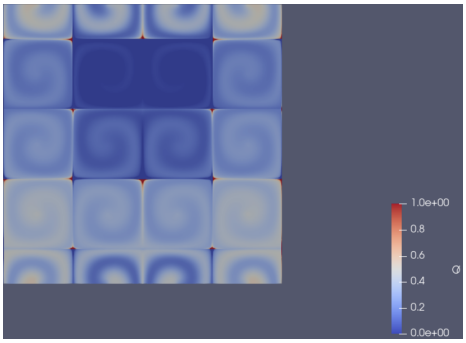
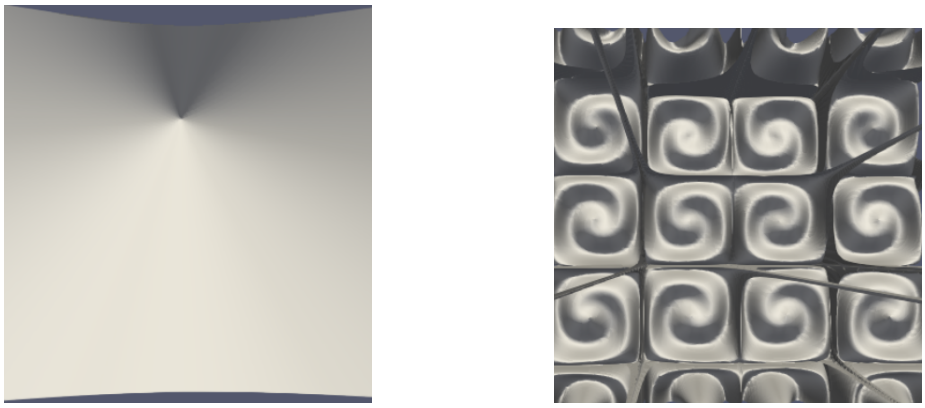


Figure 36: The contour final field for time step 1e-5

As seen from Figure 35 to Figure 36, resolution is good.

3.3.6 Mesh 400*400



(a) Initial field (b) Final field

Figure 37: Time step is 1e-5

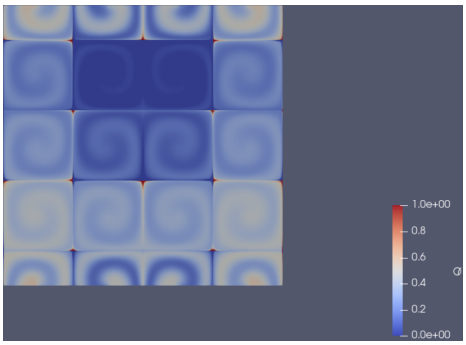


Figure 38: The contour final field for time step 1e-5

As seen from Figure 37 to Figure 38, resolution is very good.

4 Comments and Conclusion

The biggest effect of the solution is mesh resolution. The results indicate that mesh resolutions, like 20x20, 40x40, and 80x80, are not sufficient for the 2D linear advection solutions across all time steps. This insufficiency is attributed to the characteristics of the velocity fields. We use sinusoidal functions for velocity in both the x and y directions, the sinusoidal function undergoes four cycles, changing from 2 to 6. To accurately capture this type of function, a higher mesh resolution is necessary, typically around 1 element for every 2-3 degrees. 160*160 and higher resolution solutions support this idea. Although the 160*160 resolution is better, ı prefer 320*320 resolution for acceptable solutions.

After choosing the right mesh resolution, results are getting better with decreasing time steps. However, all the time steps are almost the same after choosing the right mesh resolution. This can be seen in Figure 12, 24, 36, 320*320 resolutions are good for all time steps.

References

[1] Erik Burman and Alexandre Ern. "Implicit-explicit Runge–Kutta schemes and finite elements with symmetric stabilization for advection-diffusion equations". In: *ESAIM: Mathematical Modelling and Numerical Analysis* 46.4 (2012), pp. 681–707.

[2] Sharanjeet Dhawan, S Kapoor, and Sheo Kumar. "Numerical method for advection diffusion equation using FEM and B-splines". In: *Journal of Computational Science* 3.5 (2012), pp. 429–437.