PROJECT 5: 2D ISENTROPIC VORTEX

MAE540

Adrian Zebrowski
April 6th, 2020

Problem Statement

The one-dimensional MacCormack and Rusanov methods implemented previously are extended to two dimensions, with the objective of capturing the behavior of a two-dimensional isentropic vortex in a square domain of side length L with periodic boundary conditions. Several cases are considered: a baseline case where the vortex remains in the middle of the domain, a horizontal convection case where the vortex passes through the end of the domain at x = L, a vertical convection case where the vortex passes through the end of the computational domain at y = L, a diagonal convection case where the vortex passes through the corner of the computational domain at x = L, y = L, and a compressibility case where the vortex Mach number $M_{a,c}$ is raised from 0.3 to 1.5 while it remains in the middle of the domain. Results are generated for each case at grid resolutions of N = 25, N = 50, and N = 100. Contours of vorticity ω, as well as profiles of u and ω at $x = x_c$, are plotted for the analytical solution and both numerical methods. Vorticity L2 error and normalized circulation error for each grid resolution and method are tabulated. Differences in the solutions provided by the MacCormack and Rusanov methods (relative to each other and the analytical solution) for each case are discussed qualitatively in terms of the contour and profile plots generated, and quantitatively in terms of the L2 vorticity errors and circulation errors.

The combinations of case, numerical method, and grid resolution used to generate results are tabulated below.

Table 1: Cases, methods, and grid resolutions used in study

				MacCormack			**Rusanov**		
				Grid resolution (N)					
Case	$\boldsymbol{u_\infty}$	$\boldsymbol{v_\infty}$	$\boldsymbol{M_{a,c}}$	**25**	**50**	**100**	**25**	**50**	**100**
Baseline	0	0	0.3	✓	✓	✓	✓	✓	✓
Horizontal convection	$0.3a_0$	0	0.3	✓	✓	✓	✓	✓	✓
Vertical convection	0	$0.3a_0$	0.3	✓	✓	✓	✓	✓	✓
Diagonal convection	$0.3\left(\frac{\sqrt{2}}{2}\right)a_0$	$0.3\left(\frac{\sqrt{2}}{2}\right)a_0$	0.3	✓	✓	✓	✓	✓	✓
Compressibility	0	0	1.5	x	x	✓	x	x	✓

Table 1: The cases used to generate the results in this study are tabulated, with key parameters such as vortex Mach number and far-field velocities included. Combinations of grid resolution N and case that have been considered are indicated with a check mark.

Method of Solution

The two-dimensional MacCormack method and two-dimensional 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*, vortex radius *Rc*, vortex center location *xc* and *yc*, stagnation temperature *T0*, stagnation pressure *p0*, vortex Mach number *Mac*, far-field velocities *uinf* and *vinf*, final time *t_final*, specific heat ratio *gamma*, and gas constant *R*. The functions then return spatial vectors *x* and *y*, velocity *u* and *v*, temperature *T*, pressure *p*, density *rho*, energy *e*, vorticity *omega*, L2 error of vorticity *L2omega*, circulation error *circerr,* approximate shadowgraph *shadow*, and compressibility metric *comp*.

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, omega*, and *omegaex* are created. CFL condition and minimum timestep are defined. Spatial steps *dx* and *dy* are calculated from domain length and grid resolution. Temperature at the center of the vortex *Tc* is calculated from the stagnation temperature and vortex Mach number using the isentropic temperature equation. Speed of sound at the vortex center, *ac*, is calculated using *gamma, R*, and *Tc*. The initial condition must now be generated based on these parameters, which is accomplished using a nested *for* loop that iterates over the rows (corresponding to the *y* direction, outer loop) and columns (corresponding to the *x* direction, inner loop) of the storage arrays. Inside of the nested loop, *rstar, ystar, xstar*, and *rstarexp* are calculated first for efficiency, since these values are used for multiple calculations. 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 by redirecting indices as needed, such that *i-1* with respect to the first node is the second to last node and *i+1* with respect to the last node is the second node (the same applies for vertical index *j*). 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 2 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 *rusanov2Dvortex* code. After the main time loop, the

vorticity, L2 vorticity error, normalized circulation error (which requires enstrophy and circulation to be calculated), approximate shadowgraph metric, and compressibility metric are calculated.

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*. The derivation of the equations that are used to obtain the analytical solution and initialize the flow field are included in the Appendix.

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

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

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

Table 2: 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 2: 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

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

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

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

Equations (analytical solution/initialization)

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

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

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

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

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

$$T_c = \frac{T_0}{1 + \frac{\gamma - 1}{2} M_{a,c}^2} \tag{10}$$

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

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

Results and Discussion

Part 1: Initialization

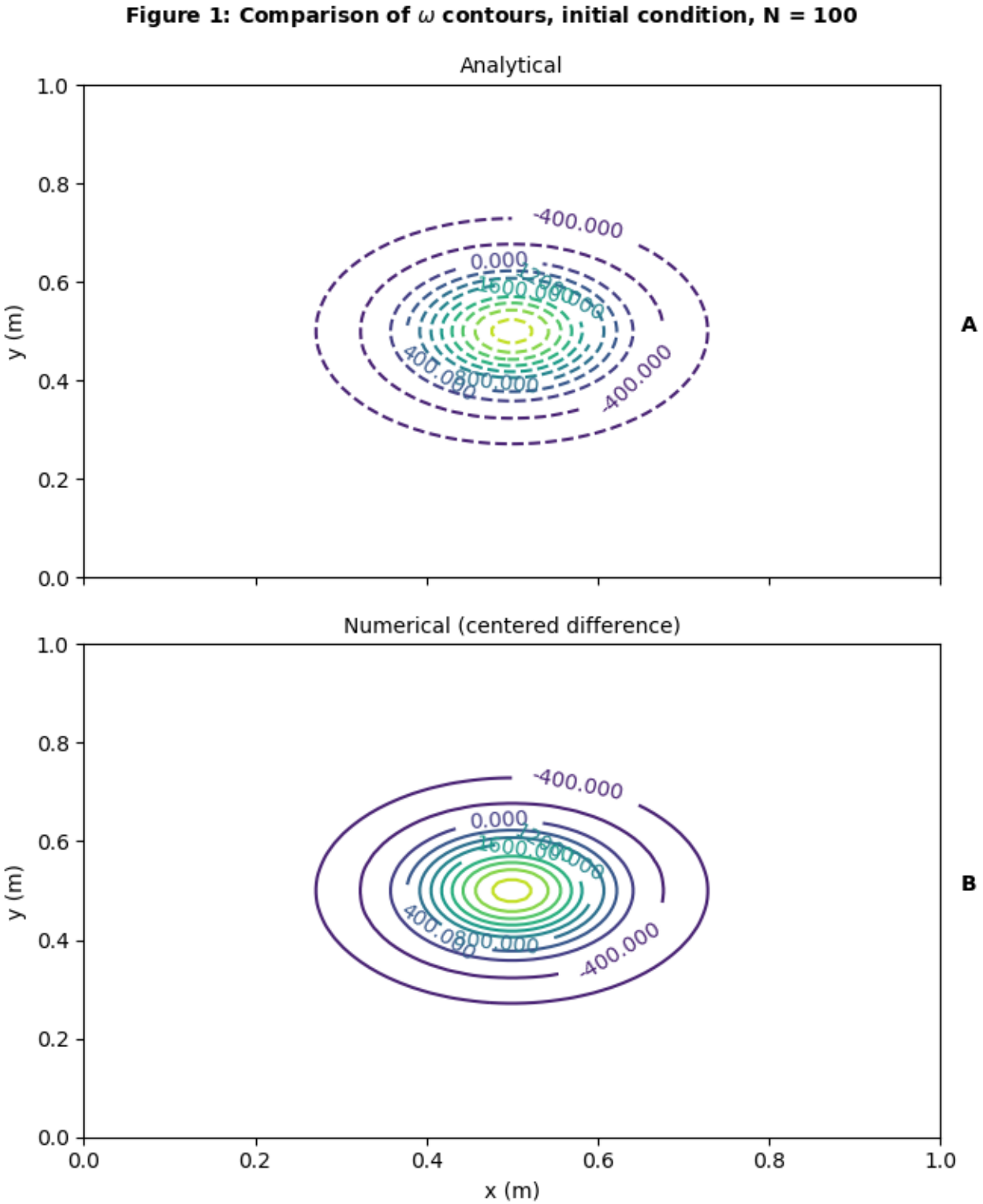

Figure 1: Contours of analytical vorticity and numerical vorticity (calculated using central differencing) are compared for the initial condition at a grid resolution of N = 100.

Figure 2: Comparison of ω profiles at $x = x_c$, initial condition

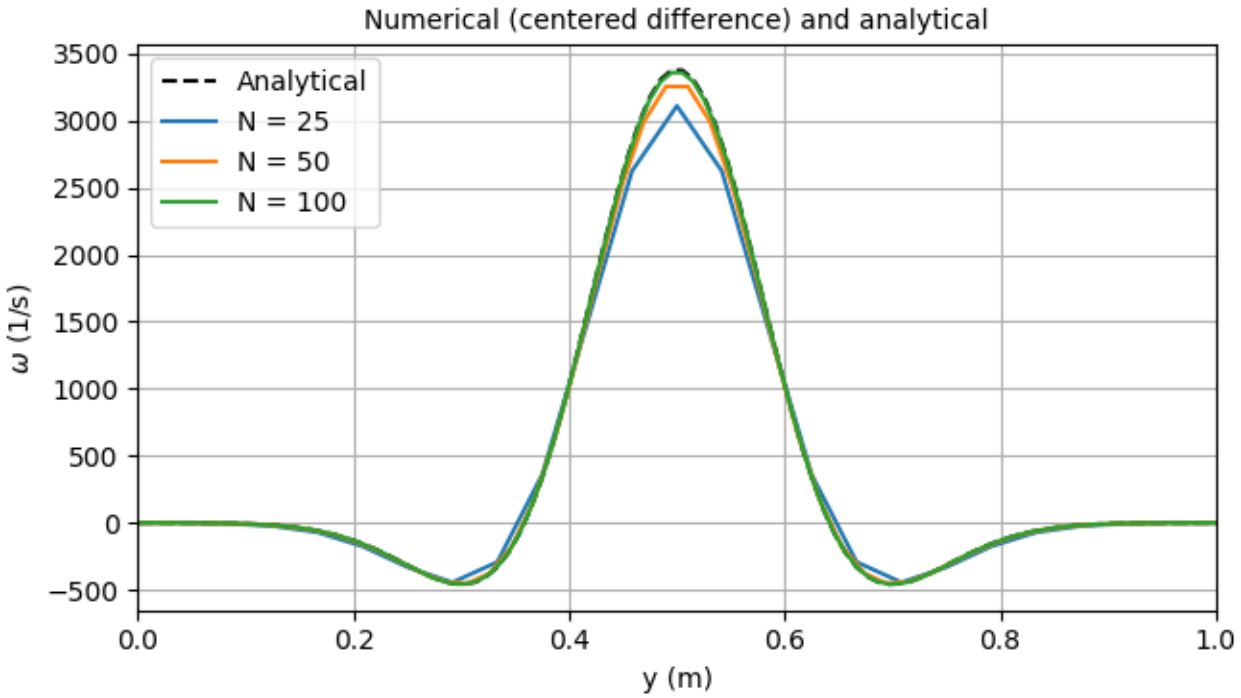

Figure 2: Profiles of analytical vorticity and numerical vorticity at $x = x_c$ (vortex center) are compared for the initial condition at a grid resolution of N = 100.

Figure 3: Comparison of error convergence rates, initial condition

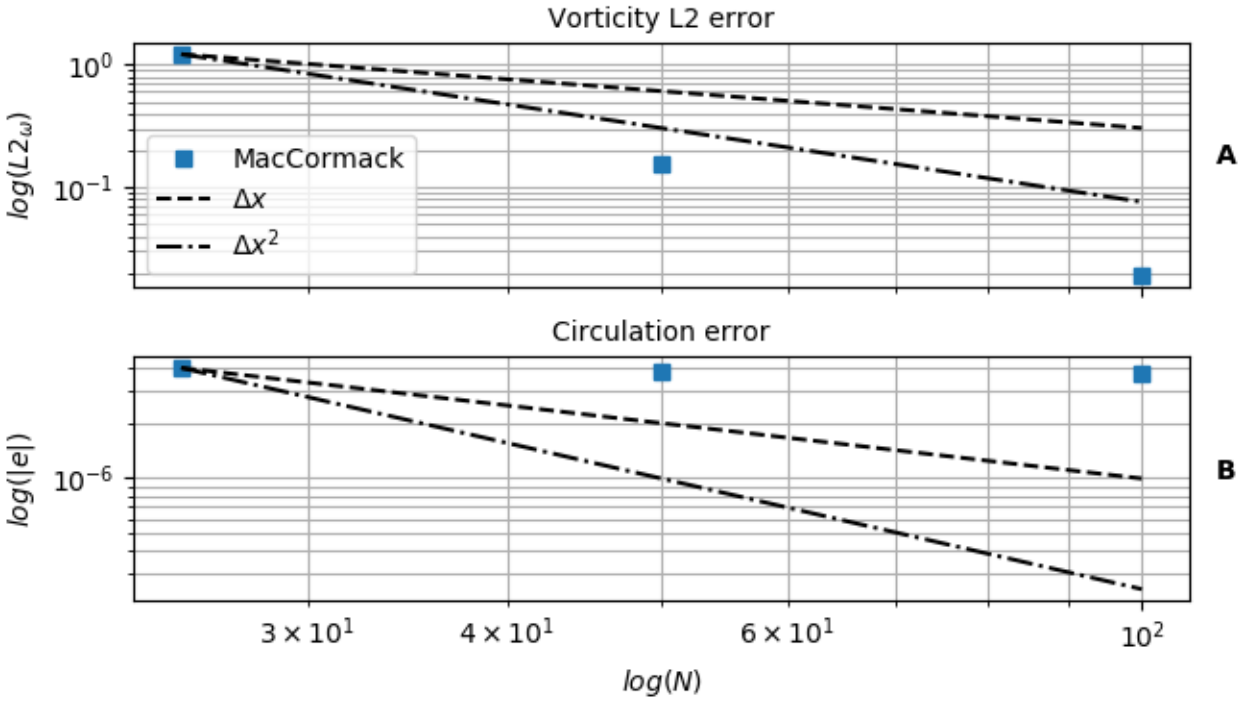

Figure 3: Convergence rates of vorticity L2 error and circulation error are compared to first-order and second-order convergence reference lines.

Table 3: Vorticity L2 error and normalized circulation error, initial condition

N	Vorticity L2 error	Circulation error
25	1.2149	3.9748e-06
50	0.1530	3.7852e-06
100	0.0191	3.7409e-06

Table 3: Vorticity L2 error and circulation error at the initial condition are tabulated for each grid resolution.

The 2D isentropic vortex is initialized using Eq. 7-12, with vorticity calculated analytically (see Appendix A for derivation) and numerically using central differencing. Vorticity contours and profiles at $x = x_c$ are generated for a grid resolution of N = 100 using both the analytical and numerical solutions. At this grid resolution, the vorticity contours are indistinguishable (Fig. 1a from the analytical solution, Fig. 1b for the numerical solution), and the vorticity profiles at $x = x_c$ (Fig. 2) of both solutions match almost exactly with peak vorticity of approximately 3360 1/s.

