PROJECT 6: 2D SHOCK-VORTEX INTERACTION

MAE540

Adrian Zebrowski
April 24th, 2020

Problem Statement

The 2D MacCormack and Rusanov methods implemented previously are used to solve a shock-vortex interaction problem in a square domain with side length L. Neumann flux conditions are specified in the x direction and periodic boundary conditions are specified in the y direction. Grid resolutions of $\Delta x = \Delta y =$ L/25, L/50, and L/100 are considered. The shock wave is initialized using the density, pressure, and velocity conditions of the Sod problem, and the shock problem is first solved for a shock location $x_s = L/4$ and a time of t = (L/4)/S without the addition of the 2D isentropic vortex. This is done to ensure that the 2D MacCormack and Rusanov solver results match those obtained using the 1D MacCormack and Rusanov solvers in Project 4. These results are also compared to the exact solution for the Sod problem obtained using an exact Riemann solver. Once this validation step is complete, the 2D isentropic vortex is initialized at a location of $x_c = y_c = L/2$ using the Sod problem conditions to the right of the shock. The solvers are then run to a final time of t = (L/4)/S and t = (L/2)/S, corresponding to shock locations of $x_s = L/2$ and $x_s = 3L/4$. The results of both methods are presented in terms of vorticity contours for the fine grid solution ($\Delta x = \Delta y = L/100$) compared to the (analytical) initial condition, as well as vorticity profiles for the coarse, medium, and fine grid solutions at y = L/2 compared to the initial condition vorticity profile. The effect of the shock on the vortex structure and the grid dependency of the solutions is discussed. Time history of enstrophy, the dilatation vorticity source term, and the baroclinic vorticity source term is plotted at each grid resolution. Relative importance of these terms in the generation of enstrophy is discussed, and the time history plots are compared to the vorticity contours and profiles to determine if the results are consistent. Contour plots of these source terms are also created to show where these terms are being generated in the flow field.

Method of Solution

The 2D MacCormack method and Rusanov method are written as separate Python functions which can then be imported and called as needed. These functions are supplied with the parameters that define each case: grid resolution *N*, domain length *L*, final time *t_final*, CFL number *CFL*, velocity array *u*, velocity array *v*, specific heat ratio *gamma*, and gas constant *R*. The velocity inputs are initialized by using the vortex initialization code from the preceding Project 5, and are then supplied to the MacCormack and Rusanov solvers. The solvers then return velocity *u* and *v*, density *rho*, pressure *p*, temperature *T*, energy *e*, vorticity *omega*, the time vector *timevect*, dilatation source term *Sdil*, baroclinic source term *Stor*, dilatation source term array *dilatation_fin*, and baroclinic source term array *baroclinic_fin*.

Both of the functions are structured similarly, so only the *maccormack2Dvortex* function will be described in detail. The relevant parameters for each case are stored inside the function and selected with an input parameter *case*. Storage arrays for *u, v, T, x, y,* and *omega* are created. The initial timestep is defined and calculated. Spatial steps *dx* and *dy* are calculated from domain length and grid resolution. Velocity, temperature, and vorticity can then be calculated at each node using the corresponding analytical expressions. Immediately following the nested loop, pressure, density, and energy can be calculated using the ideal gas law. Storage vectors for *Q*, *F* and *G* are initialized and then populated using the initial condition primitive variables. Storage vectors for the predictor step for all of these variables are also created, but initialized as zero arrays. Speed of sound *a*, total energy *et*, and total enthalpy *H* are also calculated for convenience. The time variable *t* and a *count* variable are initialized as zero, and the initial timestep is calculated based on the CFL number, speed of sound, and largest magnitude velocity in the *x* and *y* direction.

The solution for all variables is calculated inside of a *while loop* that runs until the variable *t* (which is initially zero) reaches the final time *t_final*, with *dt* dynamically calculated at the end of the loop. Inside of this while loop, nested *for* loops are utilized to calculate the predictor and corrector values of Q, F, and G. Periodic boundary conditions are implemented in the y-direction by redirecting indices as needed, such that *j-1* with respect to the first node is the second to last node and *j+1* with respect to the last node is the second node. Neumann boundary conditions are implemented in the x-direction by redirecting the index *i-1* at the first node to the second node, and the index *i+1* at the last node to the second to last node. This results in a flux of zero across these first and last nodes.

The MacCormack method utilizes a predictor-corrector scheme, for which direction of differencing (forward or backward) must be alternated to ensure the solution is not biased. This is accomplished using the *count* variable and a series of *if* statements, which switch the differencing of the predictor and corrector based on Table 1 below. The Rusanov method is not a predictor-corrector scheme and does not contain forward or backward differencing that may bias the solution, so this step is not seen in the *rusanov2D* code. After the main time loop, dilatation and baroclinic source terms are calculated for the entire flow field without the evaluation of the integral – this is used for the contour plots of these terms. The integrated form of these terms is computed at each timestep (such that there is one value for the entire flow field at each timestep). The algorithm for the two-dimensional MacCormack method and two-dimensional Rusanov method are included below, along with definitions of the conservative variable vector *Q* and flux vectors *F* and *G*.

Vector definitions

$$F = \begin{bmatrix} \rho u \\ \rho u^2 + p \\ \rho uv \\ \rho uH \end{bmatrix} \tag{1a}$$

$$G = \begin{bmatrix} \rho v \\ \rho uv \\ \rho v^2 + p \\ \rho vH \end{bmatrix} \tag{1b}$$

$$Q = \begin{bmatrix} \rho \\ \rho u \\ \rho v \\ \rho e_t \end{bmatrix} \tag{2}$$

MacCormack 2D

$$Q_{i,j}^{\overline{n+1}} = Q_{i,j}^{n} - \Delta t \left[\frac{\left(F_{i+1,j}^{n} - F_{i,j}^{n}\right)}{\Delta x} + \frac{\left(G_{i,j+1}^{n} - FG_{i,j}^{n}\right)}{\Delta y} \right] \qquad Predictor \tag{3}$$

$$Q_{i,j}^{n+1} = \frac{1}{2}\left\{ Q_{i,j}^{n} + Q_{i,j}^{\overline{n+1}} - \Delta t \left[\frac{\left(F|_{i,j}^{\overline{n+1}} - F|_{i-1,j}^{\overline{n+1}} \right)}{\Delta x} + \frac{\left(G|_{i,j}^{\overline{n+1}} - G|_{i,j-1}^{\overline{n+1}} \right)}{\Delta y} \right] \right\} \qquad Corrector \qquad (4)$$

Table 1: Predictor-corrector forward/backward difference sequencing

Time step	Predictor		Corrector	
	x	y	x	y
1	Forward	Forward	Backward	Backward
2	Forward	Backward	Backward	Forward
3	Backward	Backward	Forward	Forward
4	Backward	Forward	Forward	Backward
5	Forward	Forward	Backward	Backward

Table 1: The forward/backward differencing used in the predictor-corrector scheme must be sequenced such that all possible permutations are used to avoid biasing the solution.

Rusanov 2D

$$Q_{i,j}^{n+1} = Q_{i,j}^{n} - \Delta t \left[\frac{\left(F_{i+\frac{1}{2}}^{n} - F_{i-\frac{1}{2}}^{n} \right)}{\Delta x} + \frac{\left(G_{i,j+\frac{1}{2}}^{n} - G_{i,j-\frac{1}{2}}^{n} \right)}{\Delta y} \right] \qquad (5)$$

$$F_{i+\frac{1}{2},j}^{n} = \frac{1}{2}\left\{ F_{i,j}^{n} + F_{i+1,j}^{n} - \max\left[|u|_{i,j} + a_{i,j}, |u|_{i+1,j} + a_{i+1,j} \right] \left(Q_{i+1,j}^{n} - Q_{i,j}^{n} \right) \right\} \qquad (6a)$$

$$G_{i,j+\frac{1}{2}}^{n} = \frac{1}{2}\left\{ G_{i,j}^{n} + G_{i,j+1}^{n} - \max\left[|v|_{i,j} + a_{i,j}, |v|_{i,j+1} + a_{i,j+1} \right] \left(Q_{i,j+1}^{n} - Q_{i,j}^{n} \right) \right\} \qquad (6b)$$

Equations (analytical solution/initialization)

$$r^{*} = \frac{\sqrt{(x - x_c)^2 + (y - y_c)^2}}{R_c} \qquad (7)$$

$$x^{*} = \frac{(x - x_c)}{R_c} \qquad (8a)$$

$$y^* = \frac{(y - y_c)}{R_c} \tag{8b}$$

$$u = u_\infty - M_{a,c} a_c x^* \exp\left[\frac{1 - (r^*)^2}{2}\right] \tag{9a}$$

$$v = v_\infty - M_{a,c} a_c y^* \exp\left[\frac{1 - (r^*)^2}{2}\right] \tag{9b}$$

$$T_c = \frac{T_\infty}{1 + \frac{\gamma - 1}{2} M_{a,c}^2} \tag{10}$$

$$T = T_\infty \left\{ 1 - \frac{\frac{\gamma - 1}{2} M_{a,c}^2}{1 + \frac{\gamma - 1}{2} M_{a,c}^2} (r^*)^2 \exp\left[\frac{1 - (r^*)^2}{2}\right] \right\} \tag{11}$$

$$\omega = \frac{M_{a,c} a_c}{R_c} \exp\left[\frac{1 - (r^*)^2}{2}\right] \{2 - (x^*)^2 - (y^*)^2\} \tag{12}$$

Results and Discussion

Part 1: Baseline shock cases

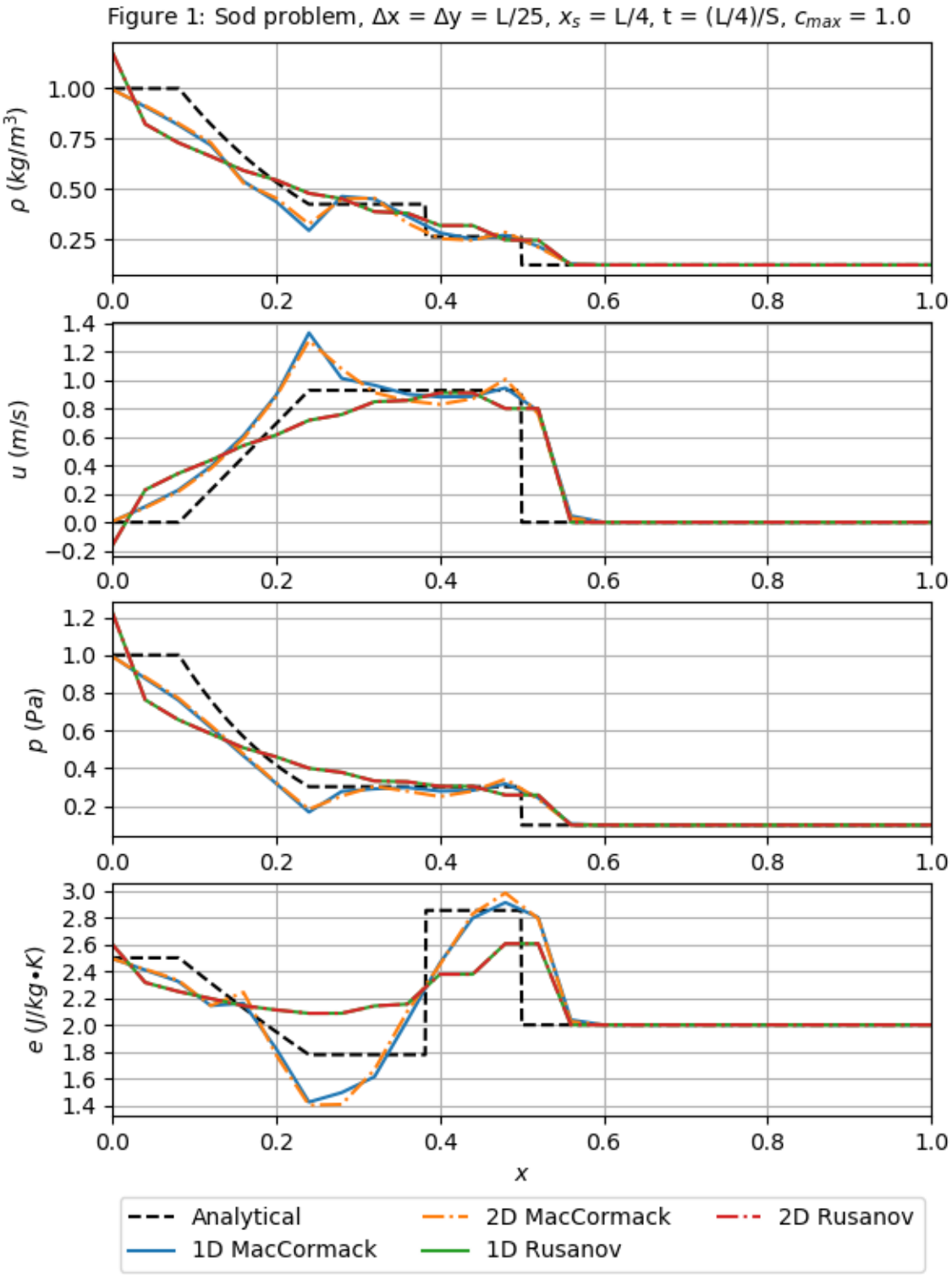

Figure 1: The solutions for the Sod problem provided by the 2D MacCormack and 2D Rusanov solvers are compared to the solutions provided by the previously implemented 1D MacCormack and 1D Rusanov solvers, with Δx = Δy = L/25.

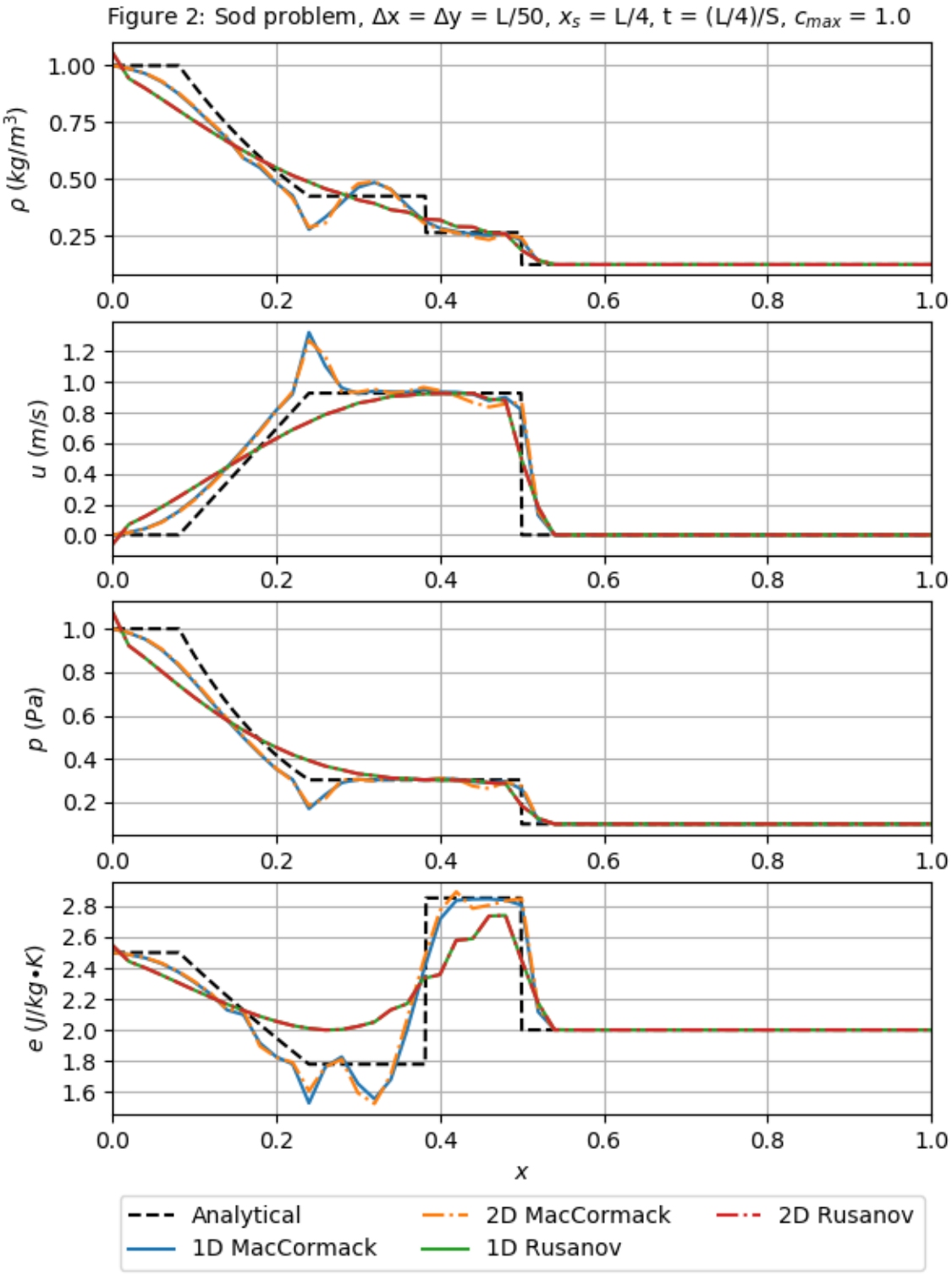

Figure 2: The solutions for the Sod problem provided by the 2D MacCormack and 2D Rusanov solvers are compared to the solutions provided by the previously implemented 1D MacCormack and 1D Rusanov solvers, with Δx = Δy = L/50.

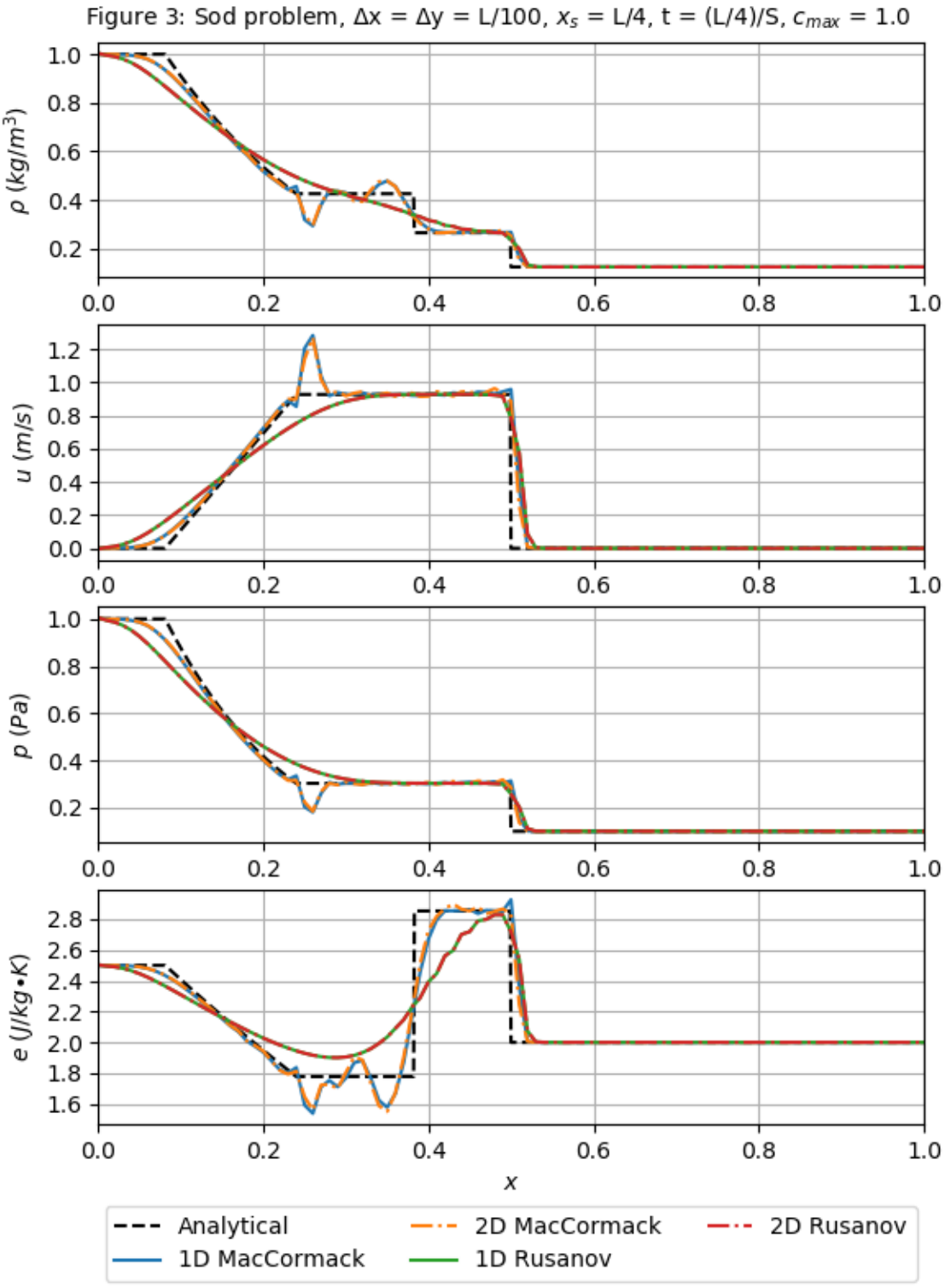

Figure 3: The solutions for the Sod problem provided by the 2D MacCormack and 2D Rusanov solvers are compared to the solutions provided by the previously implemented 1D MacCormack and 1D Rusanov solvers, with Δx = Δy = L/100.

The shock wave part of the problem (equivalent to the Sod problem tested in Project 4) without the addition of the isentropic vortex is solved using the 2D implementations of the MacCormack method and Rusanov method. The initial position of the shock is $x_s = L/4$, and the final time for the simulation is $t = (L/4)/S$ where S is the shock speed. Results are compared to the 1D implementations of the same methods used in Project 4 and to the exact solution obtained using a Riemann solver. Comparison plots of density, velocity, pressure, and energy are created at a coarse grid resolution (Fig. 1), medium grid resolution (Fig. 2), and fine grid resolution (Fig. 3) corresponding to $\Delta x = \Delta y = L/25$, L/50, and L/100 respectively.

The results of the 1D and 2D solvers match closely at all grid resolutions, with the Rusanov solvers appearing exactly equivalent and the MacCormack solvers differing only slightly. These small differences in the MacCormack method can be attributed to the forward-backward difference sequencing used to avoid biasing the solution, which differs in the 2D implementation. The 1D implementation alternates between a forward difference predictor with a backward difference corrector, and a backward difference predictor with a forward difference corrector. The 2D implementation is more complex, and cycles through all possible permutations of forward and backward differencing in both dimensions (see Table 2).

The MacCormack method solution is effective at resolving the shock front, but suffers from oscillations due to dispersive error. Overshoots and undershoots are visible following discontinuities, and only slightly decrease in magnitude with grid refinement. Overall solution accuracy is poor for the coarse grid and medium grid, with large departures from the analytical solution visible (especially to the left of the shock) in Fig. 1 and Fig. 2. The fine grid solution (Fig. 3) is significantly closer to the analytical solution apart from the oscillations mentioned previously.

The Rusanov method does not suffer from oscillations, but is highly dissipative and suffers from peak suppression and contact surface smearing. The coarse grid (Fig. 1) and medium grid (Fig. 2) solutions suffer most severely from this, and show a pronounced stair-step pattern in areas near discontinuities. This stair-step pattern is less pronounced for the finest grid solution (Fig. 3), which also suffers from the least dissipative error and resolves contact surfaces more clearly than the coarser grid solutions.