Plots of the convergence rate of the vorticity L2 error and circulation error (Fig. 3a and Fig. 3b respectively) are used to illustrate the errors in Table 3. The Δx^2 convergence rate reference line corresponds to the expected convergence rate of the second-order centered difference method used to calculate vorticity numerically. It is apparent that the convergence rate of the vorticity L2 error is close to second-order (which matches the central differencing used), while the circulation error appears to remain almost stationary. The vortex appears adequately resolved at a grid resolution of N = 50. The analytical and numerical vorticity contours and profiles match closely at this resolution (both resemble smooth concentric rings), vorticity L2 error is low at 0.1530, and circulation error is nearly zero at 3.7852e-06.

Part 2: Baseline cases

Figure 4: Comparison of ω contours, baseline case, N = 100

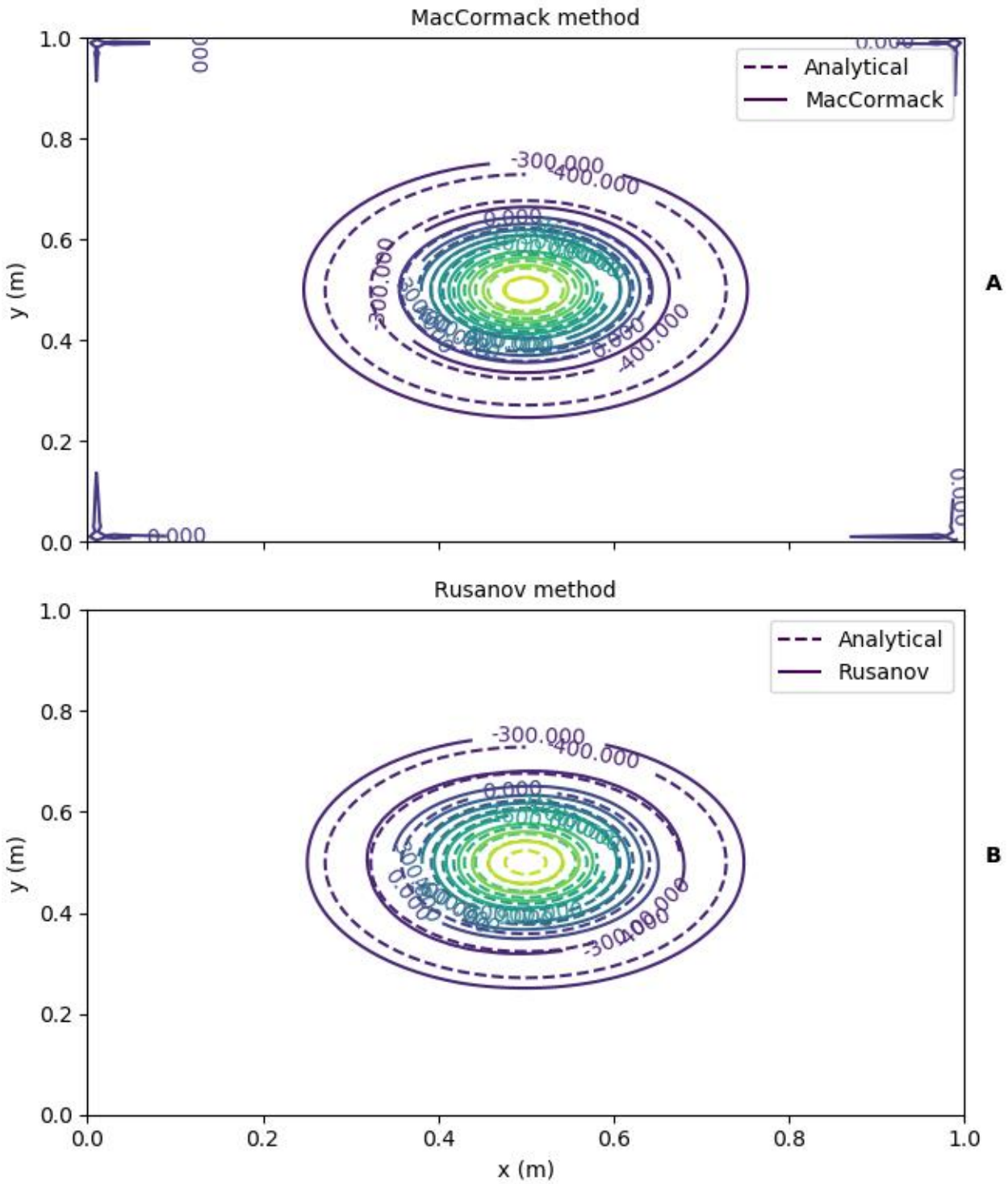

Figure 4: Contours of vorticity for the MacCormack method and Rusanov method applied to the baseline case are compared at a grid resolution of N = 100.

Figure 5: Comparison of u profiles at $x = x_c$, baseline case

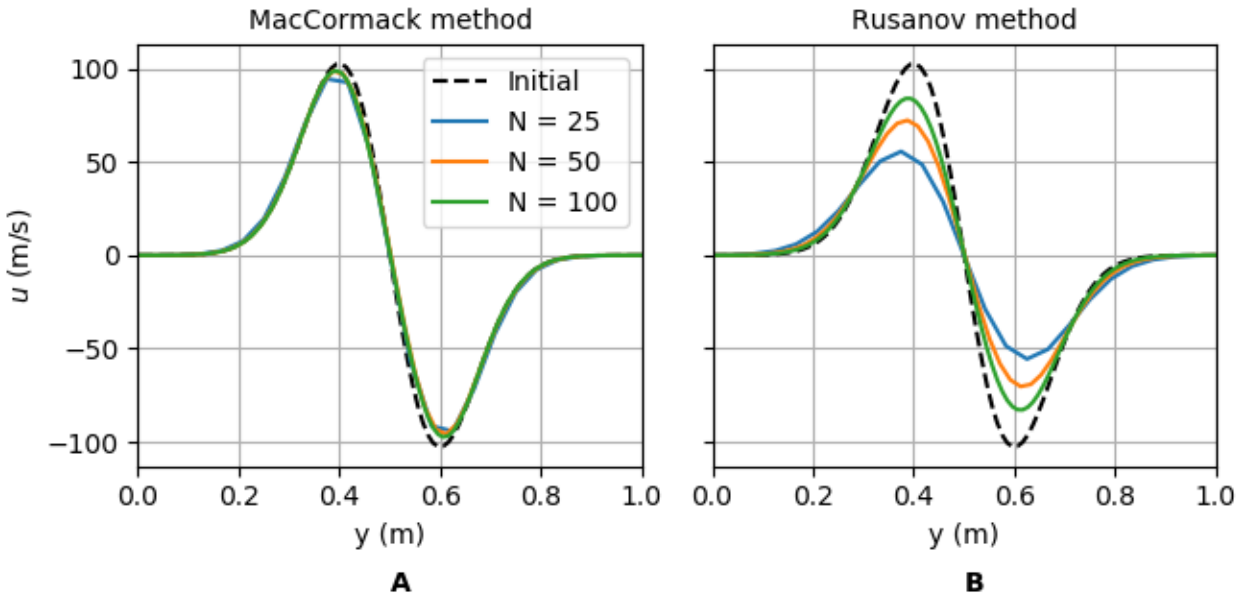

Figure 5: Profiles of u velocity at x = x_c (vortex center) for the MacCormack method and Rusanov method applied to the baseline case are compared to each other and to the analytical solution for N = 25, N = 50, and N = 100.

Figure 6: Comparison of ω profiles at $x = x_c$, baseline case

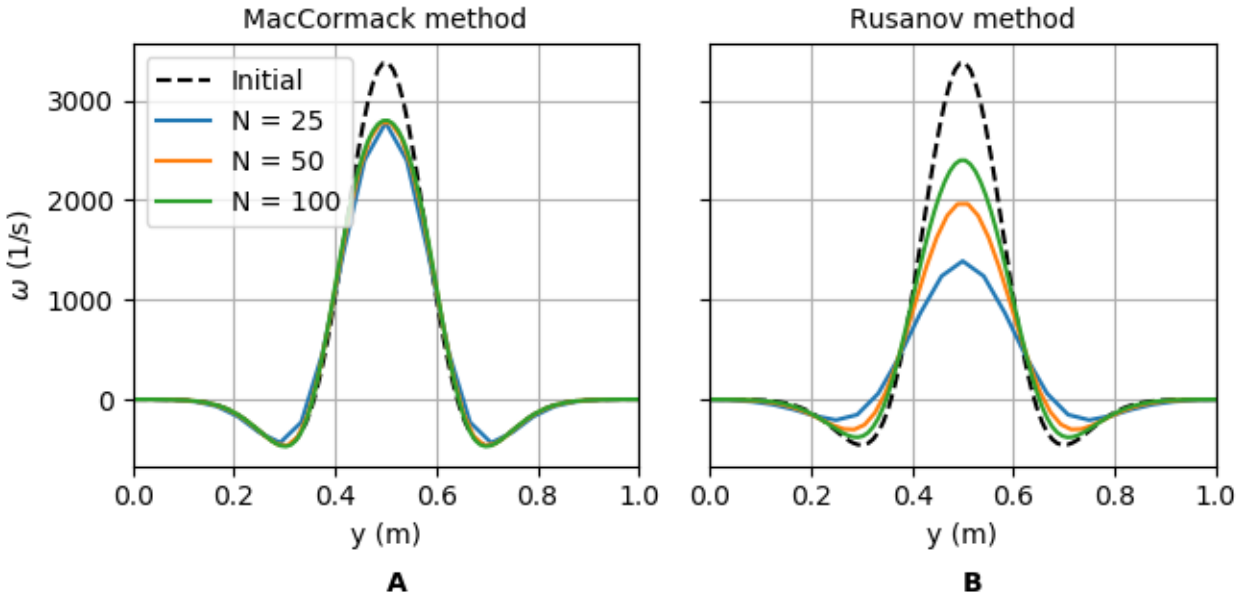

Figure 6: Profiles of vorticity at x = x_c (vortex center) for the MacCormack method and Rusanov method applied to the baseline case are compared to each other and to the analytical solution for N = 25, N = 50, and N = 100.

Figure 7: Comparison of error convergence rates, baseline case

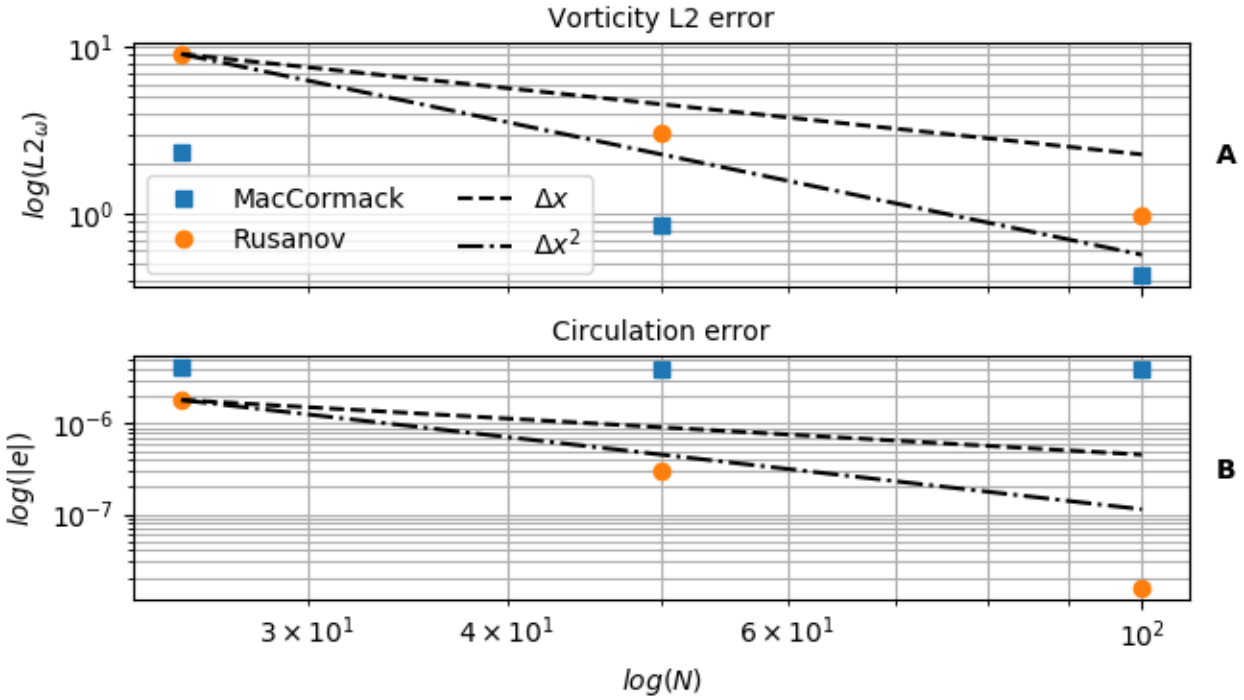

Figure 7: Convergence rates of vorticity L2 error and circulation error for the MacCormack method and Rusanov method applied to the baseline case are compared with each other and to first-order and second-order convergence reference lines.

Table 4: Vorticity L2 error and normalized circulation error, baseline case

	MacCormack method		Rusanov method	
N	**Vorticity L2 error**	**Circulation error**	**Vorticity L2 error**	**Circulation error**
25	2.3312	4.1637e-06	9.0860	1.8133e-06
50	0.8808	3.9575e-06	3.0338	2.9356e-07
100	0.4377	3.9121e-06	0.9686	1.4995e-08

Table 4: Vorticity L2 error and circulation error for the MacCormack method and Rusanov method applied to the baseline case are tabulated for each grid resolution.

The vorticity contours of the MacCormack method (Fig. 4a) and Rusanov method (Fig. 4b) differ substantially at a resolution of N = 100 for the baseline case. The concentric band structure seen for the initial condition (analytical solution) is preserved, with vorticity reaching a peak in the center of the vortex and becoming negative near the outer radius of the vortex at r = R_c. The MacCormack method reaches a peak vorticity of approximately 2800 1/s at a resolution of N = 100, while the more dissipative Rusanov method reaches a peak of just over 2400 1/s. This is reflected in the vorticity profile plots (Fig. 6), where the MacCormack method (Fig. 6a) solution slightly undershoots the initial condition at all grid resolutions (as there is still some dissipative error in this method). The vorticity profile for the Rusanov method (Fig. 6b) shows substantial peak suppression at all grid resolutions, although accuracy is improved with grid refinement (at N = 25, for example, peak vorticity is only approximately 1400 1/s).

The velocity profiles of both methods (Fig. 5) are also markedly different, with the MacCormack method (Fig. 5a) closely approximating the initial condition at all grid resolutions and the Rusanov method (Fig. 5b) again suffering from peak suppression due to dissipative error. Peak *u* velocity at N = 100 is approximately 100 m/s for the MacCormack method, while the Rusanov method undershoots at approximately 84 m/s.

Plots of the convergence rate of the vorticity L2 error and circulation error (Fig. 7a and Fig. 7b respectively) are used to illustrate the errors in Table 4. Both the MacCormack method and Rusanov method vorticity L2 errors appear to converge at a rate of approximately Δx^2 , but the MacCormack method is substantially more accurate at every grid resolution due to the larger dissipative error of the Rusanov method. The circulation error is close to zero for both methods and remains nearly constant with grid refinement for the MacCormack method, while it appears to decrease very rapidly (faster than second-order) for the Rusanov scheme.