Part 2: Vortex addition

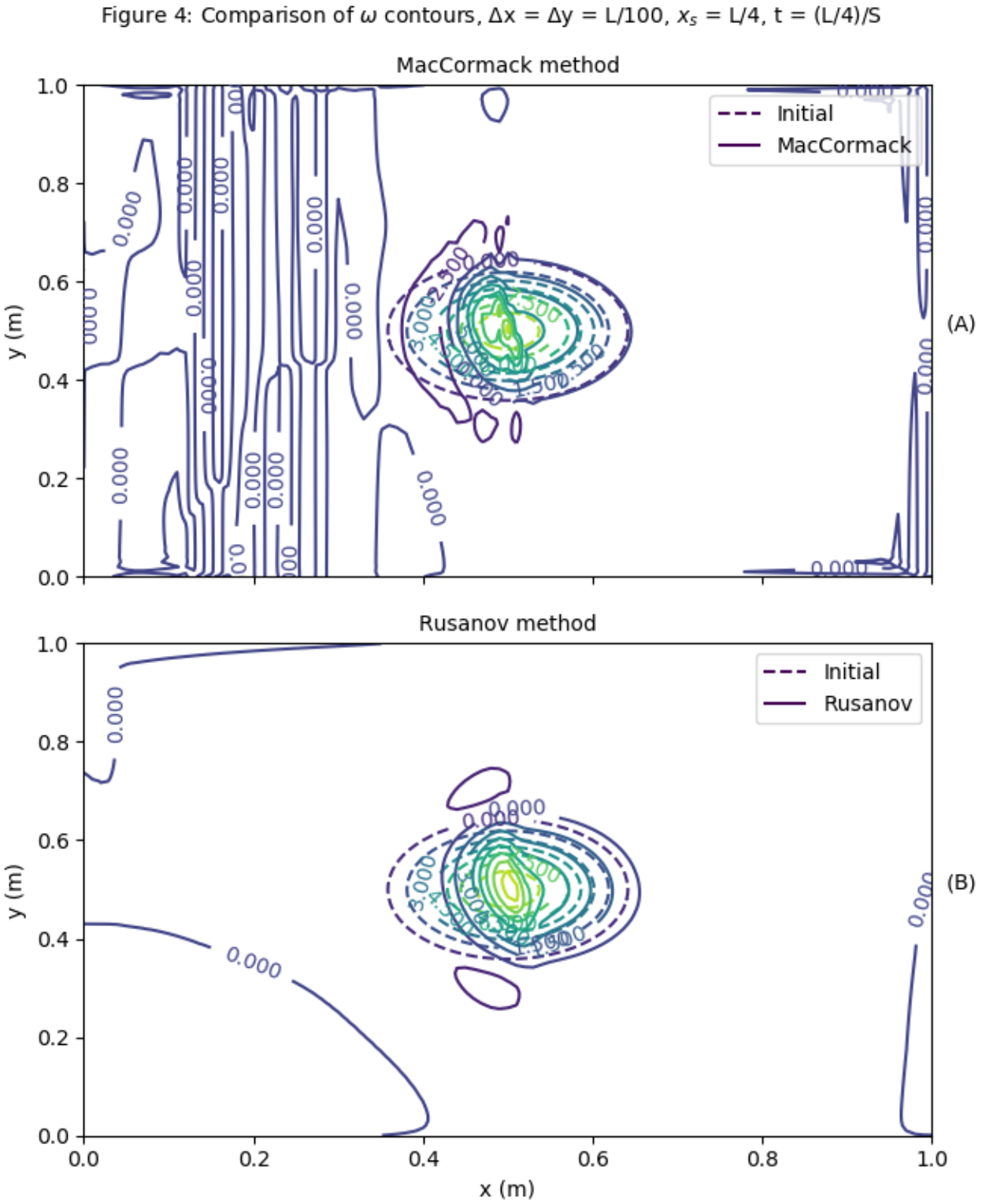

Figure 4: The MacCormack method and Rusanov method are used to model the interaction of a shock initially located at x_s = L/4 and an isentropic vortex initially located at x_c = L/2, y_c = L/2. Vorticity contours at a grid resolution of Δx = Δy = L/100 are shown for time t = (L/4)/S.

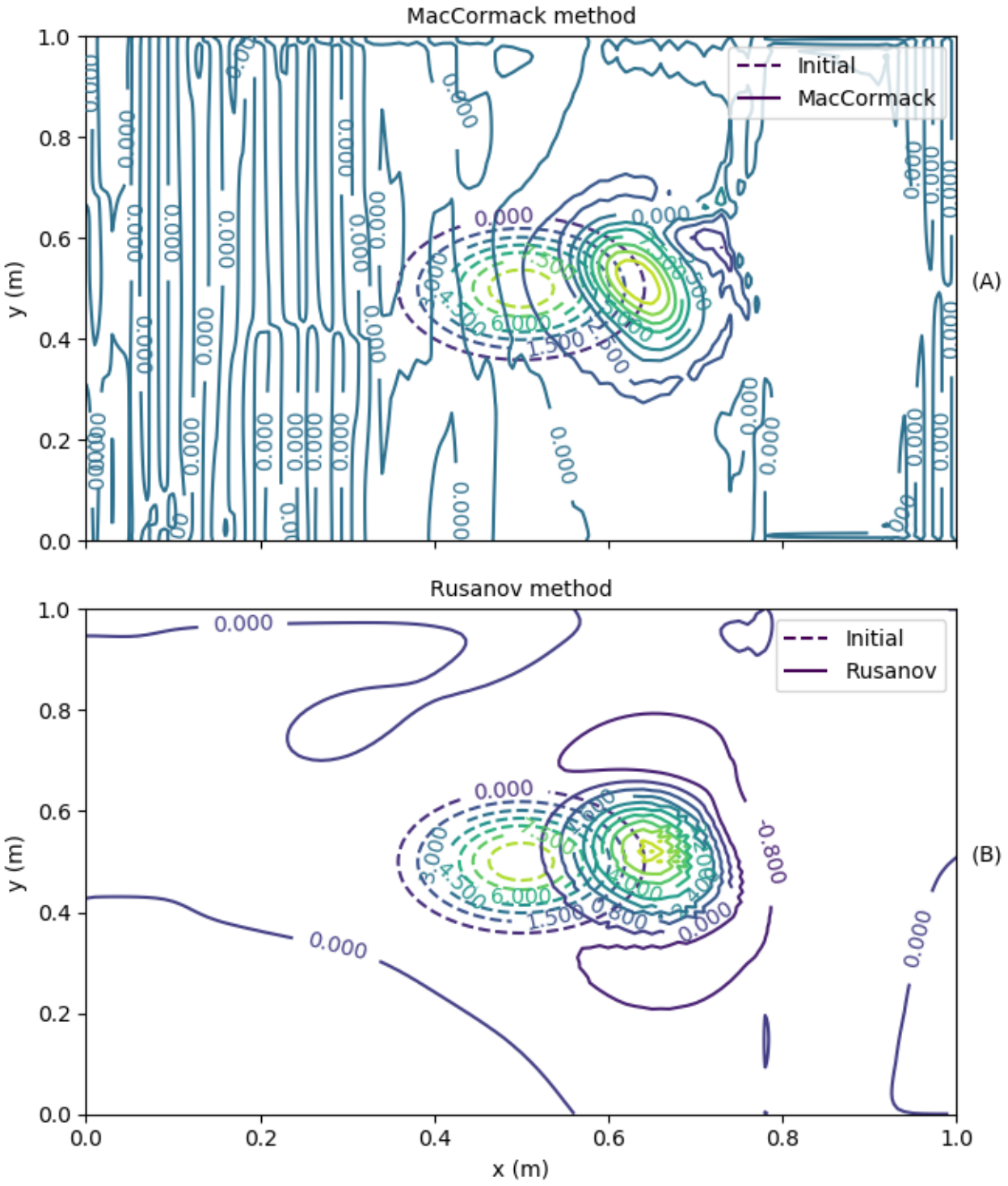

Figure 5: The MacCormack method and Rusanov method are used to model the interaction of a shock initially located at x_s = L/4 and an isentropic vortex initially located at x_c = L/2, y_c = L/2. Vorticity contours at a grid resolution of Δx = Δy = L/100 are shown for time t = (L/2)/S.

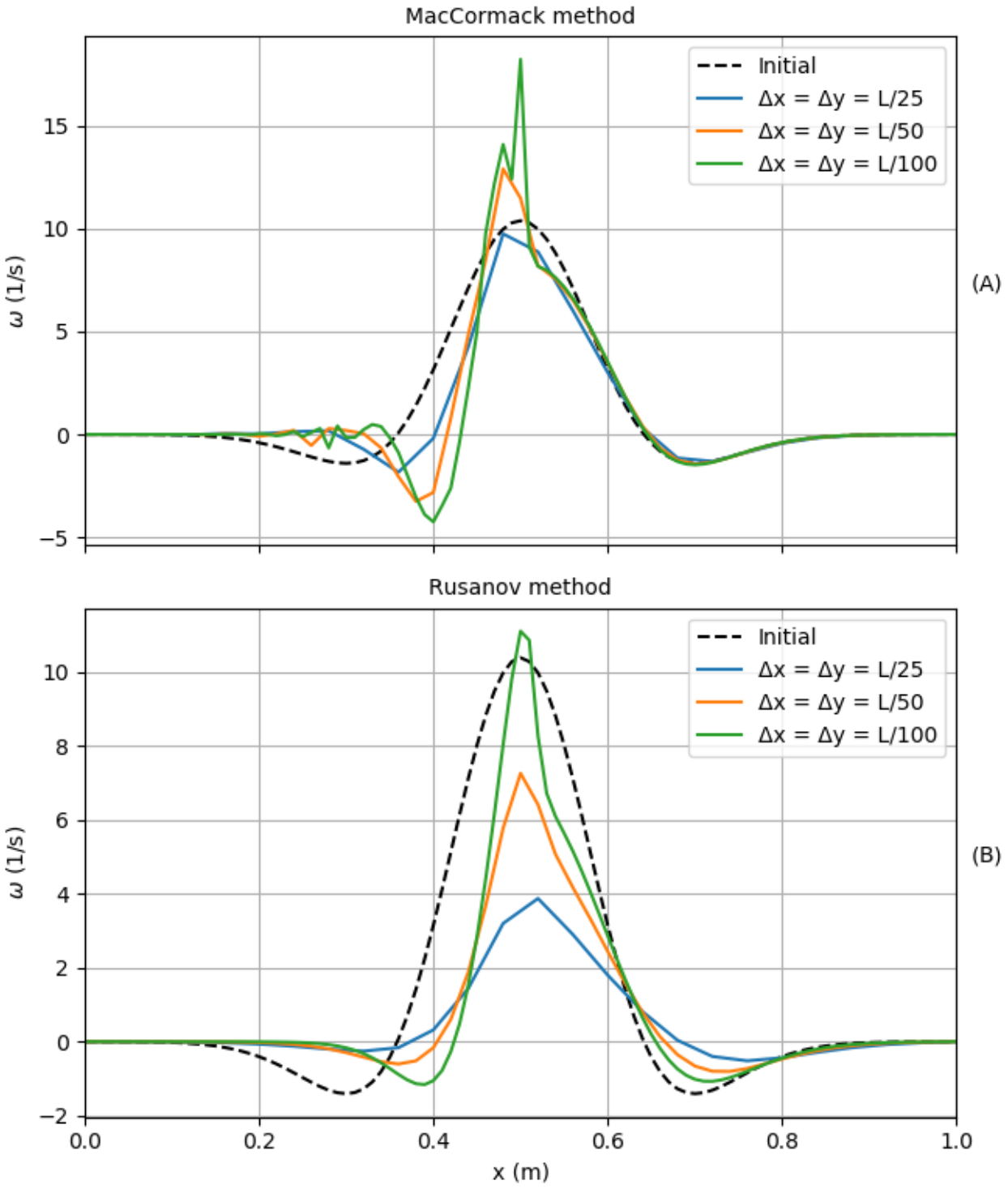

Figure 6: Vorticity profiles obtained using the MacCormack and Rusanov methods are compared to each other and to the analytical solution for the shock-vortex interaction at a time of t = (L/4)/S. Numerical results at grid resolutions of Δx = Δy = L/25, L/50, and L/100 are shown.

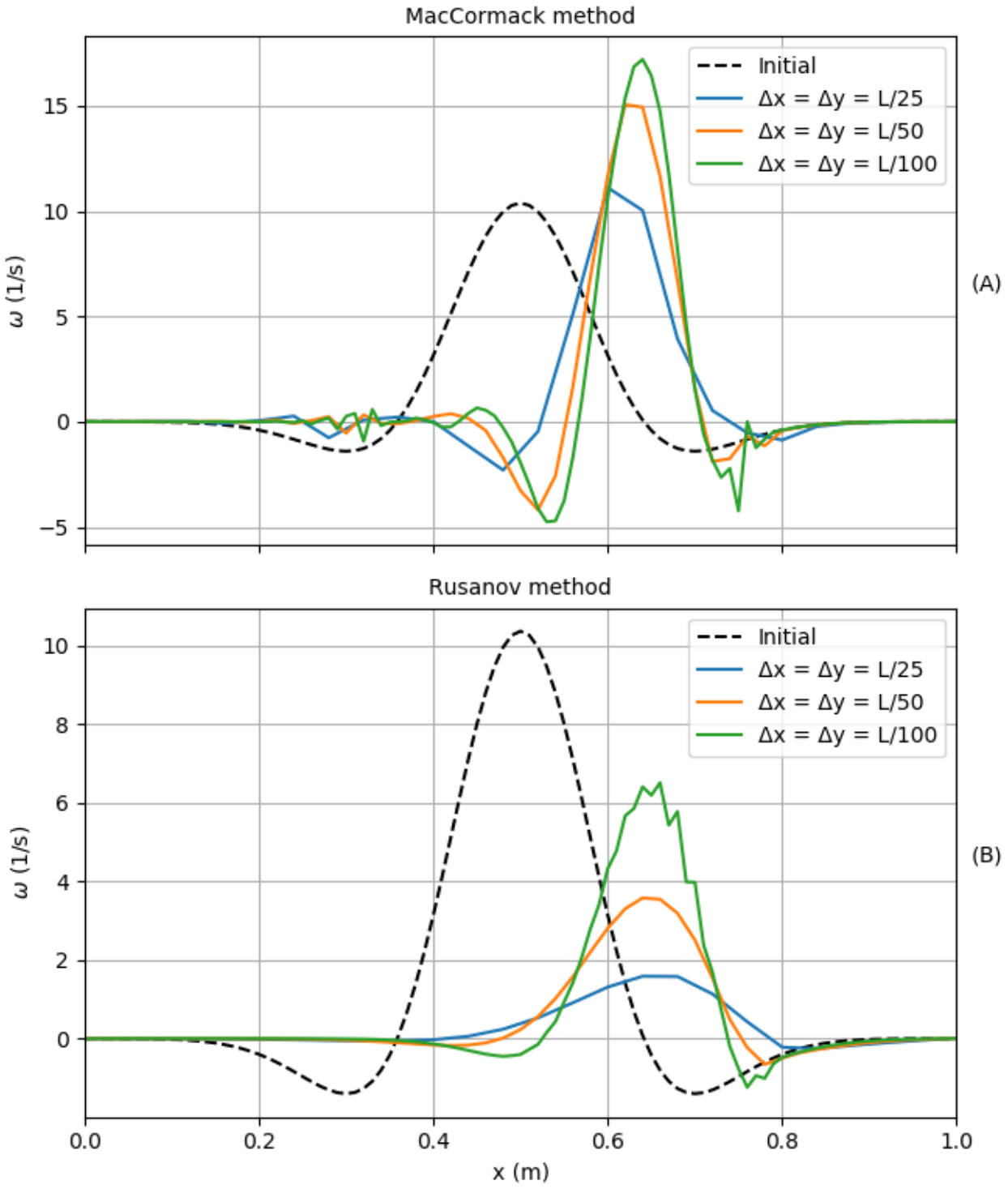

Figure 7: Vorticity profiles obtained using the MacCormack and Rusanov methods are compared to each other and to the analytical solution for the shock-vortex interaction at a time of t = (L/2)/S. Numerical results at grid resolutions of Δx = Δy = L/25, L/50, and L/100 are shown.

The interaction of a shock (initialized using the Sod problem parameters) initially located at $x_s = L/4$ and a 2D isentropic vortex initially located at $x_c = L/2$, $y_c = L/2$ is simulated using the 2D MacCormack and 2D Rusanov methods. CFL number is set to 0.9 for the MacCormack method and 0.7 for the Rusanov method. The Rusanov method suffers from odd-even decoupling if CFL number is set higher, which results in a characteristic checkerboard pattern in the vorticity contour plots.

Vorticity contour plots for both methods are shown for a grid resolution of $\Delta x = \Delta y = L/100$ at a final time of $t = (L/4)/S$, at which the shock has traveled to a position of $x_s = L/2$ (Fig. 4) and is in the center of the vortex, and a final time of $t = (L/2)/S$, at which the shock has traveled through the vortex to a position of $x_s = 3L/4$ (Fig. 5). The analytical vorticity contour is also plotted at the initial condition as a basis for comparison. Vorticity profiles for both methods are plotted at $y = L/2$ for grid resolutions of $\Delta x = \Delta y = L/25$, $L/50$, and $L/100$ for final time $t = (L/4)/S$ (Fig. 6) and $t = (L/2)/S$ (Fig. 7). These vorticity profiles are compared to the initial condition vorticity profile.

At $t = (L/4)/S$, the vorticity contour plot for the MacCormack method (Fig. 4a) shows that the shock is located in the middle of the vortex. The vorticity contour of the right side of the vortex closely matches the initial condition and appears largely unaffected, whereas the left side (which the shock has passed through) appears distorted and compressed, which results in an increase in vorticity. Oscillations are visible to the left of the shock at $x_s = L/2$ and at the right end of the computational domain at $x = L$ due to dispersive error, but these are small in magnitude. The vorticity contour plot for the Rusanov method at $t = (L/4)/S$ (Fig. 4b) exhibits similar distortion and compression on the left side, as well as a slight increase in vorticity that appears to have been largely counteracted by dissipative error.

The vorticity contour plot for the MacCormack method at $t = (L/2)/S$ (Fig. 5a) shows that the shock has passed through the vortex and is now at a position of $x_s = 3L/4$. The vortex has been displaced from its original position and now appears to have a center that lags slightly behind the shock front. The entire vortex now appears distorted, with significant narrowing of the vortex and compression of vorticity contours. Peak vorticity has also increased as a result of interaction with the shock. Oscillations due to dispersive error are now even more pronounced throughout the computational domain. The Rusanov method vorticity contour at $t = (L/2)/S$ (Fig. 5b) similarly shows displacement of the vortex center and distortion of the vorticity contours relative to the initial condition. Vorticity has also increased as a result of shock-vortex interaction, but this increase is largely mitigated by the high dissipative error in the Rusanov scheme. The spreading of the vortex due to dissipation also causes some interaction between the shock and the outermost regions of the vortex at $x_s = 3L/4$, making the distortion and compression of the vorticity contours more pronounced on the right side. This is in contrast to the MacCormack method contours, which indicate that the shock has passed entirely through the vortex structure.

Vorticity profiles for the MacCormack method at $t = (L/4)/S$ are shown in Fig. 6a, and largely support the observations made from the vorticity contours. The vorticity profile location coincides with that of the initial condition, indicating that the vortex center has not moved. Vorticity profiles appear to have narrowed relative to the initial condition due to shock-vortex interaction, and peak vorticity has increased for all but the coarsest grid solution (where dissipative error is highest). These large narrow peaks appear to be overshoots due to the discontinuity introduced by the shock, a well-known pitfall of the MacCormack method. Peak vorticity for the coarse, medium, and fine grid MacCormack method solution is 9.7394 1/s, 13.1130 1/s, and 18.2120 1/s respectively. The Rusanov method vorticity profile (Fig. 6b) shows this same narrowing effect, but the increase in vorticity from shock-vortex interaction is largely

suppressed due to dissipation. This dissipative error and consequent peak suppression and spreading is improved dramatically with grid refinement. Peak vorticity for the coarse, medium, and fine grid Rusanov method solution is 3.8699 1/s, 7.2530 1/s, and 11.2378 1/s respectively.

Vorticity profiles at t = (L/2)/S are shown in Fig. 7, and show the influence of the shock (now located at x_s = 3L/4) on the vortex structure. The MacCormack method vorticity profile (Fig. 7a) shows a significant shifting of the vorticity profile to the right relative to the initial condition, indicating that the vortex center has been displaced by the shock front. Vorticity has increased due to shock-vortex interaction, with vorticity peaks increasing with grid refinement. Peak vorticity for the coarse, medium, and fine grid is 11.1147 1/s, 16.1162 1/s, and 17.4989 1/s respectively. The fine grid velocity profile also shows a jagged oscillation that coincides with the shock front, a result of the leading dispersive error term in the MacCormack scheme. This oscillation has likely been dampened by higher dissipative error in the coarse and medium grid solutions. The Rusanov method vorticity profile (Fig. 7b) similarly shows a shifting of the vorticity profile to the right relative to the initial condition. The solution is highly dissipative, with severe peak suppression and spreading that largely counteracts the increases in vorticity that occur due to shock-vortex interaction. Dissipative error and consequent peak suppression is reduced with grid refinement, but this also results in jagged oscillations for the finest grid solution where dissipation is insufficient to dampen them. Peak vorticity for the coarse, medium, and fine grid is 1.5864 1/s, 3.6675 1/s, and 6.7549 1/s respectively.

Neither the MacCormack method nor the Rusanov method provide a grid converged solution for this shock-vortex interaction problem. Vorticity profiles of the MacCormack method and Rusanov method solutions at both simulation times show that peak vorticity increases with grid refinement. At a time of t = (L/2)/S, the MacCormack method contains an oscillation at the shock front in the fine grid solution that is not present in the medium or coarse grid solution. Similarly, the fine grid Rusanov solution at this time also contains oscillations on the entire right side of the vorticity profile – these are also not present in the more dissipative medium and coarse grid solutions.

Part 3: Vorticity analysis

Figure 8: Time history of ε, x_s = L/4, t = (L/2)/S

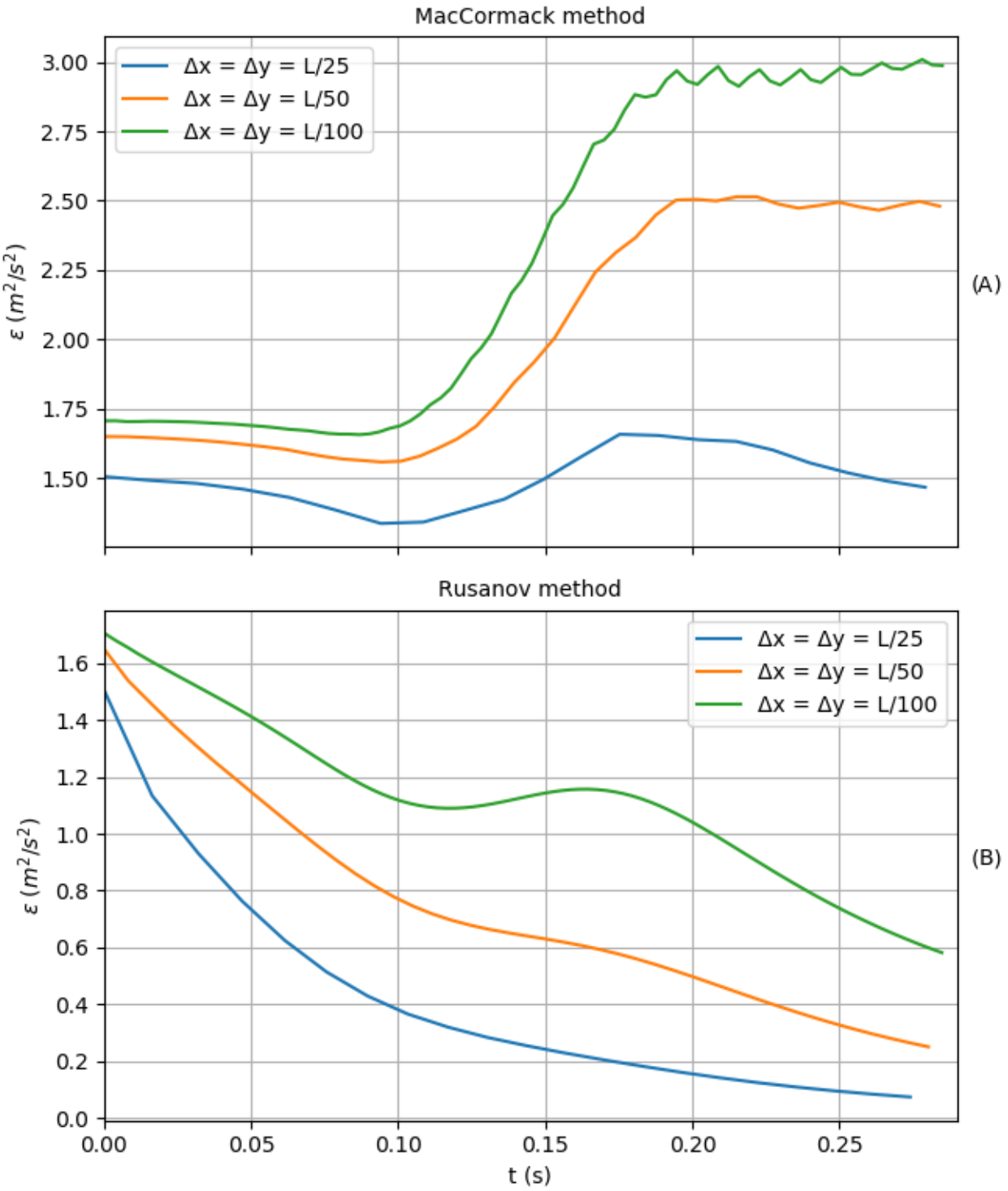

Figure 8: Time histories of enstrophy for the MacCormack and Rusanov method solutions are compared at grid resolutions of Δx = Δy = L/25, L/50, and L/100 at a final time of t = (L/2)/S.