Due to its lower dissipative error and more accurate results for vorticity and velocity, the MacCormack method proves to be a superior choice for this case.

Part 3: x-convection cases

Figure 8: Comparison of ω contours, x-convection case, N = 100

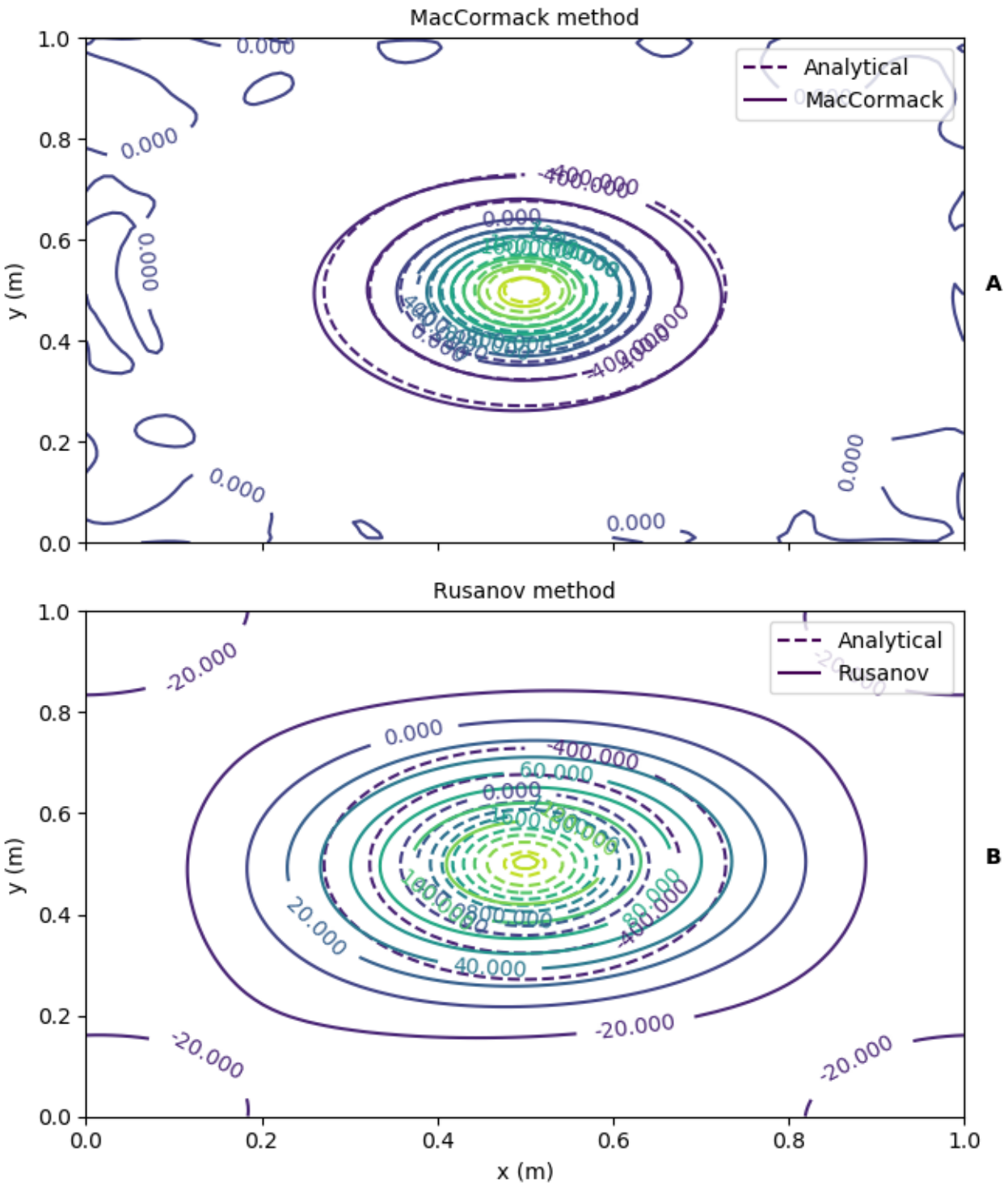

Figure 8: Contours of vorticity for the MacCormack method and Rusanov method applied to the x-convection case are compared at a grid resolution of N = 100.

Figure 9: Comparison of u profiles at $x = x_c$, x-convection case

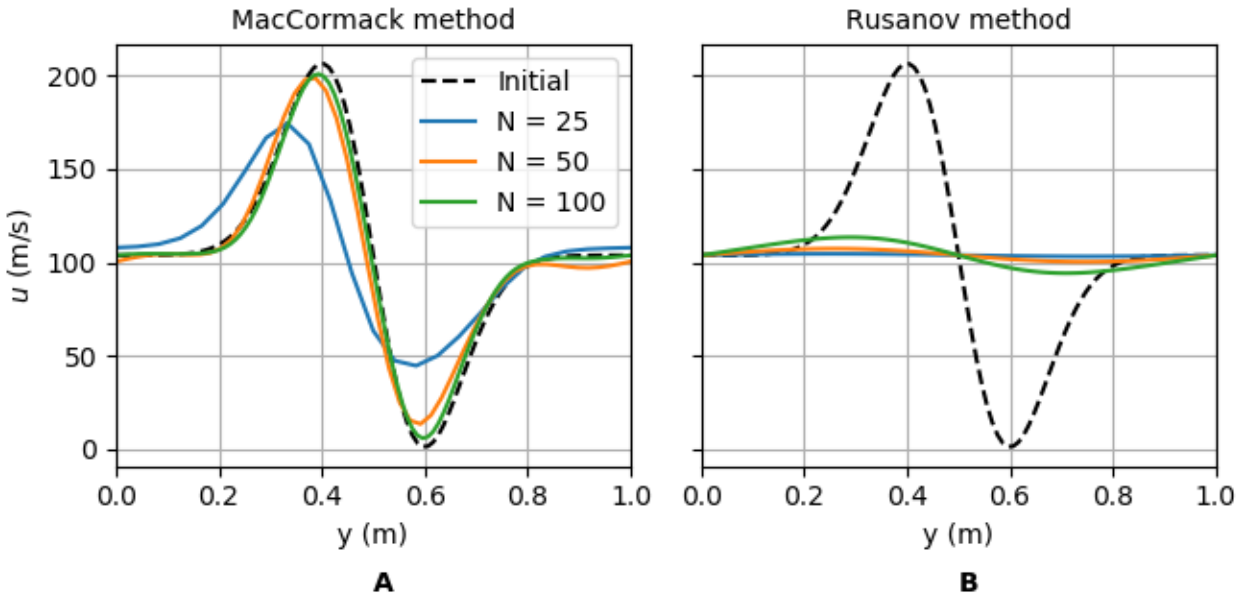

Figure 9: Profiles of u velocity at x = x_c (vortex center) for the MacCormack method and Rusanov method applied to the x-convection case are compared to each other and to the analytical solution for N = 25, N = 50, and N = 100.

Figure 10: Comparison of ω profiles at $x = x_c$, x-convection case

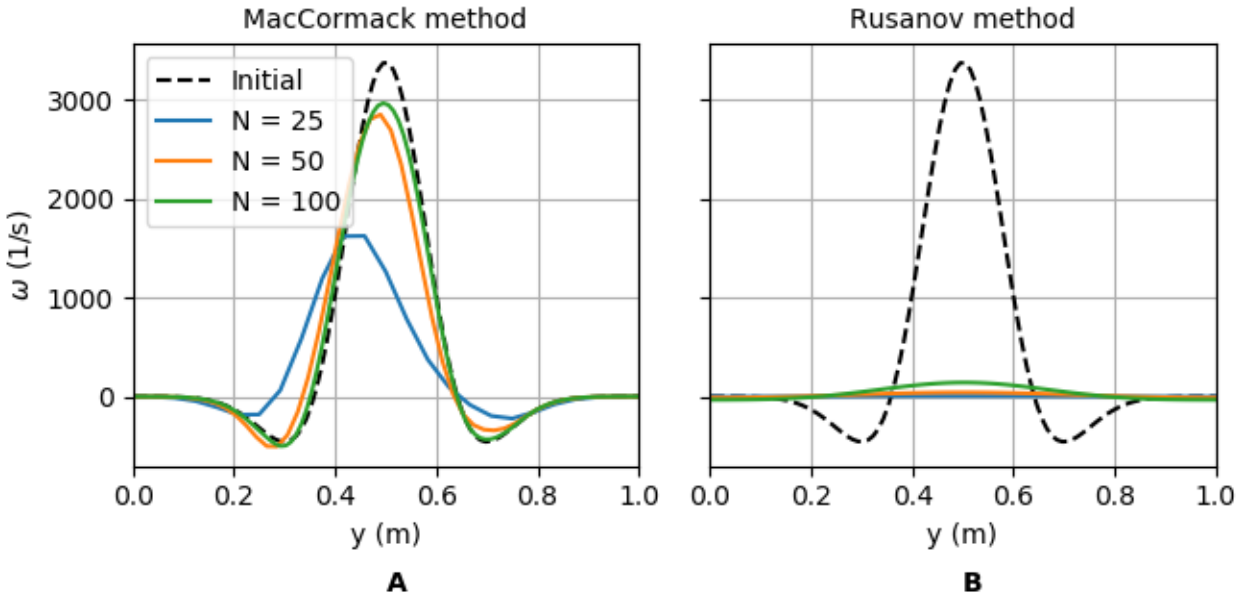

Figure 10: Profiles of vorticity at x = x_c (vortex center) for the MacCormack method and Rusanov method applied to the x-convection case are compared to each other and to the analytical solution for N = 25, N = 50, and N = 100.

Figure 11: Comparison of error convergence rates, x-convection case

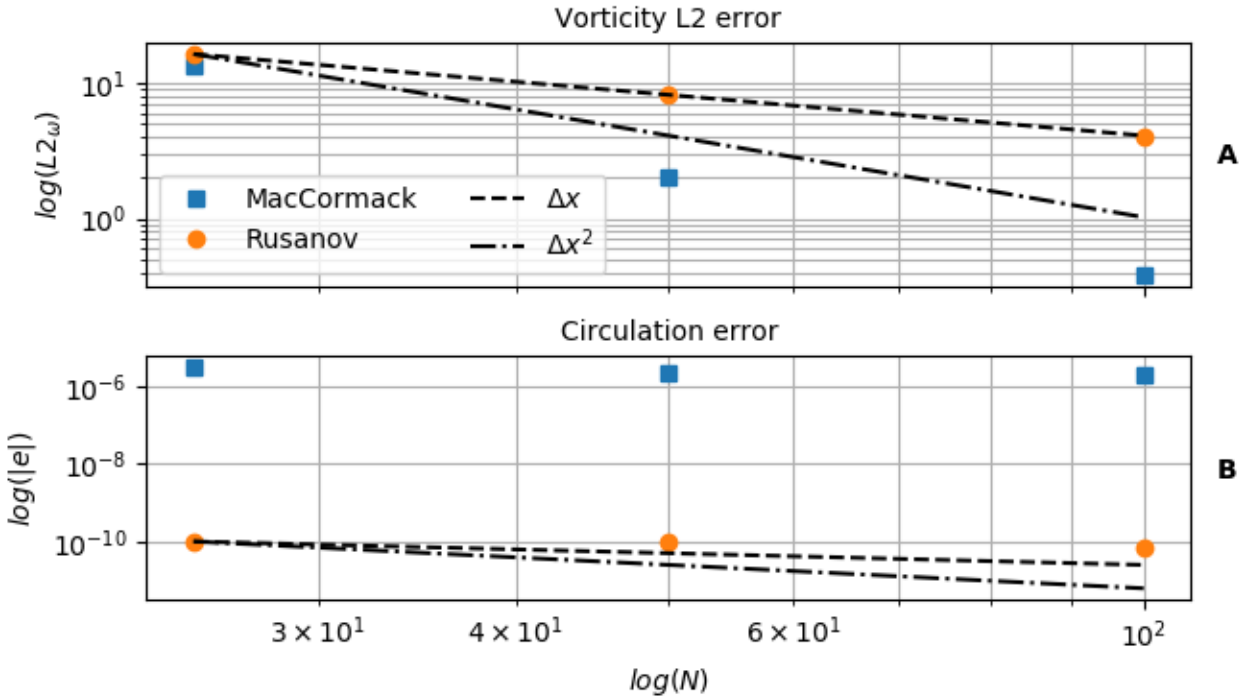

Figure 11: Convergence rates of vorticity L2 error and circulation error for the MacCormack method and Rusanov method applied to the x-convection case are compared with each other and to first-order and second-order convergence reference lines.

Table 5: Vorticity L2 error and normalized circulation error, x-convection case

	MacCormack method		Rusanov method	
N	**Vorticity L2 error**	**Circulation error**	**Vorticity L2 error**	**Circulation error**
25	13.3698	3.0430e-06	16.3088	-1.0275e-10
50	2.0189	2.2147e-06	8.2631	-1.0436e-10
100	0.3816	1.9724e-06	4.0592	-7.3821e-11

Table 5: Vorticity L2 error and circulation error for the MacCormack method and Rusanov method applied to the baseline case are tabulated for each grid resolution.

The vorticity contours of the MacCormack method (Fig. 8a) and Rusanov method (Fig. 8b) differ more significantly from both the analytical vorticity contour and each other for the x-convection case. Both methods display the expected behavior, with the vortex having moved through the right end of the computational domain at x = L, re-entered at the left end at x = 0, and returned to its original position at $x_c = L/2$. The MacCormack method suffers from some slight dissipative error (vorticity is slightly lowered to just over 3000 1/s), and oscillations in the solution are present due to dispersive error. These oscillations can be seen as contour lines in the far-field away from the vortex (where vorticity should be uniformly zero). The Rusanov method does not contain any oscillations but is extremely dissipative for this case, and has suppressed the peak vorticity to approximately 140 1/s. The vortex also appears much larger than before, due to the spreading effect of the dissipative error.

This is reflected in the vorticity profile plots (Fig. 10), where the MacCormack method (Fig. 10a) solution is similar to analytical solution at grid resolutions of N = 50 and N = 100 and has slight peak suppression. Results are notably worse at N = 25, where there is some slight misalignment between the analytical and numerical solution as well as significant peak suppression. The vorticity profile for the Rusanov method (Fig. 10b) shows severe peak suppression at all grid resolutions, although accuracy is improved with grid refinement. The velocity profiles of both methods (Fig. 9) are also markedly different from each other, with the MacCormack method (Fig. 9a) closely approximating the analytical solution at N = 50 and N = 100 while the Rusanov method (Fig. 9b) is so dissipative that the velocity profile is almost flat.

Plots of the convergence rate of the vorticity L2 error and circulation error (Fig. 11a and Fig. 11b respectively) are used to illustrate the errors in Table 5. The Rusanov method vorticity L2 error appears to converge at a rate of approximately Δx, while the MacCormack method is more accurate at every grid resolution and converges at a rate matching the Δx^2 reference line. The circulation error for both methods converges at a rate slower than first order – the convergence plot appears almost constant.

The MacCormack method is the superior choice for this case, due to its lower dissipative error and more accurate vorticity calculation.