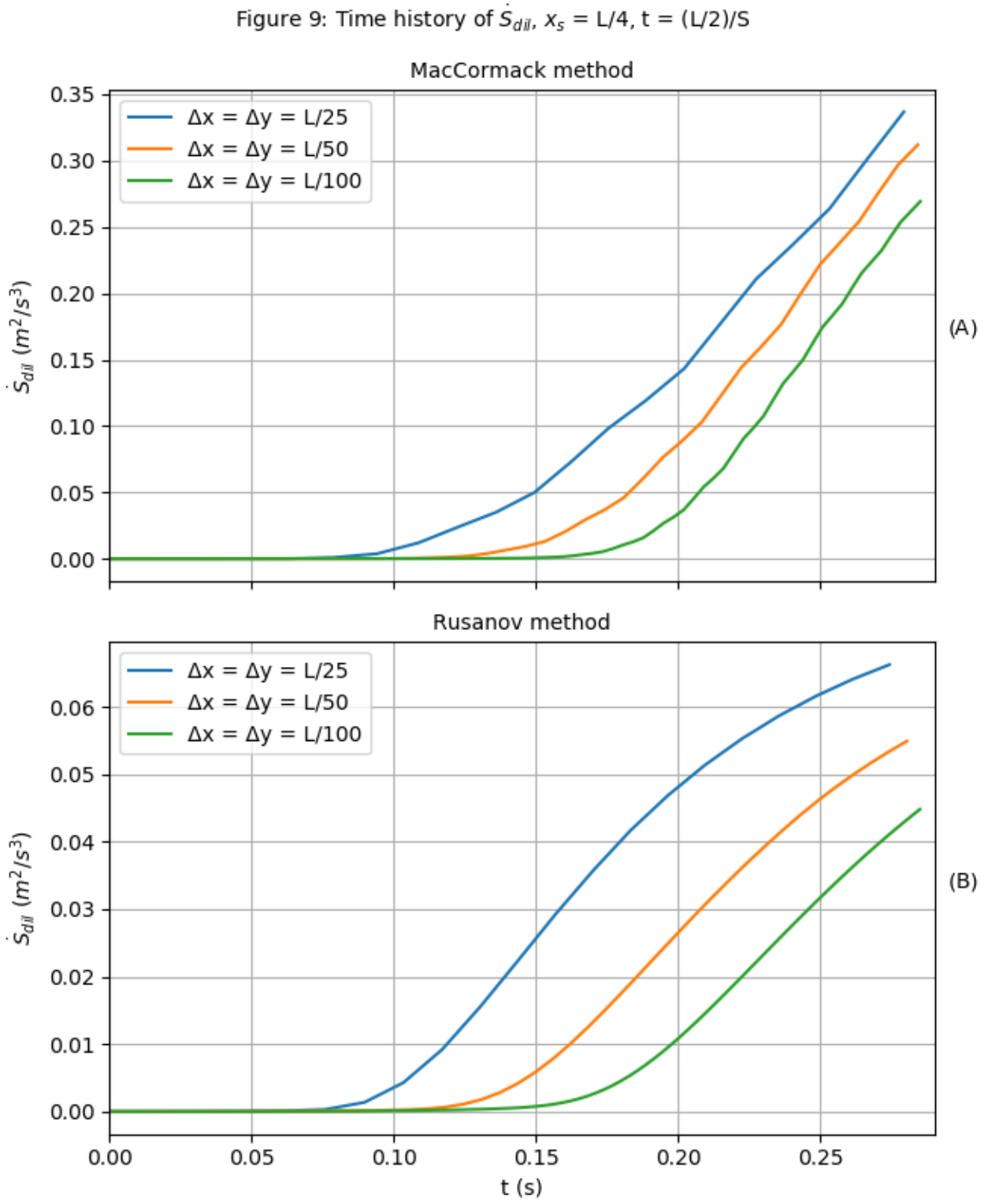

Figure 9: Time histories of the dilatation vorticity source term for the MacCormack and Rusanov method solutions are compared at grid resolutions of Δx = Δy = L/25, L/50, and L/100 at a final time of t = (L/2)/S.

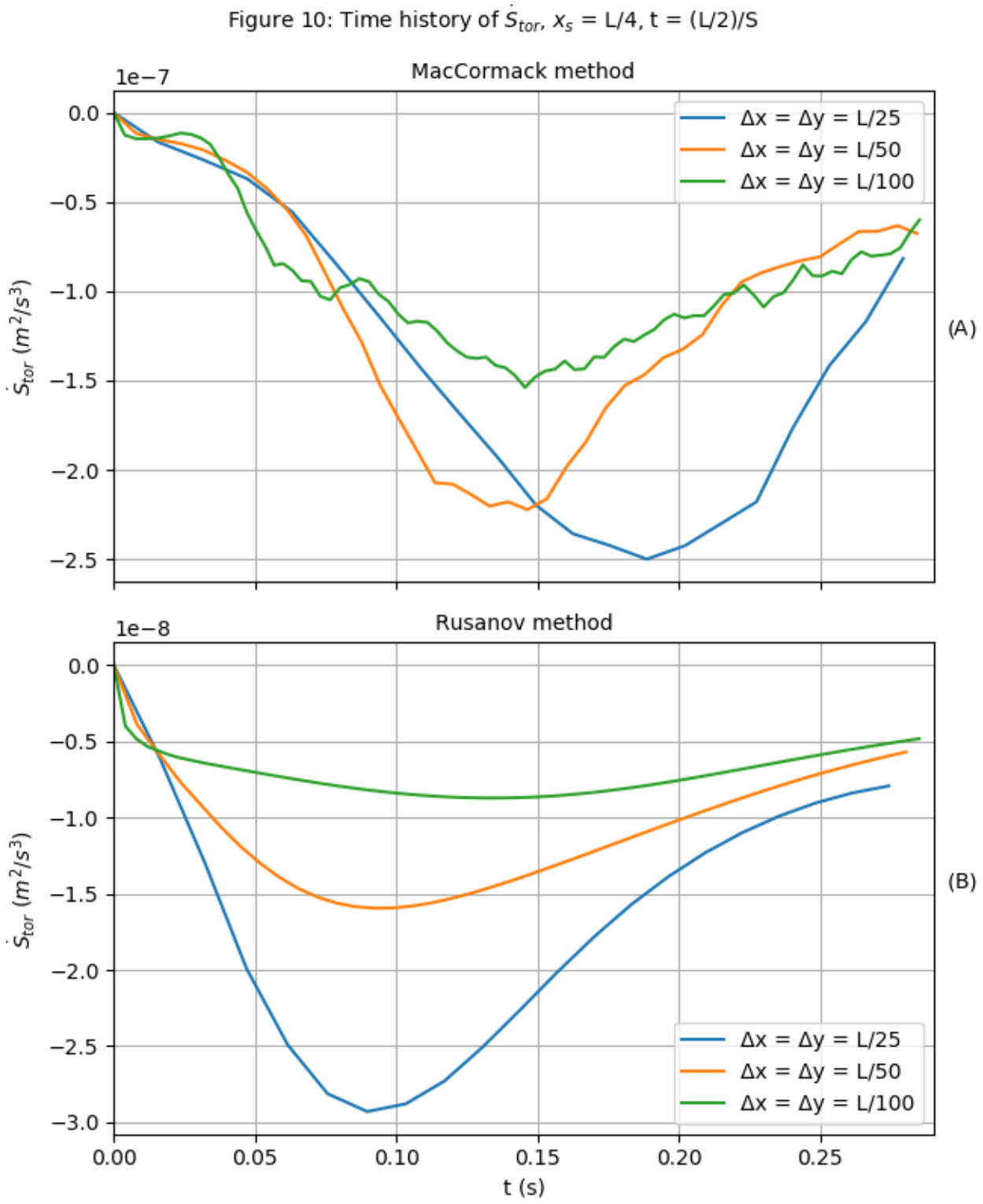

Figure 10: Time histories of the baroclinic vorticity source term for the MacCormack and Rusanov method solutions are compared at grid resolutions of Δx = Δy = L/25, L/50, and L/100 at a final time of t = (L/2)/S.

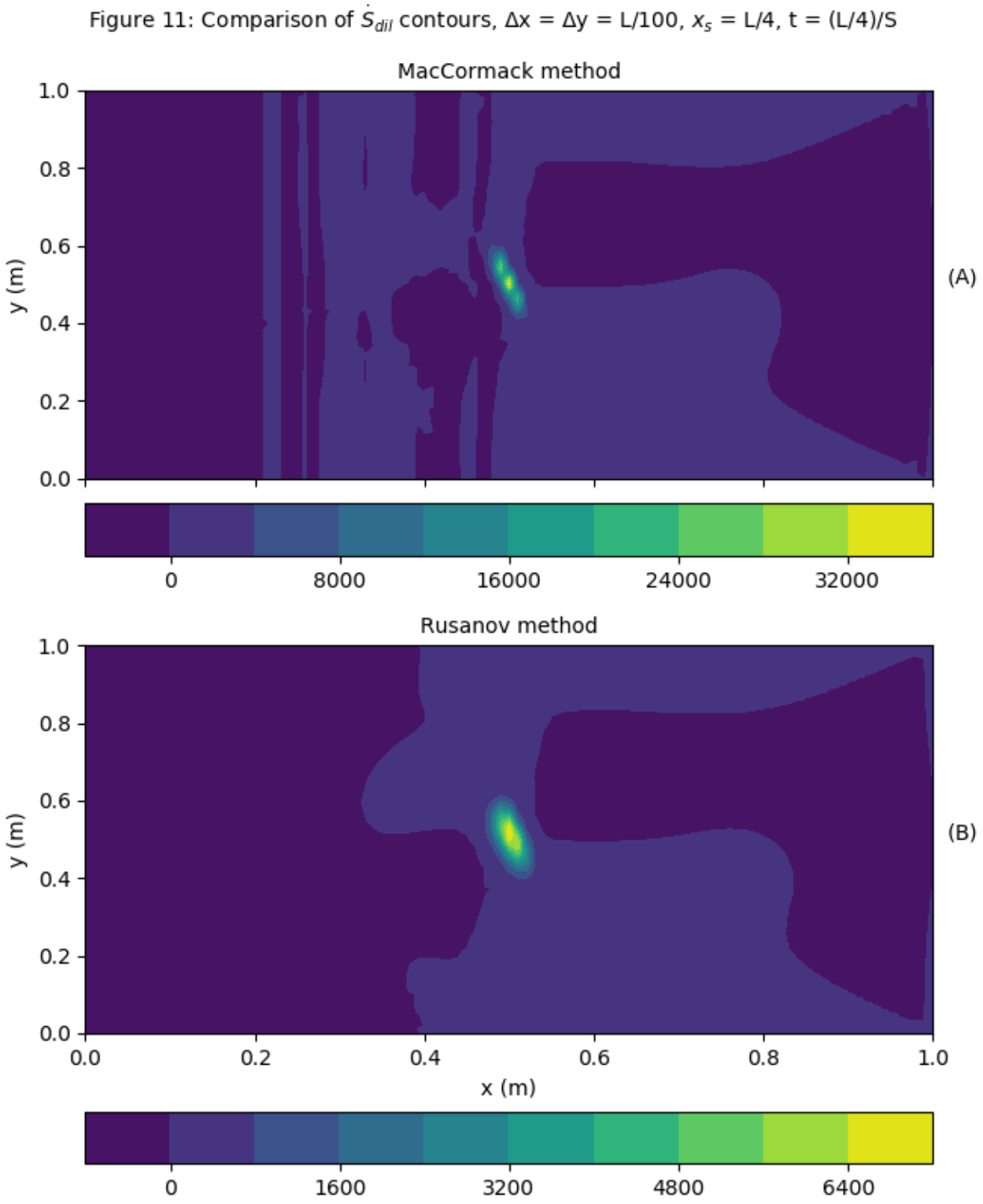

Figure 11: Contours of the dilatation source term for the MacCormack method solution and Rusanov method solution are plotted for a grid resolution of Δx = Δy = L/100 at a final time of t = (L/4)/S.

Figure 12: Comparison of $\dot{S}_{tor}$ contours, Δx = Δy = L/100, x_s = L/4, t = (L/4)/S