Part 4: y-convection cases

Figure 12: Comparison of ω contours, y-convection case, N = 100

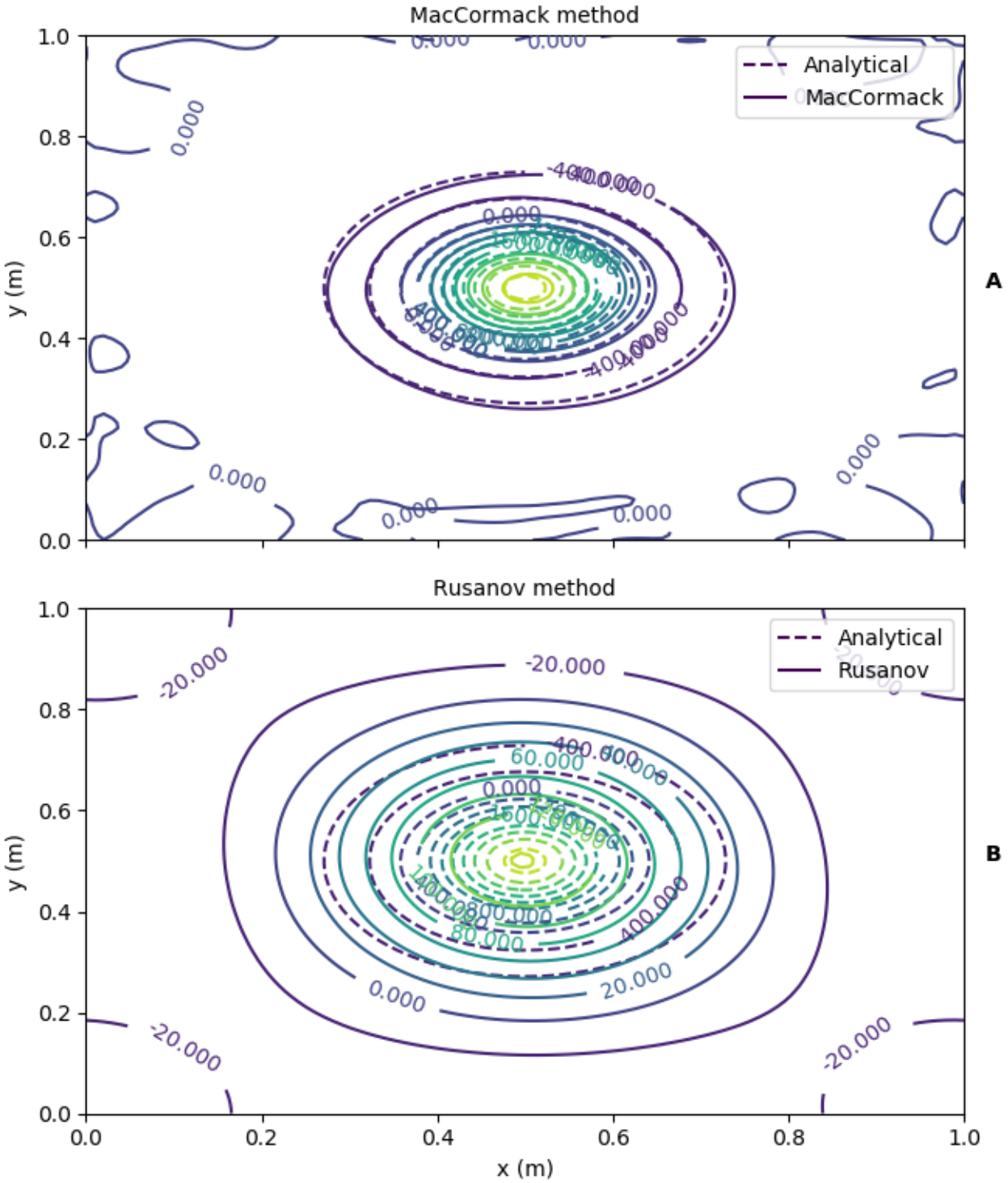

Figure 12: Contours of vorticity for the MacCormack method and Rusanov method applied to the y-convection case are compared at a grid resolution of N = 100.

Figure 13: Comparison of u profiles at $x = x_c$, y-convection case

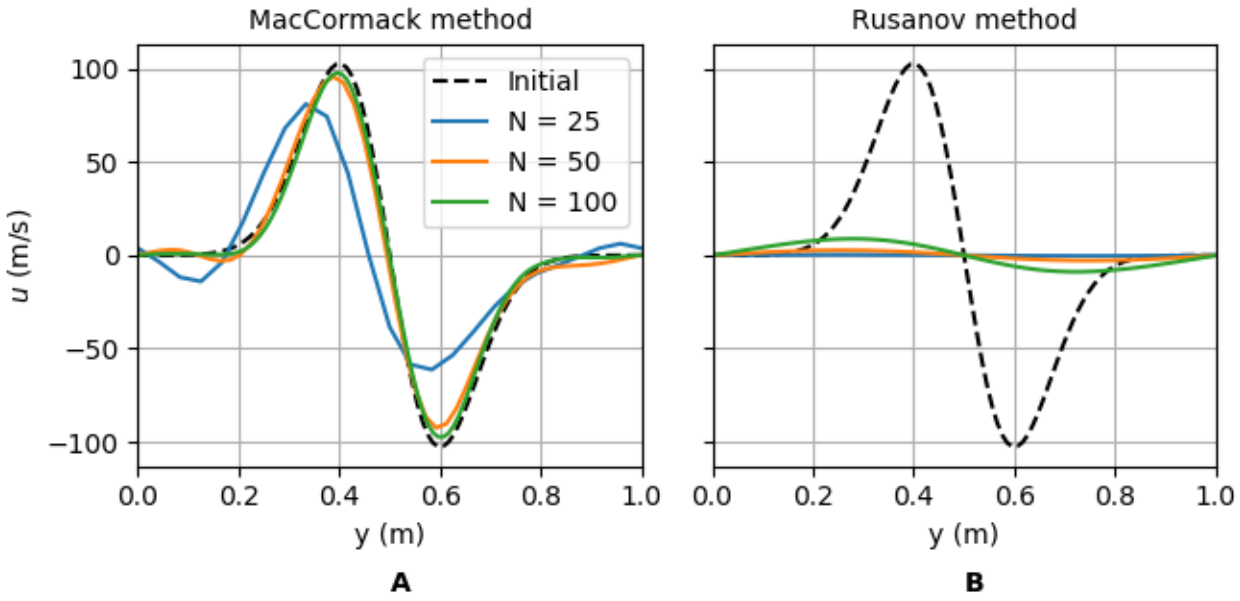

Figure 13: Profiles of u velocity at x = x_c (vortex center) for the MacCormack method and Rusanov method applied to the y-convection case are compared to each other and to the analytical solution for N = 25, N = 50, and N = 100.

Figure 14: Comparison of ω profiles at $x = x_c$, y-convection case

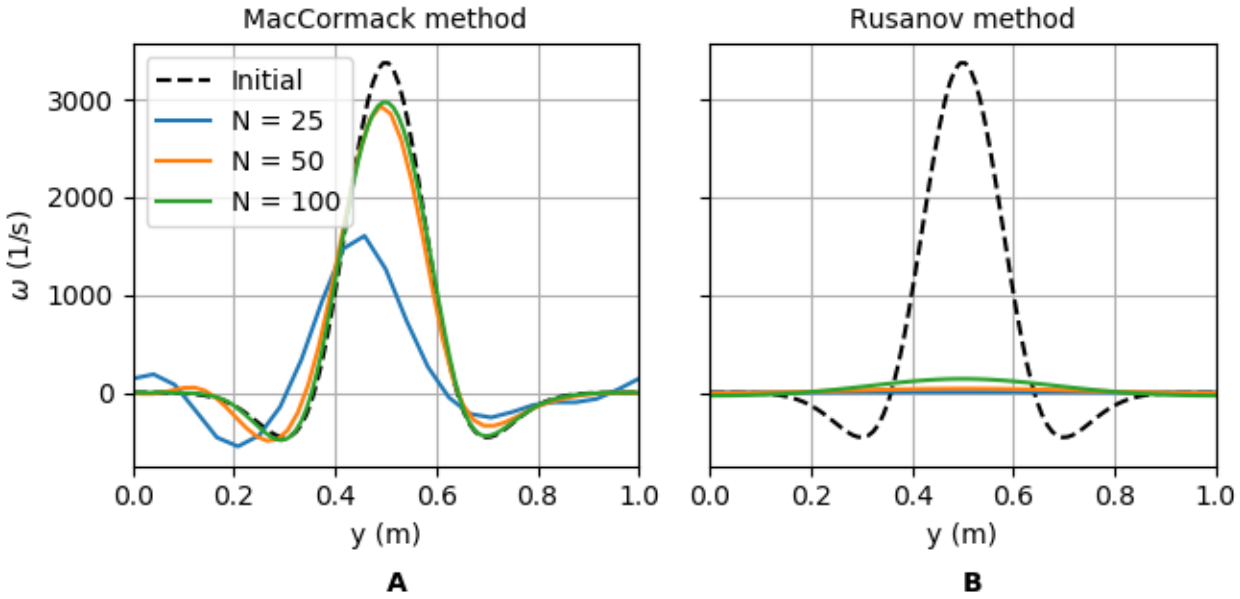

Figure 14: Profiles of vorticity at x = x_c (vortex center) for the MacCormack method and Rusanov method applied to the y-convection case are compared to each other and to the analytical solution for N = 25, N = 50, and N = 100.

Figure 15: Comparison of error convergence rates, y-convection case

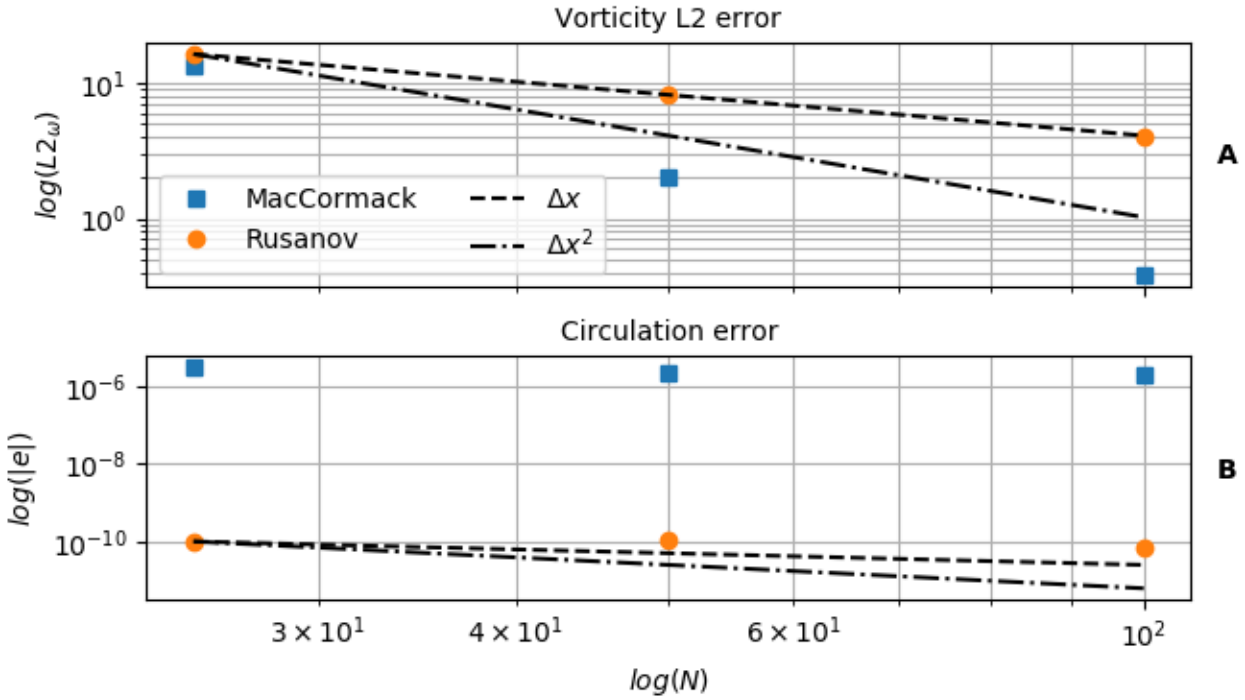

Figure 15: Convergence rates of vorticity L2 error and circulation error for the MacCormack method and Rusanov method applied to the y-convection case are compared with each other and to first-order and second-order convergence reference lines.

Table 6: Vorticity L2 error and normalized circulation error, y-convection case

	MacCormack method		Rusanov method	
N	**Vorticity L2 error**	**Circulation error**	**Vorticity L2 error**	**Circulation error**
25	13.3742	3.0426e-06	16.3088	-1.0260e-10
50	2.0187	2.2148e-06	8.2631	-1.0943e-10
100	0.3816	1.9724e-06	4.0592	-7.3885e-11

Table 6: Vorticity L2 error and circulation error for the MacCormack method and Rusanov method applied to the y-convection case are tabulated for each grid resolution.

The vorticity contours of the MacCormack method (Fig. 12a) and Rusanov method (Fig. 12b) for the y-convection case appear similar to those of the x-convection case (Fig. 8). The vortex has moved through the upper end of the computational domain at y = L, re-entered at the bottom at y = 0, and returned to its original position at y_c = L/2. The MacCormack method solution again contains oscillations due to dispersive error, as well as slight peak suppression (peak vorticity is approximately 3000 1/s) due to dissipation. The Rusanov method solution is again highly dissipative and has resulted in a larger radius vortex with substantially lower peak vorticity (approximately 140 1/s). The results for the x-convection and y-convection cases are almost identical, as expected for this symmetrical problem and symmetrical solvers.

The vorticity profile of the MacCormack method (Fig. 14a) solution is similar to analytical solution at grid resolutions of N = 50 and N = 100 and has slight peak suppression. The vorticity profile for the Rusanov method (Fig. 14b) again shows severe peak suppression at all grid resolutions. The

u velocity profile of the MacCormack method (Fig. 13a) closely approximates the analytical solution at N = 50 and N = 100, while the Rusanov method solution (Fig. 13b) is again so dissipative that the velocity profile is almost flat.

Plots of the convergence rate of the vorticity L2 error and circulation error (Fig. 15a and Fig. 15b respectively) are used to illustrate the errors in Table 6, and appear identical to those for the x-convection case (as expected). The Rusanov method vorticity L2 error appears to converge at a rate of approximately Δx, while the MacCormack method is again more accurate at every grid resolution and converges at a rate matching or slightly exceeding that of the Δx^2 reference line. Circulation error is near zero for both methods at all grid resolutions, and for the Rusanov method appears to increase slightly from N = 25 to N = 50 before decreasing from N = 50 to N = 100. The MacCormack circulation error appears to converge at a rate substantially slower than first-order.

The MacCormack method is the superior choice for this case, due to its lower dissipative error and more accurate vorticity calculation.