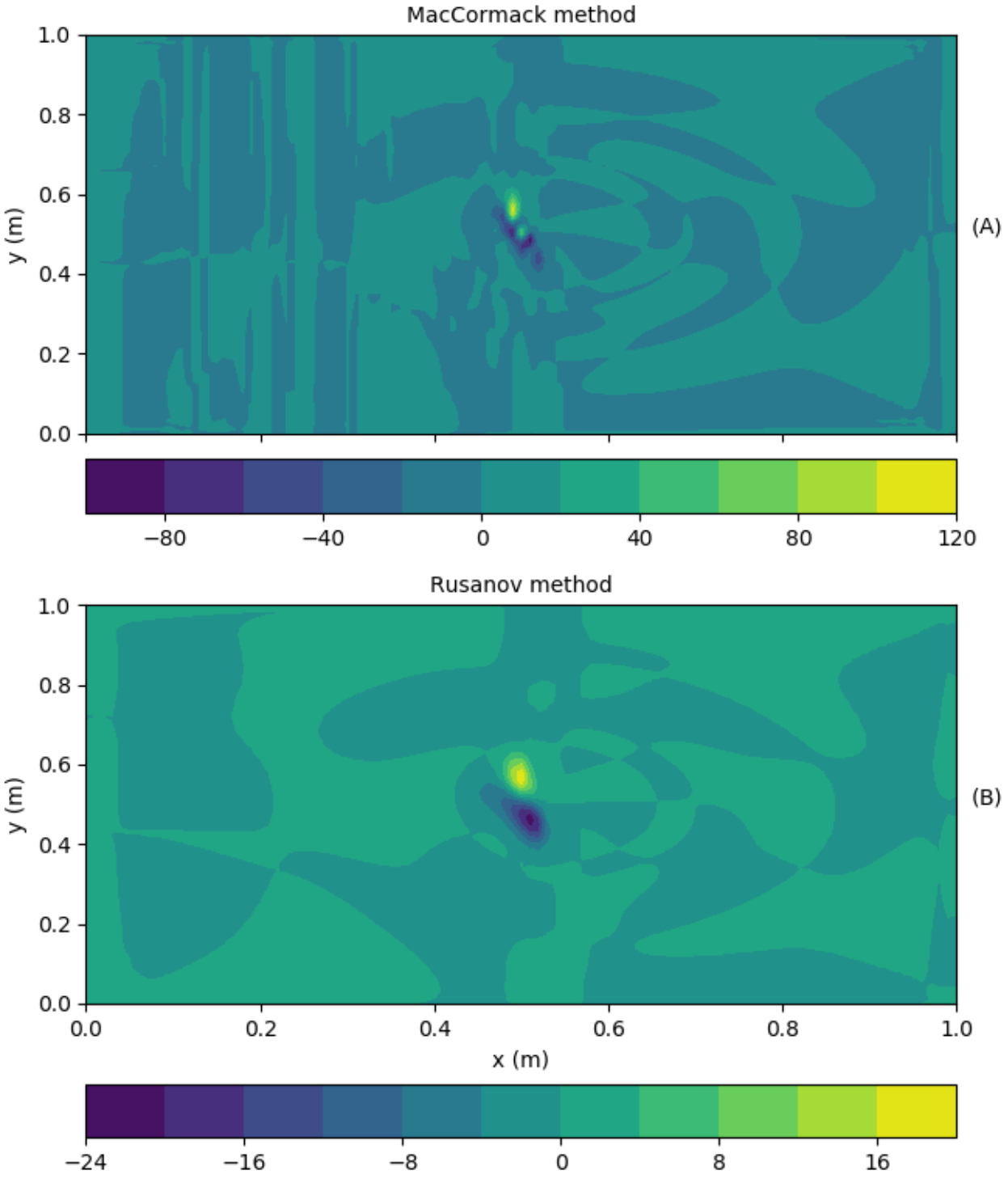

Figure 12: Contours of the baroclinic source term for the MacCormack method solution and Rusanov method solution are plotted for a grid resolution of Δx = Δy = L/100 at a final time of t = (L/4)/S.

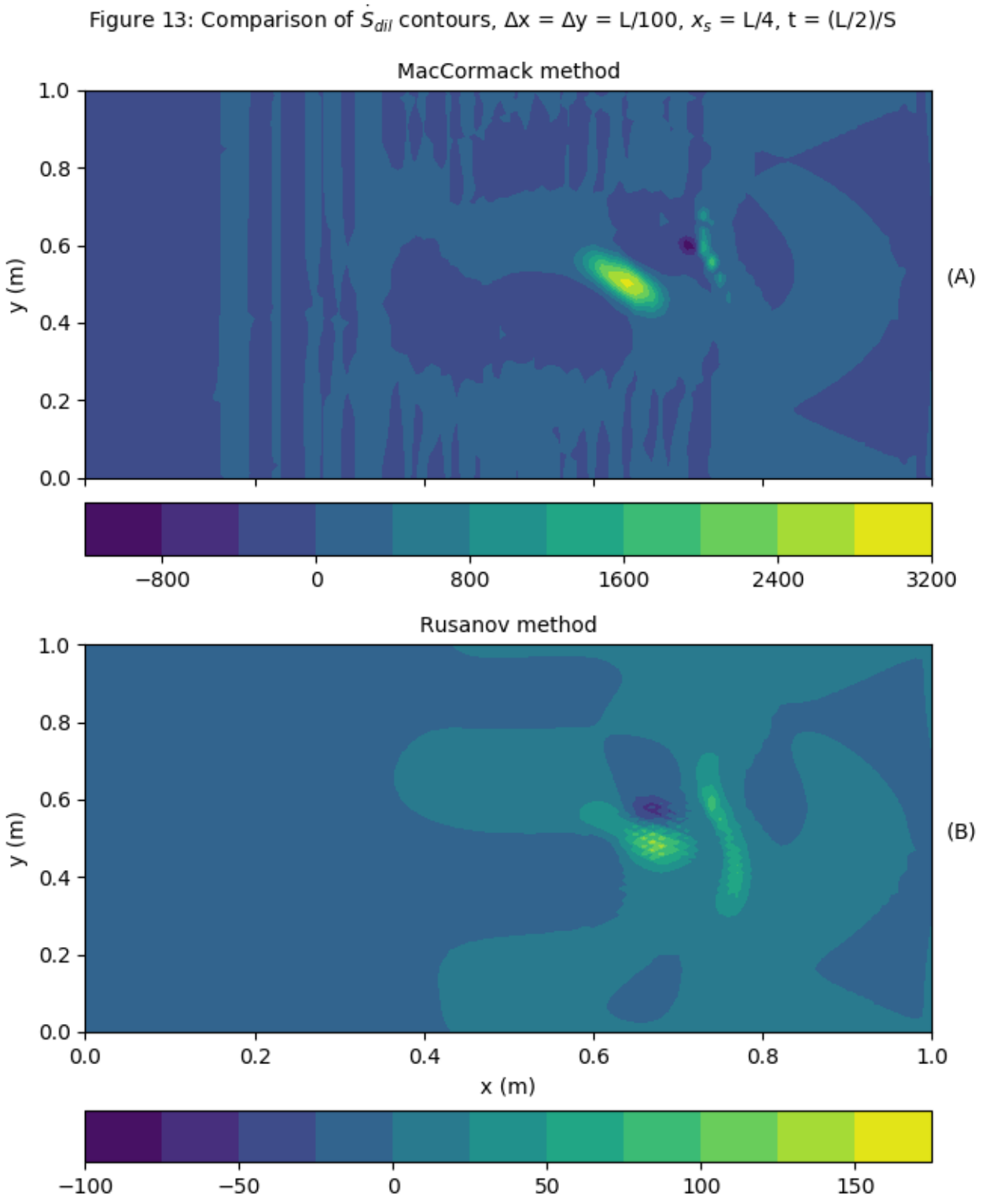

Figure 13: Contours of the dilatation source term for the MacCormack method solution and Rusanov method solution are plotted for a grid resolution of Δx = Δy = L/100 at a final time of t = (L/2)/S.

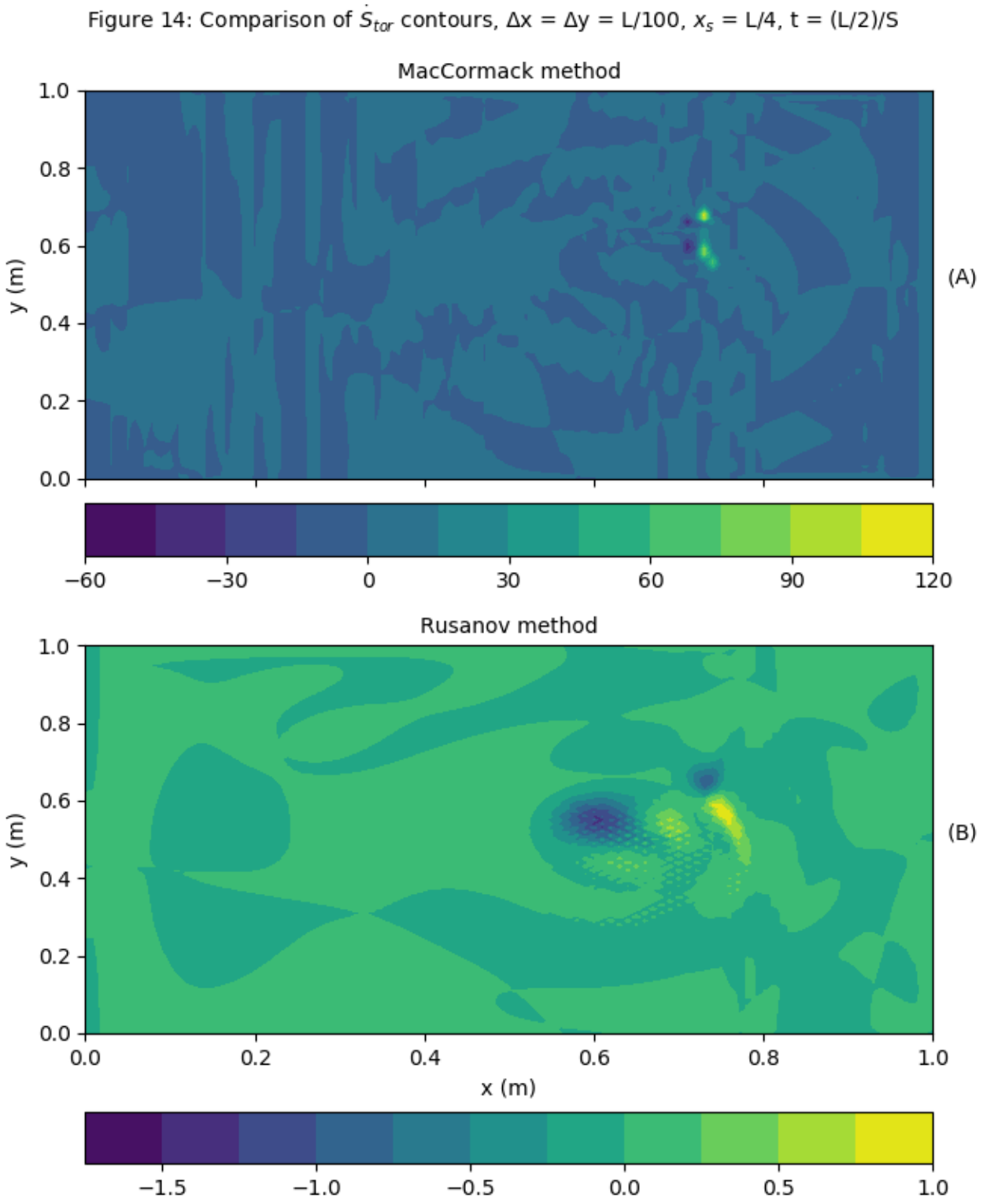

Figure 14: Contours of the baroclinic source term for the MacCormack method solution and Rusanov method solution are plotted for a grid resolution of Δx = Δy = L/100 at a final time of t = (L/2)/S.

Time histories of enstrophy (Fig. 8) and vorticity source terms corresponding to dilatation (Fig. 9) and baroclinic torque (Fig. 10) are plotted at grid resolutions of $\Delta x = \Delta y = L/25$, $L/50$, and $L/100$ for the MacCormack method and Rusanov method solutions. Contour plots of the dilatation source term and baroclinic source term are plotted for a grid resolution of $\Delta x = \Delta y = L/100$ at final times of $t = (L/4)/S$ (Fig. 11 for dilatation source term, Fig. 12 for baroclinic source term) and $t = (L/2)/S$ (Fig. 13 for dilatation source term, Fig. 12 for baroclinic source term).

The time histories of enstrophy are consistent with the vorticity contours and vorticity profiles generated in Part 2. Enstrophy remains fairly constant at all grid resolutions for the MacCormack method (Fig. 8a) until a time of approximately 0.10 seconds, when the shock begins to impact the vortex. Enstrophy then increases almost linearly until a time of approximately 0.20 seconds, at which point it remains nearly constant (other than some small oscillations for the medium and fine grid solutions). The vorticity contours and profiles for the MacCormack method reflect this – vorticity contours and profiles at $t = (L/4)/S$ show that vorticity has increased relative to the initial condition, while vorticity contours and profiles at $t = (L/2)/S$ have shifted but remain similar in magnitude to those at the earlier time. Enstrophy appears to steadily decrease at all grid resolutions for the Rusanov method (Fig. 8b) until a time of approximately 0.10 seconds, when the shock begins to impact the vortex. This reduction in enstrophy occurs due to the high dissipative error of the Rusanov method, and is considerably larger for lower grid resolutions. Enstrophy then appears to increase slightly up to a local peak (lower than the initial condition enstrophy) at a time of approximately 0.17 seconds for the $\Delta x = \Delta y = L/100$ case, while it continues to drop for the coarser grid solutions where dissipative error is higher. These findings are consistent with the changes in peak vorticity observed in the vorticity contours (Fig. 4b) and profiles (Fig. 6b) at $t = (L/4)/S$, where vorticity increased slightly due to shock-vortex interaction but was suppressed due to dissipation. Enstrophy then continues to drop once the shock has passed entirely through the vortex at approximately 0.20 seconds. This same phenomenon was observed for the vorticity contours (Fig. 5b) and profiles (Fig. 7b) at $t = (L/2)/S$ in Part 2 – while vorticity increased due to shock-vortex interaction, this increase was counteracted by dissipation and the vorticity at this later time was considerably lower than the initial condition.