Part 5: Diagonal convection cases

Figure 16: Comparison of ω contours, diagonal convection case, N = 100

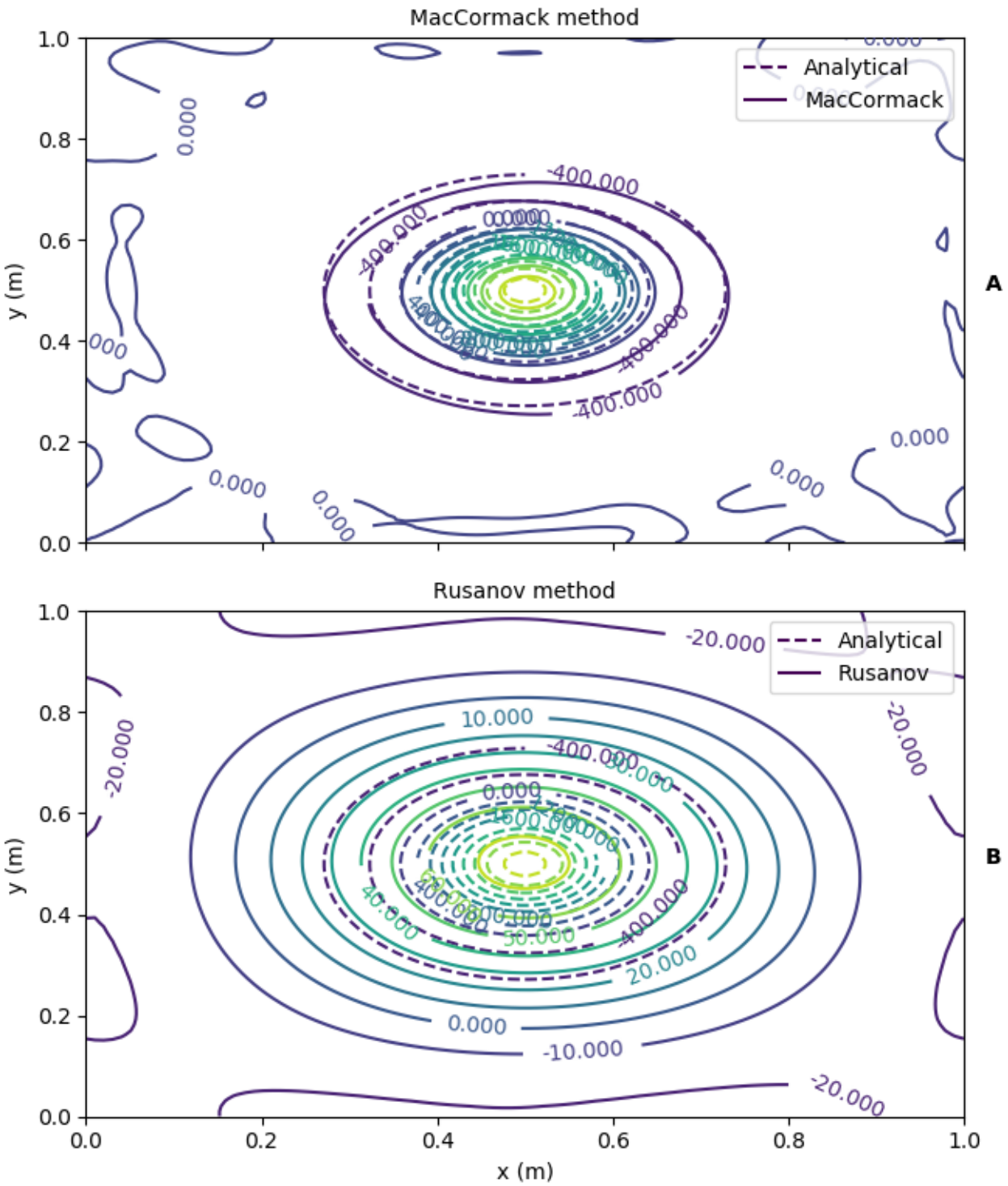

Figure 16: Contours of vorticity for the MacCormack method and Rusanov method applied to the diagonal convection case are compared at a grid resolution of N = 100.

Figure 17: Comparison of u profiles at $x = x_c$, diagonal convection case

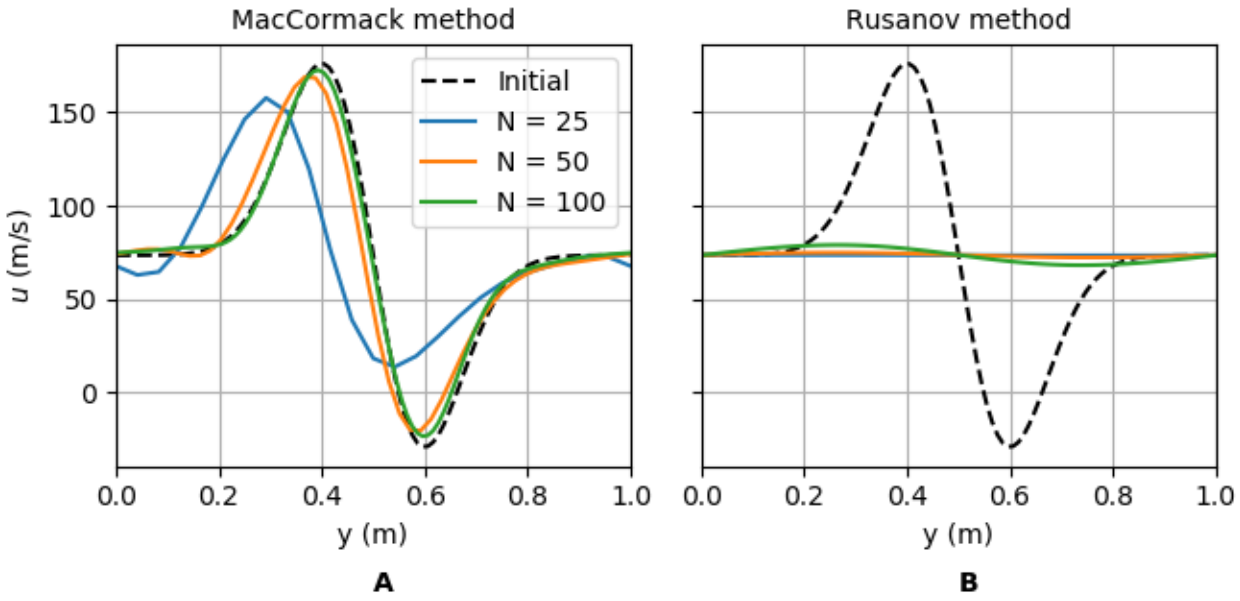

Figure 17: Profiles of u velocity at x = x_c (vortex center) for the MacCormack method and Rusanov method applied to the diagonal convection case are compared to each other and to the analytical solution for N = 25, N = 50, and N = 100.

Figure 18: Comparison of ω profiles at $x = x_c$, diagonal convection case

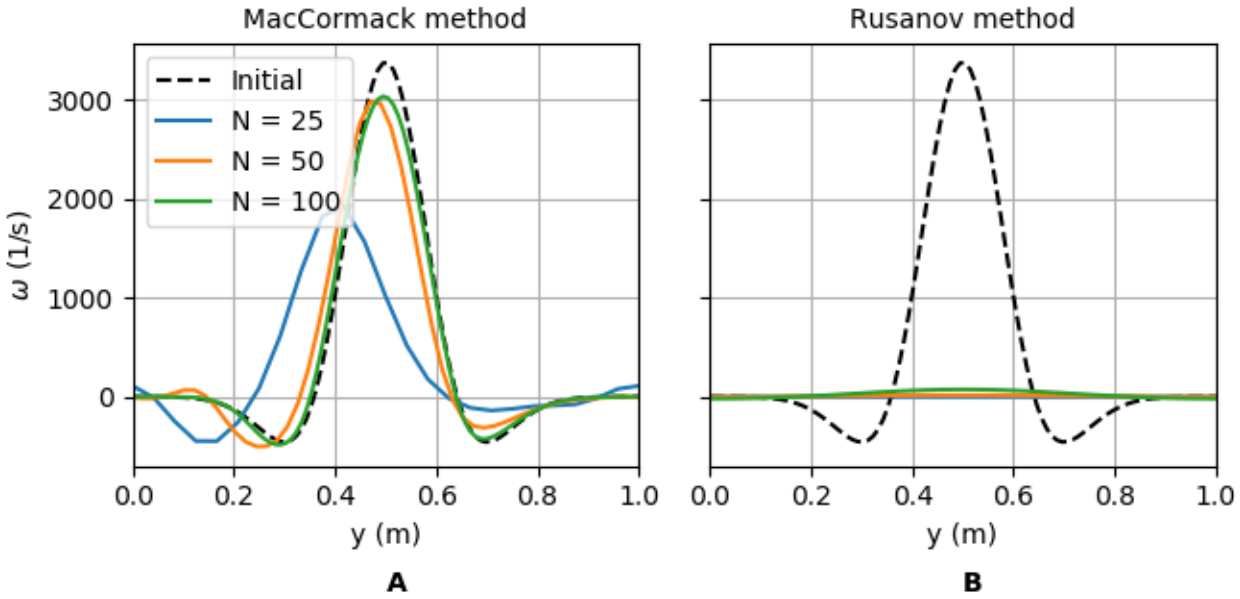

Figure 18: Profiles of vorticity at x = x_c (vortex center) for the MacCormack method and Rusanov method applied to the diagonal convection case are compared to each other and to the analytical solution for N = 25, N = 50, and N = 100.

Figure 19: Comparison of error convergence rates, diagonal convection case

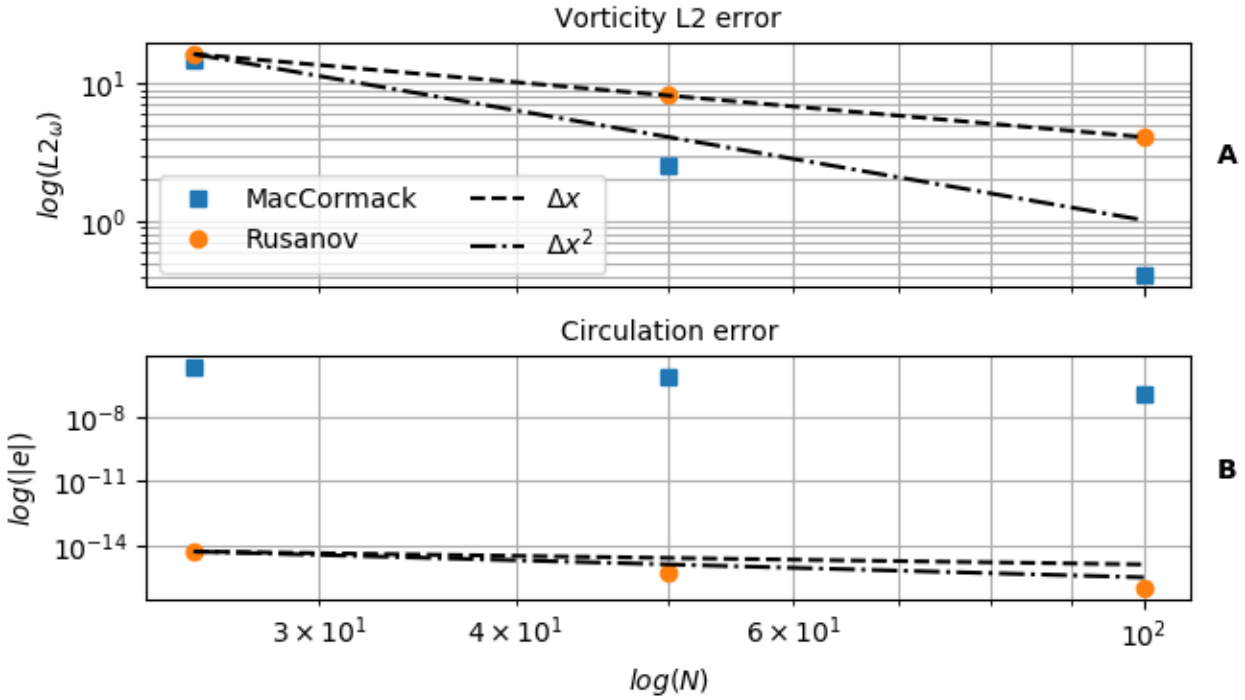

Figure 19: Convergence rates of vorticity L2 error and circulation error for the MacCormack method and Rusanov method applied to the diagonal convection case are compared with each other and to first-order and second-order convergence reference lines.

Table 7: Vorticity L2 error and normalized circulation error, diagonal convection case

	MacCormack method		Rusanov method	
N	**Vorticity L2 error**	**Circulation error**	**Vorticity L2 error**	**Circulation error**
25	14.7092	2.0466e-06	16.3255	-5.3945e-15
50	2.5610	7.0920e-07	8.3077	-5.1344e-16
100	0.4078	1.2024e-07	4.1401	1.0359e-16

Table 7: Vorticity L2 error and circulation error for the MacCormack method and Rusanov method applied to the diagonal convection case are tabulated for each grid resolution.

The diagonal convection case differs slightly from the x-convection and y-convection cases, primarily due to the longer overall distance travelled by the vortex (1.414L rather than L). The misalignment of the vortex convection direction with the computational grid may also cause the results to differ slightly. The vorticity contour plots for the MacCormack method (Fig. 16a) and Rusanov method (Fig. 16b) both show more dissipation than for the other convection cases or the baseline case, with peak vorticity suppressed to approximately 2600 1/s for the MacCormack solution and 80 1/s for the Rusanov solution.

The vorticity profile plots reflect this, with the MacCormack method (Fig. 18a) undershooting much more substantially at all grid resolutions and the Rusanov method (Fig. 18b) appearing almost totally flat. The vorticity profiles for the MacCormack method appear misaligned with the initial condition, showing that the vortex has become slightly distorted or has not returned

exactly to its original position. The u velocity profiles of the MacCormack method (Fig. 17a) also show some peak suppression and misalignment with the initial condition, though the results appear closer to the analytical solution than those for vorticity. The Rusanov method velocity profiles (Fig. 17b) have been suppressed severely due to dissipation.

Plots of the convergence rate of the vorticity L2 error and circulation error (Fig. 19a and Fig. 19b respectively) are used to illustrate the errors in Table 7, and closely resemble those for x-convection and y-convection in terms of convergence rate (slope). However, the magnitude of the errors (Table 7) differs – this is expected, due to the additional length traveled by the vortex and the misalignment of the convection with the computational grid. The Rusanov method vorticity L2 error appears to converge at a rate of approximately Δx, while the MacCormack method is more accurate at every grid resolution and converges at the same rate as the Δx^2 reference line. Circulation error is near zero and appears to converge faster than second-order for both methods, but the Rusanov method circulation error is lower at all grid resolutions.

The MacCormack method is the superior choice for this case, due to its lower dissipative error and more accurate vorticity calculation.