Comparison of the time history of dilatation (Fig. 9) and baroclinic (Fig. 10) source terms suggests that dilatation is the dominant source of enstrophy for this problem. The dilatation source term begins to increase at a time of approximately 0.14 seconds for both numerical methods at the highest grid resolution, and earlier for the coarse (0.08 seconds) and medium (0.12 seconds) grid resolutions. The dilatation term reaches a peak value of approximately 0.345 for the MacCormack method and approximately 0.066 for the Rusanov method, both occurring at the final time $t = (L/2)/S$. The Rusanov method dilatation source terms are significantly smaller due to the effects of dissipative error. The baroclinic source term begins to decrease immediately for both numerical methods at the highest grid resolution. This term has a magnitude significantly lower than the dilatation source term (order 10^{-7} for the MacCormack method, 10^{-8} for the Rusanov method) and reaches its lowest value at approximately 0.14 seconds for the highest grid resolution for both methods. The Rusanov method baroclinic source term reaches a negative peak earlier for the medium (0.09 seconds) and coarse (0.08 seconds) grids, while the MacCormack method baroclinic source term reaches negative peaks at the same time for the medium and fine grid and slightly later (0.18 seconds) for the coarse grid.

The contour plots of dilatation at t = (L/4)/S (Fig. 11a for the MacCormack method, Fig. 11b for the Rusanov method) show that this source term is concentrated at x = L/2, where the shock has impacted the vortex and compressed it. This source term is significantly higher for the MacCormack method (a peak of approximately 32000) than the Rusanov method (a peak of approximately 6400) due to the high dissipative error of the Rusanov scheme. At a time of t = (L/2)/S (Fig. 13a for the MacCormack method, Fig. 13b for the Rusanov method), dilatation appears to have two peaks for both methods – one peak coincides with the vortex center, which lags slightly behind the shock, and the other peak coincides with the outermost regions of the vortex interacting with the shock at $x_s = 3L/4$. This is reflected by large values of the dilatation source term in these regions, which reach peaks of approximately 3200 for the MacCormack method and 150 for the Rusanov method.

The contour plots of baroclinic torque at t = (L/4)/S (Fig. 12a for the MacCormack method, Fig. 12b for the Rusanov method) show that this source term is also concentrated where the shock and vortex interact, and appears to have a dipole structure with positive and negative peaks of similar magnitudes. The baroclinic torque term has positive and negative peaks of approximately 120 and -80 for the MacCormack method, and peaks of approximately 16 and -24 for the more dissipative Rusanov method. At a time of t = (L/2)/S, the baroclinic source term contour plots (Fig. 14a for the MacCormack method, Fig. 14b for the Rusanov method) show a similar dipole structure that appears both at the vortex center and at the shock front. The positive and negative peaks of this source term are almost unchanged for the MacCormack method, and are approximately 120 and -60 respectively. The Rusanov method baroclinic source term has been reduced substantially, and how has a positive peak of approximately 1.0 and a negative peak of approximately -1.5. The contour plots of the dilatation and baroclinic source terms are consistent with the time history plots, and support the conclusion that the baroclinic source term is an insignificant contributor to enstrophy relative to the dilatation source term for this problem.

Summary and Conclusions

The interaction of a right-moving shock initially located at $x_s = L/4$ with an isentropic 2D vortex initially located at $x_c = L/2$, $y_c = L/2$ is solved numerically using the MacCormack method and Rusanov method. Contour plots of vorticity for both methods at a grid resolution of $\Delta x = \Delta y = L/100$ at final times of t = (L/4)/S and t = (L/2)/S show that the shock displaces and distorts the vortex as it passes through it. Vorticity contours become compressed as the shock passes through them, an effect most noticeable at t = (L/4)/S when the shock is located at the vortex center and the differences between the pre-shock and post-shock structure can be easily observed.

This compression of vorticity contours is evidence of negative dilatation in the post-shock region, something that is corroborated by the time history plots of the dilatation source term in the 2D inviscid vorticity transport equation. The dilatation source term remains steady for the MacCormack method and decreases due to dissipative error in the Rusanov method until the shock first interacts with the vortex. This interaction causes a linear positive increase in the dilatation source term for the MacCormack method as the shock interacts with the vortex, followed by a nearly constant value once the shock has passed through the vortex. The Rusanov method also shows an increase in this term during the shock-vortex interaction, which results in a local peak followed by a steady decrease due to dissipative error once the shock has passed through the vortex.

Mismatches in the pressure gradient and density gradient occur when the shock and vortex interact. Fluid elements that have density gradients that are not aligned with the pressure gradient around them will experience some torque, and as a result will rotate about their own axis to correct this misalignment. This can be explained with a physical analogy: consider a square block where half is polystyrene and the other half is lead. This is analogous to a density gradient in a fluid element. If this block is placed into a body of water such that the lead side is initially at the top and the polystyrene side is at the bottom, the block will spin such that the lead side is on the bottom. The body of water has hydrostatic pressure that increases with depth, which creates a pressure gradient that is misaligned with the density gradient of the block in its initial position. This effect, called baroclinic torque, is a source of vorticity generation in this shock-vortex interaction problem. Time history of the baroclinic torque term shows that this effect is relatively minor relative to the dilatation term, but this term often becomes an important factor in vorticity generation in combusting flows where temperature gradients (and therefore density gradients) are high.

The grid resolutions used in this study were insufficient to produce grid converged solutions with either numerical method tested. Preliminary testing with a more refined grid of $\Delta x = \Delta y = L/150$ resulted in changes to the peak vorticity observed for the MacCormack method, but appears to have caused a numerical instability in the Rusanov method. It is possible that this is another instance of odd-even decoupling, and it can be rectified by lowering the CFL number from 0.7 to 0.6 – however, making this change to maintain stability also disqualifies the trial from being used to judge grid convergence (since all preceding trials used a CFL number of 0.7). A formal grid convergence study is beyond the scope of this project, but such a study would necessitate using a CFL number such that stability is maintained for the entire range of grid resolutions tested.

Appendix – MATLAB Code

Rusanov method

```
# -*- coding: utf-8 -*-
"""
Created on Sat Apr  4 22:26:59 2020

@author: az
"""
def rusanov2D(N,L,t_final,CFL,u,v,rho,p,gamma,R):
    import numpy as np
    # First, establish some storage vectors
    omega = np.zeros((N,N)) # vorticity using centered differencing
    dilatation = np.zeros((N,N))
    baroclinic = np.zeros((N,N))
    dilatation_fin = np.zeros((N,N))
    baroclinic_fin = np.zeros((N,N))
    dx = L/(N-1)
    dy = L/(N-1)
    # Establish some more storage vectors for Q, F, G, Fhalf, Ghalf
    Q = np.zeros((4,N,N))
    #Q_update = np.zeros((4,N,N))
    F = np.zeros((4,N,N))
    F_half = np.zeros((4,N,N))
    G = np.zeros((4,N,N))
    G_half = np.zeros((4,N,N))
    x = np.linspace(0,L,N) # vectors for storing x and y coordinates
    y = np.linspace(0,L,N)
    ens1D = np.zeros(N)
    baroclinic1D = np.zeros(N)
    dilatation1D = np.zeros(N)

    # Calculate a, e, et, H at the initial condition for use in flux vectors
    e = p/((gamma-1)*rho)
    et = e+(u**2+v**2)/2
    H = e+p/rho+(u**2+v**2)/2
    a = np.sqrt(gamma*p/rho)

    # Set up initial time
    t = 0
    timestep = 0
    timevect = []
    ens = []
    Sdil = []
    Stor = []
    # Calculate dtx and dty similarly to the 1D method. I suspect this is where my problem is, since the solution appears to be valid if CFL number
is made low enough but blows up and becomes nonsense even with CFL = 0.5 or so
    dtx = CFL*dx/np.max(np.abs(u)+a)
    dty = CFL*dy/np.max(np.abs(v)+a)

    # Take the smallest value of the three and use that as the timestep
    dt = min(dtx,dty)

    #calculate initial Q and F vectors
    Q[0,:,:] = rho
    Q[1,:,:] = rho*u
    Q[2,:,:] = rho*v
    Q[3,:,:] = rho*et

    F[0,:,:] = rho*u
    F[1,:,:] = rho*u**2+p
    F[2,:,:] = rho*u*v
    F[3,:,:] = rho*u*H

    G[0,:,:] = rho*v
```

```
  G[1,:,:] = rho*u*v
  G[2,:,:] = rho*v**2+p
  G[3,:,:] = rho*v*H

  # This loop calculates the solution
  while t <= t_final:
     # This loop calculates vorticity dilatation and baroclinic
    for j in range(0,N):
      for i in range(0,N):

        # Same implementation of periodic BCs
        im=i-1
        ip=i+1
        jm=j-1
        jp=j+1

        if i == 0:
          im = 1
          #im = N-2
        if i == N-1:
          #ip = 1
          ip = N-2
        if j == 0:
          jm = N-2
        if j == N-1:
          jp = 1

        # Calculate vorticity numerically using central differencing
        omega[j,i] = (v[j,ip]-v[j,im])/(2*dx)-(u[jp,i]-u[jm,i])/(2*dy)
        dilatation[j,i] = -(u[j,ip]-u[j,im])/(2*dx)+(v[jp,i]-v[jm,i])/(2*dy)
        baroclinic[j,i] = ((rho[j,ip]-rho[j,im])/(2*dx))*((p[jp,i]-p[jm,i])/(2*dy))-((rho[jp,i]-rho[jm,i])/(2*dy))*((p[j,ip]-p[j,im])/(2*dx))

    for j in range(0,N):
      ens1D[j] = np.trapz(omega[j,:]**2,x)
      dilatation1D[j] = np.trapz(dilatation[j,:],x)
      baroclinic1D[j] = np.trapz(baroclinic[j,:],x)

    dilatation2D = np.trapz(dilatation1D,y)
    baroclinic2D = np.trapz(baroclinic1D,y)
    ens2D = np.trapz(ens1D,y)

    ens.append(ens2D)
    Sdil.append(dilatation2D)
    Stor.append(baroclinic2D)
    timevect.append(t)

    for j in range(0,N):
      for i in range(0,N): # PREDICTOR

        # This is just using a dummy index so I can implement the boundary conditions a bit more elegantly - I can just make ip=1 (second
node) if i = N-1 (last node)
        im=i-1
        ip=i+1
        jm=j-1
        jp=j+1

        # Periodic BCs implemented here using a series of if statements
        if i == 0:
          im = 1
          #im = N-2
        if i == N-1:
          #ip = 1
          ip = N-2
        if j == 0:
          jm = N-2
        if j == N-1:
```

```
            jp = 1

        sx = np.maximum(np.abs(u[j,i])+a[j,i],np.abs(u[j,ip])+a[j,ip])
        sy = np.maximum(np.abs(v[j,i])+a[j,i],np.abs(v[jp,i])+a[jp,i])

        F_half[:,j,i] = 0.5*(F[:,j,i]+F[:,j,ip]-sx*(Q[:,j,ip]-Q[:,j,i])) # calculate F_half
        G_half[:,j,i] = 0.5*(G[:,j,i]+G[:,jp,i]-sy*(Q[:,jp,i]-Q[:,j,i])) # calculate G_half

    for j in range(0,N):
      for i in range(0,N):

        im=i-1
        ip=i+1
        jm=j-1
        jp=j+1

        if i == 0:
          im = 1
          #im = N-2
        if i == N-1:
          #ip = 1
          ip = N-2
        if j == 0:
          jm = N-2
        if j == N-1:
          jp = 1

        Q[:,j,i] = Q[:,j,i]-dt*((F_half[:,j,i]-F_half[:,j,im])/dx+(G_half[:,j,i]-G_half[:,jm,i])/dy) # update value of Q

    # Calculate all the update values of properties we care about
    rho = Q[0,:,:]
    u = Q[1,:,:]/rho
    v = Q[2,:,:]/rho
    et = Q[3,:,:]/rho
    e = et-(u**2+v**2)/2
    p = e*(gamma-1)*rho # calculate values of rho, u, e, and p based on predictor Q
    T = p/(rho*R)
    H = e+p/rho +(u**2+v**2)/2
    a = np.sqrt(gamma*p/rho)

    F[0,:,:] = rho*u
    F[1,:,:] = rho*u**2+p
    F[2,:,:] = rho*u*v
    F[3,:,:] = rho*u*H

    G[0,:,:] = rho*v
    G[1,:,:] = rho*u*v
    G[2,:,:] = rho*v**2+p
    G[3,:,:] = rho*v*H

    dtx = CFL*dx/np.max(np.abs(u)+a)
    dty = CFL*dy/np.max(np.abs(v)+a)
    dt = min(dtx,dty)
    t = t+dt
    timestep = timestep+1
    print(t)

  dilatation_fin = ((2*omega**2)*dilatation)
  baroclinic_fin = ((omega/(2*rho**2))*(baroclinic))

  return u,v,rho,p,T,e,omega,timevect,ens,Sdil,Stor,dilatation_fin,baroclinic_fin
```

MacCormack method

```
# -*- coding: utf-8 -*-
"""
```

```
Created on Sat Apr  4 22:26:59 2020

@author: az
"""
def maccormack2D(N,L,t_final,CFL,u,v,rho,p,gamma,R):
    import numpy as np
    # First, establish some storage vectors
    omega = np.zeros((N,N)) # vorticity using centered differencing
    dilatation = np.zeros((N,N))
    baroclinic = np.zeros((N,N))
    dilatation_fin = np.zeros((N,N))
    baroclinic_fin = np.zeros((N,N))
    dx = L/(N-1)
    dy = L/(N-1)

    # Establish some more storage vectors for Q, F, G, Fhalf, Ghalf
    Q = np.zeros((4,N,N))
    Q_pred = np.zeros((4,N,N))
    Q_update = np.zeros((4,N,N))
    F = np.zeros((4,N,N))
    F_pred = np.zeros((4,N,N))
    G = np.zeros((4,N,N))
    G_pred = np.zeros((4,N,N))
    x = np.linspace(0,L,N) # vectors for storing x and y coordinates
    y = np.linspace(0,L,N)
    ens1D = np.zeros(N)
    baroclinic1D = np.zeros(N)
    dilatation1D = np.zeros(N)

    # Calculate a, e, et, H at the initial condition for use in flux vectors
    e = p/((gamma-1)*rho)
    et = e+(u**2+v**2)/2
    H = e+p/rho+(u**2+v**2)/2
    a = np.sqrt(gamma*p/rho)

    # Set up initial time and count
    t = 0
    count = 0
    timestep = 0
    timevect = []
    ens = []
    Sdil = []
    Stor = []

    # Calculate dtx and dty similarly to the 1D method. I suspect this is where my problem is, since the solution appears to be valid if CFL number
is made low enough but blows up and becomes nonsense even with CFL = 0.5 or so
    dtx = CFL*dx/np.max(np.abs(u)+a)
    dty = CFL*dy/np.max(np.abs(v)+a)

    # Take the smallest value of the three and use that as the timestep
    dt = min(dtx,dty)

    #calculate initial Q and F vectors
    Q[0,:,:] = rho
    Q[1,:,:] = rho*u
    Q[2,:,:] = rho*v
    Q[3,:,:] = rho*et

    F[0,:,:] = rho*u
    F[1,:,:] = rho*u**2+p
    F[2,:,:] = rho*u*v
    F[3,:,:] = rho*u*H

    G[0,:,:] = rho*v
    G[1,:,:] = rho*u*v
    G[2,:,:] = rho*v**2+p
```

```
  G[3,:,:] = rho*v*H

  # This loop calculates the solution
  while t <= t_final:
    # This loop calculates vorticity dilatation and baroclinic
    for j in range(0,N):
      for i in range(0,N):

        # Same implementation of periodic BCs
        im=i-1
        ip=i+1
        jm=j-1
        jp=j+1

        if i == 0:
          im = 1
          #im = N-2
        if i == N-1:
          #ip = 1
          ip = N-2
        if j == 0:
          jm = N-2
        if j == N-1:
          jp = 1

        # Calculate vorticity numerically using central differencing
        omega[j,i] = (v[j,ip]-v[j,im])/(2*dx)-(u[jp,i]-u[jm,i])/(2*dy)
        dilatation[j,i] = -(u[j,ip]-u[j,im])/(2*dx)+(v[jp,i]-v[jm,i])/(2*dy)
        baroclinic[j,i] = ((rho[j,ip]-rho[j,im])/(2*dx))*((p[jp,i]-p[jm,i])/(2*dy))-((rho[jp,i]-rho[jm,i])/(2*dy))*((p[j,ip]-p[j,im])/(2*dx))

    for j in range(0,N):
      ens1D[j] = np.trapz(omega[j,:]**2,x)
      dilatation1D[j] = np.trapz(dilatation[j,:],x)
      baroclinic1D[j] = np.trapz(baroclinic[j,:],x)

    dilatation2D = np.trapz(dilatation1D,y)
    baroclinic2D = np.trapz(baroclinic1D,y)
    ens2D = np.trapz(ens1D,y)

    ens.append(ens2D)
    Sdil.append(dilatation2D)
    Stor.append(baroclinic2D)
    timevect.append(t)

    for j in range(0,N):
      for i in range(0,N): # PREDICTOR

        # This is just using a dummy index so I can implement the boundary conditions a bit more elegantly - I can just make ip=1 (second
node) if i = N-1 (last node)
        im=i-1
        ip=i+1
        jm=j-1
        jp=j+1

        if i == 0:
          im = 1
          #im = N-2
        if i == N-1:
          #ip = 1
          ip = N-2
        if j == 0:
          jm = N-2
        if j == N-1:
          jp = 1
```

```
        # Use the count variable to go through all the permutations of forward/backward differencing for the predictor-corrector to avoid
biasing solution
        if count == 0:
            Q_pred[:,j,i] = Q[:,j,i]-dt*((F[:,j,ip]-F[:,j,i])/dx+(G[:,jp,i]-G[:,j,i])/dy) # calculate predictor value of Q
        if count == 1:
            Q_pred[:,j,i] = Q[:,j,i]-dt*((F[:,j,ip]-F[:,j,i])/dx+(G[:,j,i]-G[:,jm,i])/dy)
        if count == 2:
            Q_pred[:,j,i] = Q[:,j,i]-dt*((F[:,j,i]-F[:,j,im])/dx+(G[:,j,i]-G[:,jm,i])/dy)
        if count == 3:
            Q_pred[:,j,i] = Q[:,j,i]-dt*((F[:,j,i]-F[:,j,im])/dx+(G[:,jp,i]-G[:,j,i])/dy)

    rho_pred = Q_pred[0,:,:]
    u_pred = Q_pred[1,:,:]/rho_pred
    v_pred = Q_pred[2,:,:]/rho_pred
    et_pred = Q_pred[3,:,:]/rho_pred
    e_pred = et_pred-(u_pred**2+v_pred**2)/2
    p_pred = e_pred*(gamma-1)*rho_pred # calculate values of rho, u, e, and p based on predictor Q
    H_pred = e_pred + p_pred/rho_pred +(u_pred**2+v_pred**2)/2

    F_pred[0,:,:] = rho_pred*u_pred
    F_pred[1,:,:] = rho_pred*u_pred**2+p_pred
    F_pred[2,:,:] = rho_pred*u_pred*v_pred
    F_pred[3,:,:] = rho_pred*u_pred*H_pred

    G_pred[0,:,:] = rho_pred*v_pred
    G_pred[1,:,:] = rho_pred*u_pred*v_pred
    G_pred[2,:,:] = rho_pred*v_pred**2+p_pred
    G_pred[3,:,:] = rho_pred*v_pred*H_pred

    for j in range(0,N):
      for i in range(0,N): # CORRECTOR

        # Same deal with the periodic BCs here for the corrector step
        im=i-1
        ip=i+1
        jm=j-1
        jp=j+1

        if i == 0:
            im = 1
            #im = N-2
        if i == N-1:
            #ip = 1
            ip = N-2
        if j == 0:
            jm = N-2
        if j == N-1:
            jp = 1

        # Same deal here with the forward/backwards difference sequencing
        if count == 0:
            Q_update[:,j,i] = 0.5*(Q[:,j,i]+Q_pred[:,j,i]-dt*((F_pred[:,j,i]-F_pred[:,j,im])/dx+(G_pred[:,j,i]-G_pred[:,jm,i])/dy)) # Calculate corrector
value of Q
        if count == 1:
            Q_update[:,j,i] = 0.5*(Q[:,j,i]+Q_pred[:,j,i]-dt*((F_pred[:,j,i]-F_pred[:,j,im])/dx+(G_pred[:,jp,i]-G_pred[:,j,i])/dy))
        if count == 2:
            Q_update[:,j,i] = 0.5*(Q[:,j,i]+Q_pred[:,j,i]-dt*((F_pred[:,j,ip]-F_pred[:,j,i])/dx+(G_pred[:,jp,i]-G_pred[:,j,i])/dy))
        if count == 3:
            Q_update[:,j,i] = 0.5*(Q[:,j,i]+Q_pred[:,j,i]-dt*((F_pred[:,j,ip]-F_pred[:,j,i])/dx+(G_pred[:,j,i]-G_pred[:,jm,i])/dy))

    Q = Q_update[:,:,:]
    rho = Q[0,:,:]
    u = Q[1,:,:]/rho
    v = Q[2,:,:]/rho
    et = Q[3,:,:]/rho
    e = et-(u**2+v**2)/2
```

```
    p = e*(gamma-1)*rho # calculate values of rho, u, e, and p based on predictor Q
    T = p/(rho*R)
    H = e+p/rho +(u**2+v**2)/2
    a = np.sqrt(gamma*p/rho)

    F[0,:,:] = rho*u
    F[1,:,:] = rho*u**2+p
    F[2,:,:] = rho*u*v
    F[3,:,:] = rho*u*H

    G[0,:,:] = rho*v
    G[1,:,:] = rho*u*v
    G[2,:,:] = rho*v**2+p
    G[3,:,:] = rho*v*H

    dtx = CFL*dx/np.max(np.abs(u)+a)
    dty = CFL*dy/np.max(np.abs(v)+a)
    dt = min(dtx,dty)
    t = t+dt
    timestep = timestep+1
    print(t)
    count = count+1

    # Use an if statement to reset the counter if it exceeds 3
    if count > 3:
        count = 0

  dilatation_fin = ((2*omega**2)*dilatation)
  baroclinic_fin = ((omega/(2*rho**2))*(baroclinic))

  return u,v,rho,p,T,e,omega,timevect,ens,Sdil,Stor,dilatation_fin,baroclinic_fin
```

Isentropic vortex initialization

```
# -*- coding: utf-8 -*-
"""
Created on Sat Apr  4 22:26:59 2020

@author: az
"""
def init2Dvortex(N,L,Rc,xc,yc,Tinf,pinf,gamma,R,Mac,uinf,vinf):
  import numpy as np

  # First, establish some storage vectors
  u = np.zeros((N,N))
  v = np.zeros((N,N))
  T = np.zeros((N,N))

  omega = np.zeros((N,N)) # vorticity using centered differencing
  omegaex = np.zeros((N,N)) # vorticity using analytical expression

  dx = L/(N-1)
  dy = L/(N-1)

  x = np.linspace(0,L,N) # vectors for storing x and y coordinates
  y = np.linspace(0,L,N)

  Tc = Tinf/(1+((gamma-1)/2)*Mac**2) # Tc and ac are calculated since we need ac to use equation 4a and 4b
  ac = np.sqrt(gamma*R*Tc)

  # This nested loop initializes the flow field
  for j in range(0,N):
      for i in range(0,N):
          rstar = np.sqrt((x[i]-xc)**2+(y[j]-yc)**2)/Rc #Calculate rstar, ystar, and xstar here since they are used repeatedly
          ystar = (y[j]-yc)/Rc
          xstar = (x[i]-xc)/Rc
```

```
        rstarexp = np.exp((1-(rstar)**2)/2) #This is the exponential function and everything inside it
        u[j,i] = uinf-Mac*ac*ystar*rstarexp #Calculate u and v using equation 4 at every node - note that the indexing is j,i since rows in the array
correspond to the y-axis
        v[j,i] = vinf+Mac*ac*xstar*rstarexp
        T[j,i] = Tinf*(1-(((gamma-1)/2)*Mac**2/(1+((gamma-1)/2)*Mac**2))*rstarexp*rstar**2) # Calculate temperature using eq. 6
        omegaex[j,i] = (Mac*ac/Rc)*rstarexp*(2-xstar**2-ystar**2) # Calculate vorticity at each node using analytical expression

    p = pinf*(T/Tinf)**((gamma)/(gamma-1)) # Can now calculate p, rho, e based on the other values we have
    rho = p/(R*T)
    e = p/((gamma-1)*rho)
    # Initialization complete

    # This loop calculates vorticity
    for j in range(0,N):
        for i in range(0,N):

            # Same implementation of periodic BCs
            im=i-1
            ip=i+1
            jm=j-1
            jp=j+1

            if i == 0:
                im = N-2
            if i == N-1:
                ip = 1
            if j == 0:
                jm = N-2
            if j == N-1:
                jp = 1

            # Calculate vorticity numerically using central differencing
            omega[j,i] = (v[j,ip]-v[j,im])/(2*dx)-(u[jp,i]-u[jm,i])/(2*dy)

    # Calculate L2 vorticity error and circulation error here
    L2omega = np.sqrt(np.sum((omega-omegaex)**2))/(N**2)
    circ1D = np.zeros(N)
    eps1D = np.zeros(N)

    for j in range(0,N):
        circ1D[j] = np.trapz(omega[j,:],x)
        eps1D[j] = np.trapz(omega[j,:]**2,x)

    circ2D = np.trapz(circ1D,y)
    eps2D = np.trapz(eps1D,y)
    circerr = circ2D/np.sqrt(eps2D)

    return u, v, rho, p, T, e, omega, omegaex, L2omega, circerr
```