Part 6: Compressibility cases

Figure 20: Comparison of ω contours, $M_{a,c} = 0.3$, N = 100

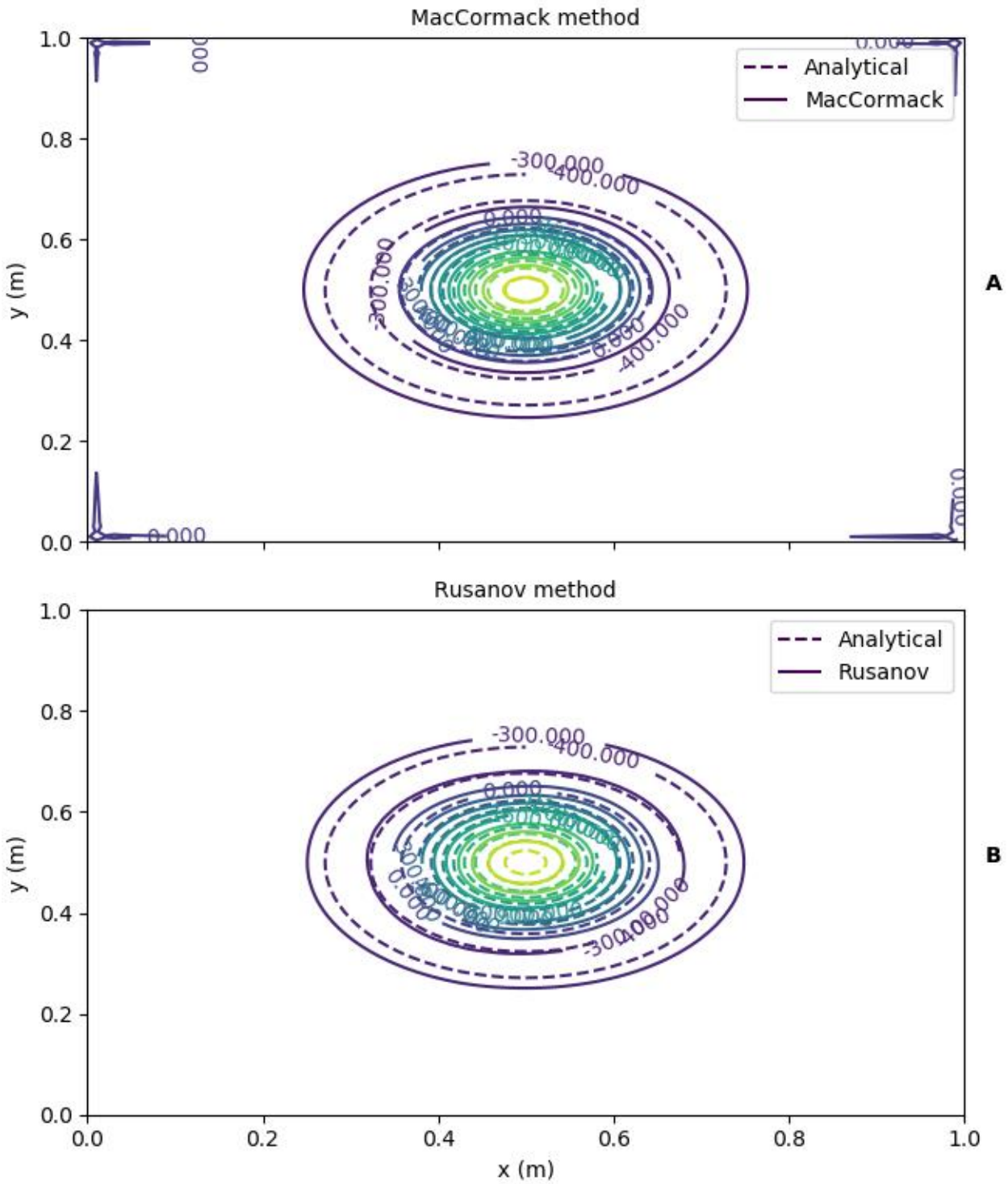

Figure 20: Vorticity contours for the MacCormack and Rusanov method are compared to the analytical solution for a vortex Mach number of 0.3 at a grid resolution of N = 100 (note that this figure is identical to Fig. 4, and is repeated here for convenience).

Figure 21: Comparison of ω contours, $M_{a,c} = 1.5$, N = 100

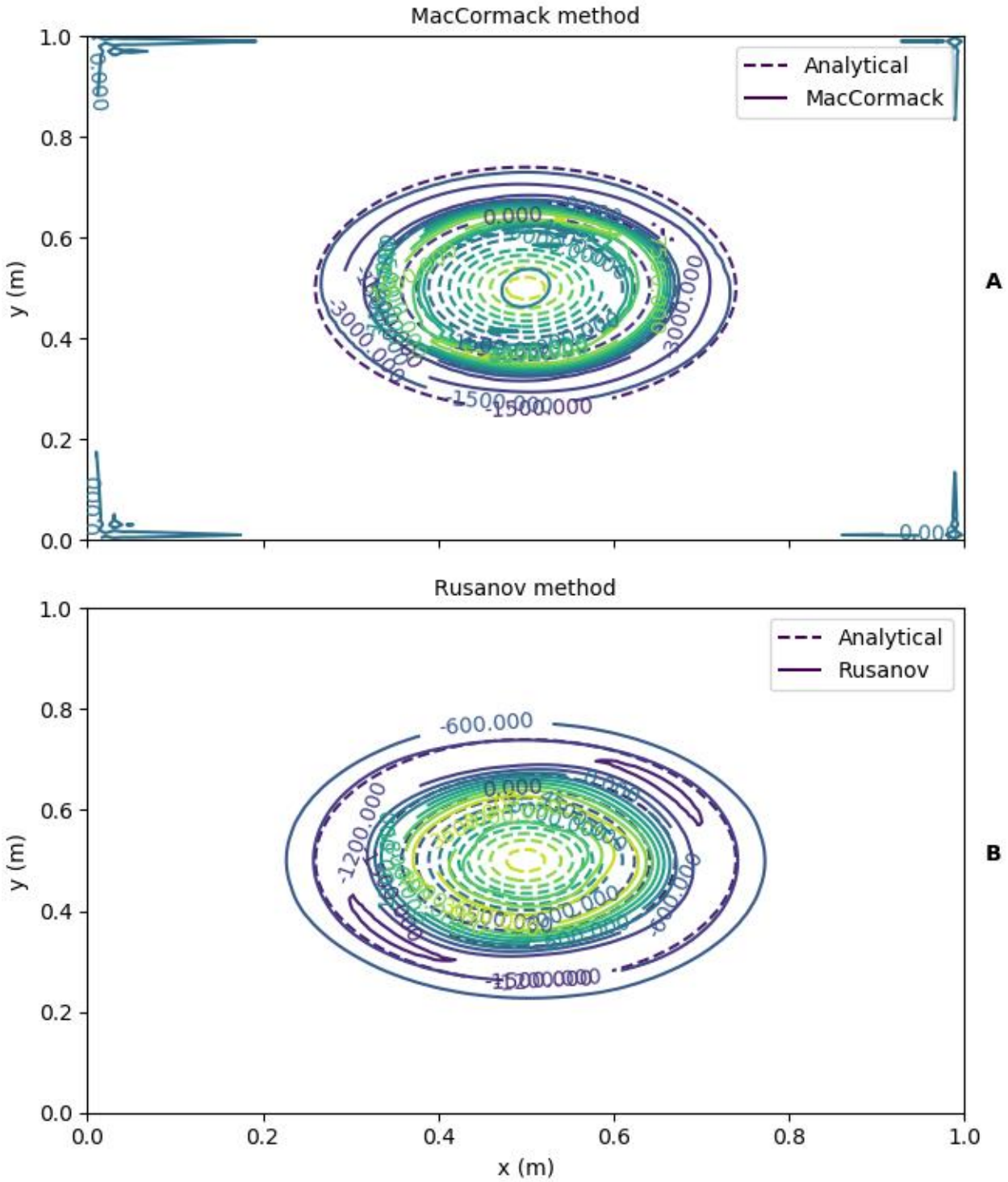

Figure 21: Vorticity contours for the MacCormack and Rusanov method are compared to the analytical solution for a vortex Mach number of 1.5 at a grid resolution of N = 100.

Figure 22: Comparison of shadowgraph, $M_{a,c} = 0.3$, N = 100

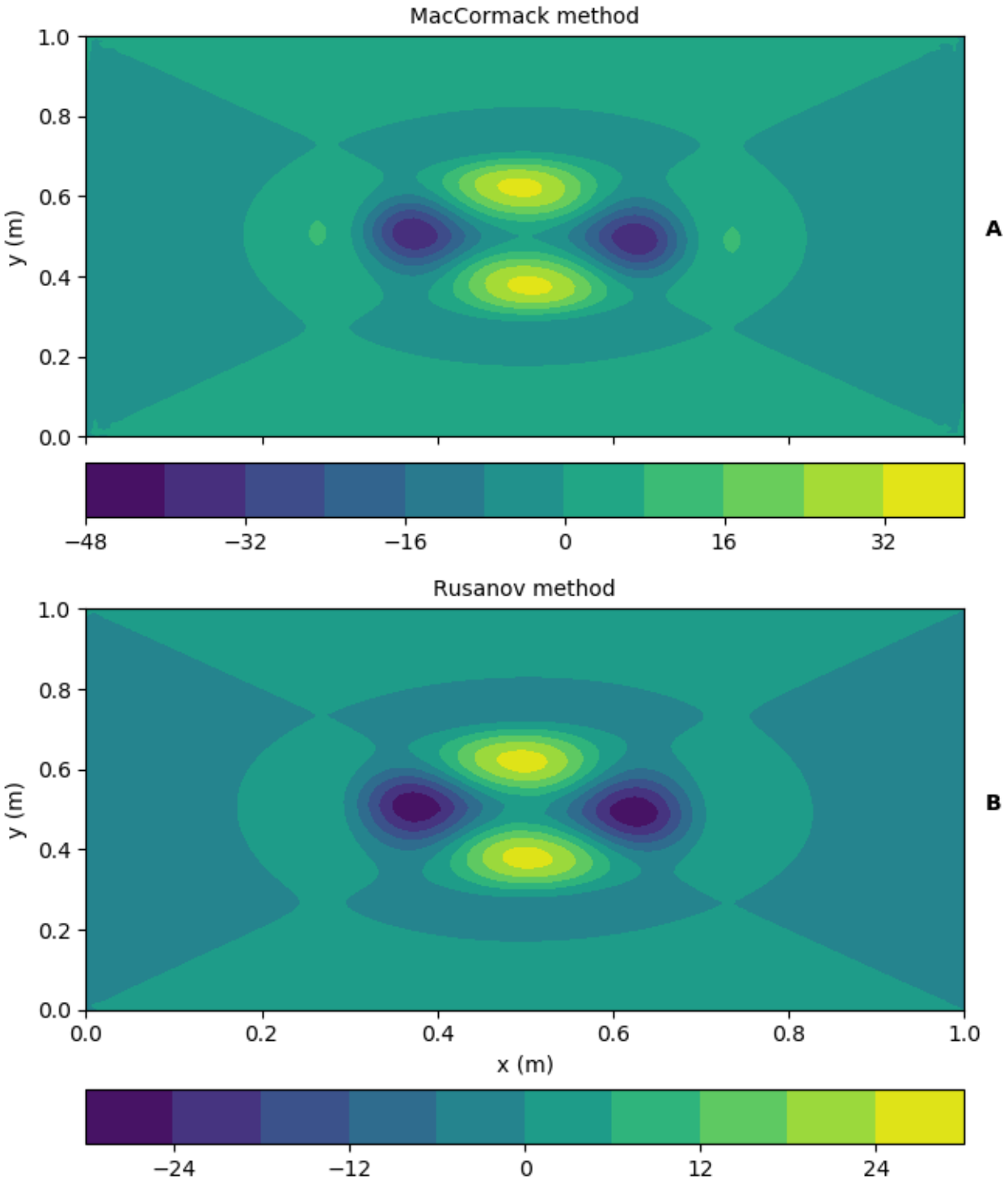

Figure 22: Approximate shadowgraph metrics for the MacCormack and Rusanov method are compared for a vortex Mach number of 0.3 at a grid resolution of N = 100.

Figure 23: Comparison of shadowgraph, $M_{a,c} = 1.5$, N = 100

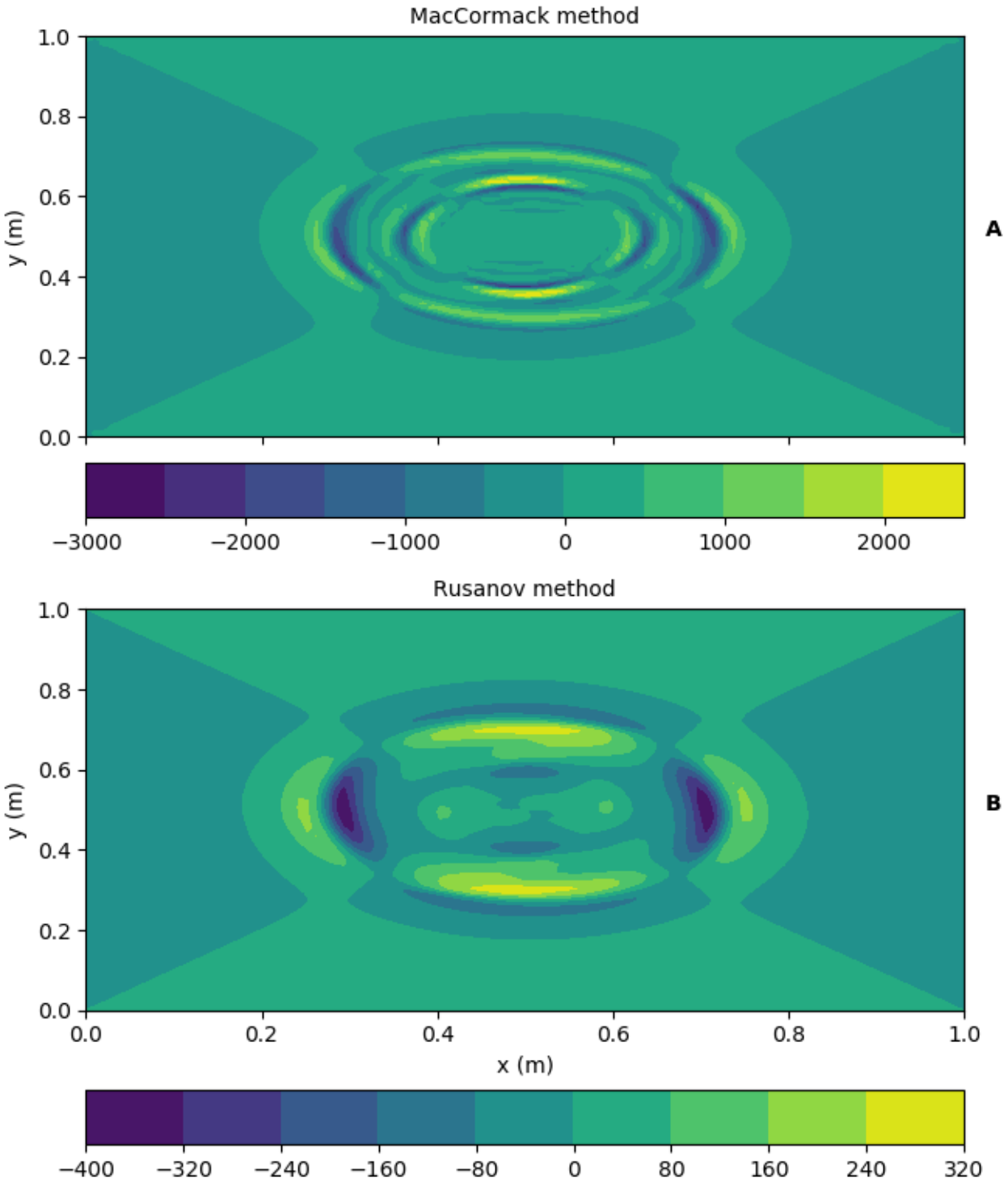

Figure 23: Approximate shadowgraph metrics for the MacCormack and Rusanov method are compared for a vortex Mach number of 1.5 at a grid resolution of N = 100.

Figure 24: Comparison of compressibility, $M_{a,c} = 0.3$, N = 100

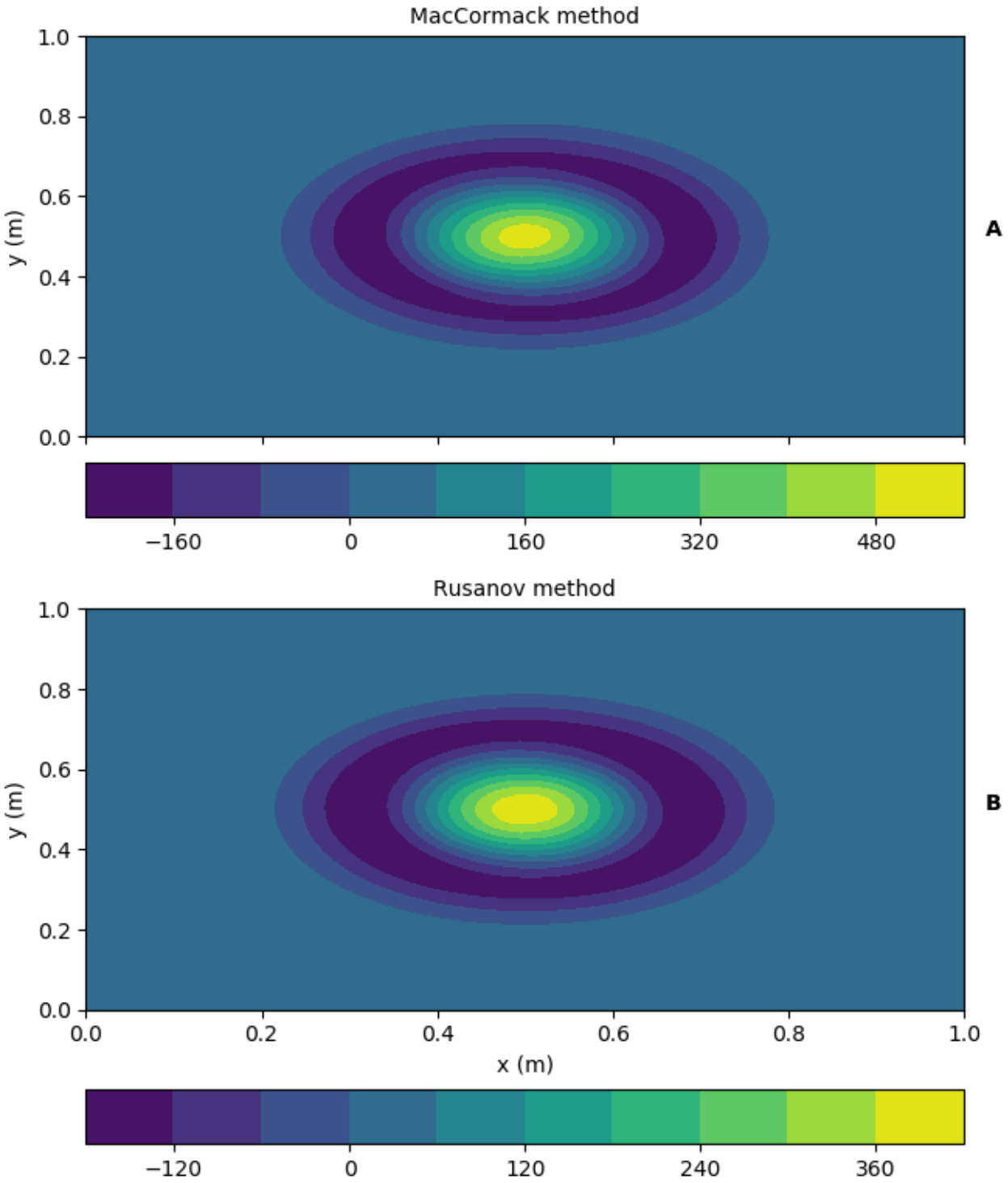

Figure 24: Compressibility metrics for the MacCormack and Rusanov method are compared for a vortex Mach number of 0.3 at a grid resolution of N = 100.

Figure 25: Comparison of compressibility, $M_{a,c} = 1.5$, N = 100

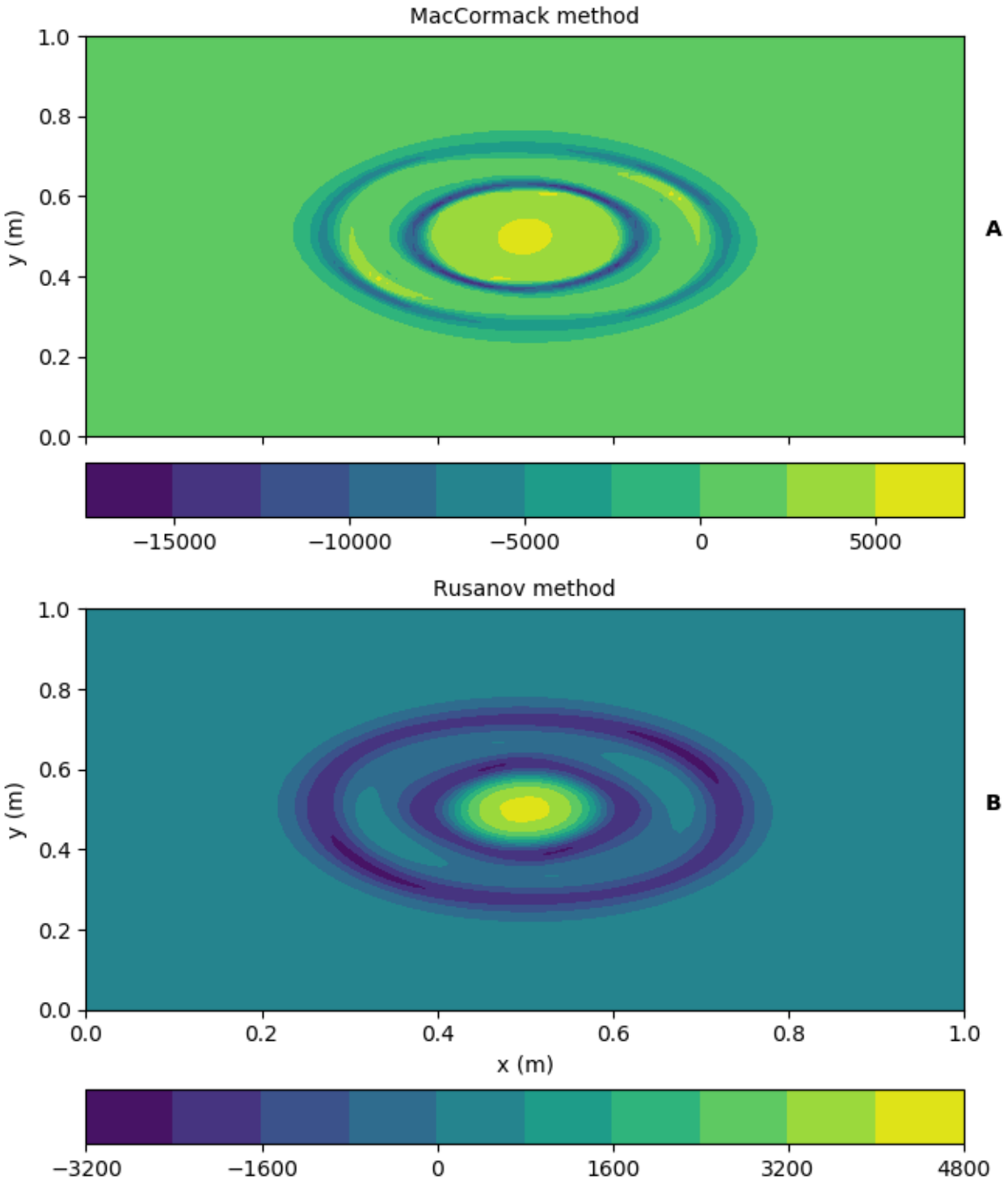

Figure 25: Compressibility metrics for the MacCormack and Rusanov method are compared for a vortex Mach number of 1.5 at a grid resolution of N = 100.

The vorticity contours of the MacCormack method (Fig. 20a) and Rusanov method (Fig. 20b) at a vortex Mach number of 0.3 are used as a baseline, and have been explored in detail in Part 2 of this report. These contours are reproduced here for convenience. Compressibility effects must be taken into consideration at Mach numbers past 0.3 – this is illustrated by Fig. 21, which shows the vorticity contour for a vortex Mach number of 1.5. The MacCormack method vorticity contour (Fig. 21a) shows that the vortex structure has changed substantially from the initial condition. The radius has decreased and vorticity has been concentrated towards the mid-radius of the vortex, forming a ring structure of high vorticity. The core region of the vortex appears to have expanded substantially, and now has an almost constant vorticity of approximately 1500 1/s rather than the peak observed in Fig. 20. Maximum positive vorticity is increased to approximately 9000 1/s in the high vorticity ring region, while maximum negative vorticity is increased to approximately -4000 1/s in another ring region just outside of the high vorticity ring. The Rusanov method solution undergoes similar changes with increased Mach number. The dissipative spreading of the vortex has been somewhat counteracted by compressibility effects, and the vorticity concentrates in a ring at the mid-radius of the vortex. Maximum positive vorticity is approximately 4000 1/s, while maximum negative vorticity is approximately -1800 1/s.

The approximate shadowgraph metric is calculated as $\nabla^2\rho$ (units of kg/m^5) using centered differences. The incompressible case (Fig. 22) appears qualitatively similar for both the MacCormack method (Fig. 22a) and Rusanov method (Fig. 22b), but density gradients are higher for the MacCormack method. This is expected – the Rusanov method is highly dissipative, which acts to suppress peaks and smooth gradients in the solution. The highest magnitude density gradients for both methods occur in regions at the top, bottom, left, and right of the vortex. Fig. 23 shows the shadowgraph metric for a Mach number of 1.5. Similarly to the vorticity contour, the gradients are much more concentrated in a ring that occurs at the mid-radius and increase dramatically in magnitude. The MacCormack method shows values as high as 3000 (Fig. 23a), while the dissipative Rusanov method shows values of only 320 (Fig. 23b).

Compressibility is measured using the divergence of the velocity field (or dilatation), $\nabla \cdot V$ (units of 1/s). The incompressible case (Fig. 24) shows that divergence is largest in the center of the vortex, where it is positive (meaning that this region is acting as a source). The MacCormack method (Fig. 24a) and Rusanov method (Fig. 24b) appear qualitatively similar, with the Rusanov method having lower values due to dissipation. The compressible case (Fig. 25) divergence forms a mid-radius ring, as observed previously, but the values in this region are closest to zero. The MacCormack method (Fig. 25a) shows large peaks of positive divergence that occur in the vortex center and negative peaks just outside and inside of the near-zero divergence ring, as does the Rusanov method (Fig. 25b). These large peaks of negative dilatation result in an increase in vorticity. These peaks are again smaller in the Rusanov method due to dissipation.

These shadowgraph and compressibility metrics imply that at higher Mach numbers, the vortex core tends to expand such that vorticity is concentrated in a ring around the core. The MacCormack method appears to suffer much more severely from this concentration effect, most likely due to the dispersive error in the scheme. The pitfalls of the MacCormack method when it comes to solutions with large gradients or discontinuities has been discussed in previous studies,

and it is likely that this is playing a role for this case. While for the other cases the MacCormack method has proven superior to the Rusanov method, the tendency of the MacCormack method to overshoot the analytical solution near large discontinuities may not make it ideal for solving problems involving vortices at high Mach numbers where compressibility effects concentrate the vorticity into a narrow ring structure.

Conclusion and Summary

The MacCormack method and Rusanov method extended to two-dimensions have been applied to several cases concerning an isentropic two-dimensional vortex. The MacCormack method proves to be the superior method for the baseline case and all convective cases, primarily due to the high amount of dissipation present in the Rusanov method which causes suppression of peaks and smoothing of gradients. At low vortex Mach numbers, gradients are sufficiently low that the MacCormack method does not suffer from the overshooting behavior observed in previous studies, and the method shows only minor oscillations in the far-field for the convective cases. The vorticity contours for all of these cases closely resemble the analytical solution. The higher Mach number compressible case shows that vorticity is concentrated in a ring at the mid-radius of the vortex, creating regions with high gradients. The dispersive error present in the MacCormack scheme tends to overshoot the solution in these high gradient regions, as evidenced by extremely high peaks in the vorticity contours (Fig. 21), approximate shadowgraph contours (Fig. 23), and velocity divergence contours (Fig. 25). At higher Mach numbers, it is inadvisable to use the MacCormack method without implementing flux-corrected transport to limit this overshooting effect.

Vorticity L2 error in the solutions appeared to converge at a first-order rate for the Rusanov scheme, and a second-order rate for the MacCormack scheme. The central differencing scheme used to calculate vorticity has a truncation error of order Δx^2, which implies that the convergence of the MacCormack method vorticity L2 error is either matching or being limited by the convergence of this central differencing. Conversely, the Rusanov method must be the bottleneck in vorticity L2 error convergence, since it does not reach the second-order convergence rate expected for central differencing. Circulation error is near zero for all methods, but appears to behave differently for different cases. For some cases, circulation error converges very slowly, while for others it converges much faster than the second-order rate reference line. The exact cause of this is not known, but may involve the particularities of the trapezoidal integration scheme used to calculate circulation and enstrophy. The Rusanov method appears to consistently have lower circulation error, which may be a result of the high amount of dissipation in the scheme – the linear approximation made by the trapezoidal integration algorithm may more closely resemble the node-to-node changes in vorticity when gradients have been smoothed due to dissipation.

Appendix A

1

1) $\psi(x,y) = \rho_\infty u_\infty y - \rho_\infty V_\infty x + S\exp\left[-(r^*)^2/2\right]$

$$\frac{\partial \psi}{\partial y} = \rho u_\infty + \frac{\partial}{\partial y}\left(-\frac{(r^*)^2}{2}\right) S\exp\left[-(r^*)^2/2\right]$$

$$-\frac{(r^*)^2}{2} = -\frac{1}{2}\left(\frac{\sqrt{(x-x_c)^2+(y-y_c)^2}}{R_c}\right)^2 = -\frac{(x-x_c)^2+(y-y_c)^2}{2R_c^2}$$

$$\frac{\partial}{\partial y}\left(-\frac{(r^*)^2}{2}\right) = -\frac{2(y-y_c)(1)}{2R_c^2} = -\frac{(y-y_c)}{R_c^2}$$

$$u = \frac{1}{\rho_\infty}\frac{\partial \psi}{\partial y} = \frac{\rho_\infty u_\infty}{\rho_\infty} - \frac{S(y-y_c)}{\rho_\infty R_c^2}\exp\left[-(r^*)^2/2\right]$$

$$\boxed{u = u_\infty - \frac{S(y-y_c)}{\rho_\infty R_c^2}\exp\left[-\frac{(r^*)^2}{2}\right]} \quad (2a)$$

$$\frac{\partial \psi}{\partial x} = -\rho_\infty V_\infty + \frac{\partial}{\partial x}\left(-\frac{(r^*)^2}{2}\right) S\exp\left[-(r^*)^2/2\right]$$

$$\frac{\partial}{\partial x}\left(-\frac{(r^*)^2}{2}\right) = -\frac{2(x-x_c)(1)}{2R_c^2} = -\frac{(x-x_c)}{R_c^2}$$

$$v = \frac{-\rho_\infty V_\infty}{\rho_\infty} + \frac{S(x-x_c)}{\rho_\infty R_c^2}\exp\left[-(r^*)^2/2\right]$$

$$\boxed{V = V_\infty + \frac{S(x-x_c)}{\rho_\infty R_c^2}\exp\left[-(r^*)^2/2\right]} \quad (2b)$$

2) Assume that $u_\infty = 0$, $V_\infty = 0$

$$(M_a a)^2 = M_a^2 a^2 = V^2 = u^2 + v^2$$

$$u^2 = \frac{S^2(y-y_c)^2}{\rho_\infty^2 R_c^4}\exp\left[-(r^*)^2\right] \qquad v^2 = \frac{S^2(x-x_c)^2}{\rho_\infty^2 R_c^4}\exp\left[-(r^*)^2\right]$$

$$V^2 = \frac{S^2}{\rho_\infty^2 R_c^4}\exp\left[-(r^*)^2\right]\left\{(x-x_c)^2+(y-y_c)^2\right\}$$

→

at $r = R_c$, $r^* = 1$ $\quad 1 = \frac{\sqrt{(x-x_c)^2 + (y-y_c)^2}}{R_c}$

$R_c^2 = (x-x_c)^2 + (y-y_c)^2$ → substitute into eq.

$M_{a,c}^2 a_c^2 = \frac{S^2 R_c^2}{\rho_\infty^2 R_c^4} \exp(-1) = \frac{S^2}{\rho_\infty^2 R_c^2} \exp(-1)$

$\rho_\infty^2 R_c^2 M_{a,c}^2 a_c^2 \exp(1) = S^2$

$$\boxed{S = \rho_\infty R_c M_{a,c} a_c \exp\left(\tfrac{1}{2}\right)} \quad (3)$$

3) $u = u_\infty - \frac{\rho_\infty R_c M_{a,c} a_c (y-y_c)}{\rho_\infty R_c^2} \exp\left(\tfrac{1}{2}\right) \exp\left(-(r^*)^2/2\right)$

$u = u_\infty - M_{a,c} a_c \frac{(y-y_c)}{R_c} \exp\left[(1-(r^*)^2)/2\right]$

$$\boxed{u = u_\infty - M_{a,c} a_c y^* \exp\left[(1-(r^*)^2)/2\right]} \quad (4a)$$

$V = V_\infty + \frac{\rho_\infty R_c M_{a,c} a_c (x-x_c)}{\rho_\infty R_c^2} \exp\left(\tfrac{1}{2}\right) \exp\left(-(r^*)^2/2\right)$

$V = V_\infty + M_{a,c} a_c \frac{(x-x_c)}{R_c} \exp\left[(1-(r^*)^2)/2\right]$

$$\boxed{V = V_\infty + M_{a,c} a_c x^* \exp\left[(1-(r^*)^2)/2\right]} \quad (4b)$$

4) $\frac{T_0}{T} = 1 + \frac{\gamma-1}{2} M_a^2$ first, assume isentropic flow

$$M_a^2 = \frac{v^2}{a^2} = \frac{S^2}{a^2 \rho_\infty^2 R_c^4} \exp\left[-(r^*)^2\right]\left\{(x-x_c)^2 + (y-y_c)^2\right\}$$

from eq. 3, $S^2 = \rho_\infty^2 R_c^2 M_{a,c}^2 a_c^2 \exp(1)$

$$M_a^2 = \frac{\rho_\infty^2 R_c^2 M_{a,c}^2 a_c^2}{a^2 \rho_\infty^2 R_c^4} \exp(1) \exp\left[-(r^*)^2\right]\left\{(x-x_c)^2 + (y-y_c)^2\right\}$$

$$M_a^2 = \frac{M_{a,c}^2 a_c^2}{a^2 R_c^2} \exp\left[1-(r^*)^2\right]\left\{(x-x_c)^2 + (y-y_c)^2\right\}$$

$$= (r^*)^2$$

$$M_a^2 = M_{a,c}^2 \frac{a_c^2}{a^2} (r^*)^2 \exp\left[1-(r^*)^2\right]$$

substitute into isentropic relation

$$\frac{T_0}{T} = 1 + \frac{\gamma-1}{2} M_{a,c}^2 \frac{a_c^2}{a^2} (r^*)^2 \exp\left[1-(r^*)^2\right]$$

evaluate at $r = R_c$, where

$T = T_c$ and $r^* = \frac{r}{R_c} = 1$ and $a = a_c$

$$\frac{T_0}{T_c} = 1 + \frac{\gamma-1}{2} M_{a,c}^2 \frac{a_c^2}{a_c^2} (1)^2 \exp\left[1-(1)^2\right]$$

$$\boxed{T_c = \frac{T_0}{1 + \frac{\gamma-1}{2} M_{a,c}^2}} \quad (5)$$

5) start with previously derived relation:

$$\frac{T_0}{T} = 1 + \frac{\gamma-1}{2} M_{a,c}^2 \frac{a_c^2}{a^2} (r^*)^2 \exp\left[1-(r^*)^2\right] \longrightarrow$$

need to recast $\frac{a_c^2}{a^2}$ term in terms of $M_{a,c}$, might be able to do this via T and T_c:

$$a_c^2 = \left(\sqrt{\gamma R T_c}\right)^2 = \gamma R T_c$$

$$a^2 = \left(\sqrt{\gamma R T}\right)^2 = \gamma R T$$

$$\frac{a_c^2}{a^2} = \frac{\gamma R T_c}{\gamma R T} = \frac{T_c}{T} \rightarrow \text{substitute this in}$$

$$\frac{T_0}{T} = 1 + \frac{\gamma-1}{2} M_{a,c}^2 \frac{T_c}{T} (r^*)^2 \exp\left[1-(r^*)^2\right]$$

Try to rewrite $\frac{T_c}{T}$ in terms of known isentropic relations for this flow

$$\frac{T_c}{T} = \frac{T_0}{1+\frac{\gamma-1}{2}M_{a,c}^2} \cdot \frac{1}{T} = \frac{T_0}{T\left(1+\frac{\gamma-1}{2}M_{a,c}^2\right)}$$

$$\frac{T_0}{T} = 1 + \frac{\gamma-1}{2} M_{a,c}^2 \frac{T_0}{T} \frac{1}{\left(1+\frac{\gamma-1}{2}M_{a,c}^2\right)} (r^*)^2 \exp\left[1-(r^*)^2\right]$$

$$\frac{T_0}{T} = 1 + \frac{T_0}{T} \frac{\frac{\gamma-1}{2}M_{a,c}^2}{1+\frac{\gamma-1}{2}M_{a,c}^2} (r^*)^2 \exp\left[1-(r^*)^2\right]$$

$$1 = \frac{T_0}{T} - \frac{T_0}{T} \frac{\frac{\gamma-1}{2}M_{a,c}^2}{1+\frac{\gamma-1}{2}M_{a,c}^2} (r^*)^2 \exp\left[1-(r^*)^2\right]$$

$$1 = \frac{T_0}{T}\left(1 - \frac{\frac{\gamma-1}{2}M_{a,c}^2}{1+\frac{\gamma-1}{2}M_{a,c}^2} (r^*)^2 \exp\left[1-(r^*)^2\right]\right)$$

$$\boxed{\frac{T}{T_0} = 1 - \frac{\frac{\gamma-1}{2}M_{a,c}^2}{1+\frac{\gamma-1}{2}M_{a,c}^2} (r^*)^2 \exp\left[1-(r^*)^2\right]} \quad (6)$$

6) Analytical expression for vorticity:

$$\omega = \frac{\partial V}{\partial x} - \frac{\partial u}{\partial y}$$

$$V = V_\infty + M_{a,c} a_c x^* \exp\left(\frac{1}{2}\right) \exp\left(-(r^*)^2/2\right)$$

$$\frac{\partial V}{\partial x} = M_{a,c} a_c \frac{\partial x^*}{\partial x} \exp\left(\frac{1}{2}\right) \exp\left(-(r^*)^2/2\right)$$

$$+ M_{a,c} a_c x^* \exp\left(\frac{1}{2}\right) \frac{\partial}{\partial x}\left[-(r^*)^2/2\right] \exp\left(-(r^*)^2/2\right)$$

$$\rightarrow \frac{\partial x^*}{\partial x} = \frac{\partial}{\partial x}\left(\frac{x - x_c}{R_c}\right) = \frac{1}{R_c}$$

$$\rightarrow \frac{\partial}{\partial x}\left[-(r^*)^2/2\right] = \frac{\partial}{\partial x}\left[-\frac{(x-x_c)^2 + (y-y_c)^2}{2R_c^2}\right] = -\frac{2(x-x_c)(1)}{2R_c^2}$$

$$= -\frac{(x-x_c)}{R_c^2}$$

$$\therefore \frac{\partial V}{\partial x} = \frac{M_{a,c} a_c}{R_c} \exp\left(\frac{1}{2}\right) \exp\left(-(r^*)^2/2\right) - \frac{M_{a,c} a_c x^* (x - x_c)}{R_c^2} \exp\left(\frac{1}{2}\right) \exp\left(-(r^*)^2/2\right)$$

$$\frac{\partial V}{\partial x} = \frac{M_{a,c} a_c}{R_c} \exp\left[(1 - (r^*)^2)/2\right] \left\{1 - (x^*)^2\right\}$$

$$\rightarrow = \frac{x^*}{R_c}$$

$$\frac{\partial u}{\partial y} = -M_{a,c} a_c \frac{\partial y^*}{\partial y} \exp\left(\frac{1}{2}\right) \exp\left(-(r^*)^2/2\right)$$

$$- M_{a,c} a_c y^* \exp\left(\frac{1}{2}\right) \frac{\partial}{\partial y}\left[-(r^*)^2/2\right] \exp\left(-(r^*)^2/2\right)$$

$$\rightarrow \frac{\partial}{\partial y}\left(\frac{y - y_c}{R_c}\right) = \frac{1}{R_c}$$

$$\rightarrow \frac{\partial}{\partial y}\left[-(r^*)^2/2\right] = -\frac{2(y-y_c)(1)}{2R_c^2} = -\frac{(y-y_c)}{R_c^2} = -\frac{y^*}{R_c}$$

→

$$\therefore \quad \frac{\partial u}{\partial y} = -\frac{M_{ac} a_c}{R_c} \exp\left[(1-(r^*)^2)/2\right] + \frac{M_{ac} a_c (y^*)^2}{R_c} \exp\left[(1-(r^*)^2)/2\right]$$

$$\frac{\partial u}{\partial y} = -\frac{M_{ac} a_c}{R_c} \exp\left[(1-(r^*)^2)/2\right] \left\{1-(y^*)^2\right\}$$

$$\omega = \frac{\partial v}{\partial x} - \frac{\partial u}{\partial y} = \frac{M_{ac} a_c}{R_c} \exp(\xi) \left\{1-(x^*)^2\right\}$$

$$+ \frac{M_{ac} a_c}{R_c} \exp(\xi) \left\{1-(y^*)^2\right\}$$

$$= \frac{M_{ac} a_c}{R_c} \exp(\xi) \left\{\left[1-(x^*)^2 + 1-(y^*)^2\right]\right\}$$

$$\boxed{\omega = \frac{M_{ac} a_c}{R_c} \exp\left[(1-(r^*)^2)/2\right] \left\{2-(x^*)^2-(y^*)^2\right\}}$$

Analytical expression for vorticity ω

Appendix B

Initialization code

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

@author: az
"""
def init2Dvortex(N,case):
    import numpy as np

    # These parameters are the same for all cases, so define them here
    L = 1
    Rc = L/10
    xc = L/2
    yc = L/2
    T0 = 298
    p0 = 101300
    gamma = 1.4
    R = 287.058

    # Depending on which "case" parameter is supplied, we set different values for Mac, uinf,
vinf, and t_final
    if case == 'base':
        Mac = 0.3
        uinf = 0
        vinf = 0
        Tc = T0/(1+((gamma-1)/2)*Mac**2)
        ac = np.sqrt(gamma*R*Tc)
    if case == 'xconv':
        Mac = 0.3
        a0 = np.sqrt(gamma*R*T0)
        uinf = 0.3*a0
        vinf = 0
    if case == 'yconv':
        Mac = 0.3
        a0 = np.sqrt(gamma*R*T0)
        uinf = 0
        vinf = 0.3*a0
    if case == 'diagconv':
        Mac = 0.3
        a0 = np.sqrt(gamma*R*T0)
        uinf = 0.3*(np.sqrt(2)/2)*a0
        vinf = 0.3*(np.sqrt(2)/2)*a0
```

```
    if case == 'comp':
        Mac = 1.5
        Tc = T0/(1+((gamma-1)/2)*Mac**2)
        ac = np.sqrt(gamma*R*Tc)
        uinf = 0
        vinf = 0

    # First, establish some storage vectors
    u = np.zeros((N,N))
    u_init = np.zeros((N,N))
    v = np.zeros((N,N))
    v_init = 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 = T0/(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_init[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_init[j,i] = vinf+Mac*ac*xstar*rstarexp
            T[j,i] = T0*(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

    u[:,:] = u_init
    v[:,:] = v_init
    p = p0*(T/T0)**((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)

# Calculate shadowgraph metric and compressibility measure here
shadow = np.zeros((N,N))
comp = np.zeros((N,N))

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

        shadow[j,i] = (rho[j,ip]-2*rho[j,i]+rho[j,im])/(dx**2)-(rho[jp,i]-
2*rho[j,i]+rho[jm,i])/(dy**2)
        comp[j,i] = (u[j,ip]-u[j,im])/(2*dx)+(v[jp,i]-v[jm,i])/(2*dy)

  return x,y,u,v,T,p,rho,e,omega,omegaex,L2omega,circerr,shadow,comp
```

MacCormack method code

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

@author: az
"""
def maccormack2Dvortex(N,case):
    import numpy as np

    # These parameters are the same for all cases, so define them here
    L = 1
    Rc = L/10
    xc = L/2
    yc = L/2
    T0 = 298
    p0 = 101300
    gamma = 1.4
    R = 287.058

    # Depending on which "case" parameter is supplied, we set different values for Mac, uinf,
vinf, and t_final
    if case == 'base':
        Mac = 0.3
        uinf = 0
        vinf = 0
        Tc = T0/(1+((gamma-1)/2)*Mac**2)
        ac = np.sqrt(gamma*R*Tc)
        t_final = Rc/ac # Calculate final time
    if case == 'xconv':
        Mac = 0.3
        a0 = np.sqrt(gamma*R*T0)
        uinf = 0.3*a0
        vinf = 0
        t_final = L/uinf
    if case == 'yconv':
        Mac = 0.3
        a0 = np.sqrt(gamma*R*T0)
        uinf = 0
        vinf = 0.3*a0
        t_final = L/vinf
    if case == 'diagconv':
        Mac = 0.3
        a0 = np.sqrt(gamma*R*T0)
```

```
        uinf = 0.3*(np.sqrt(2)/2)*a0
        vinf = 0.3*(np.sqrt(2)/2)*a0
        t_final = (np.sqrt(2))*L/np.sqrt(uinf**2+vinf**2)
    if case == 'comp':
        Mac = 1.5
        Tc = T0/(1+((gamma-1)/2)*Mac**2)
        ac = np.sqrt(gamma*R*Tc)
        t_final = Rc/ac
        uinf = 0
        vinf = 0

    CFL = 0.5 # I have to set this very low for stability for some reason, this is what I need help
with I think

    # First, establish some storage vectors
    u = np.zeros((N,N))
    u_init = np.zeros((N,N))
    v = np.zeros((N,N))
    v_init = 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 = T0/(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_init[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_init[j,i] = vinf+Mac*ac*xstar*rstarexp
            T[j,i] = T0*(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

    u[:,:] = u_init
    v[:,:] = v_init
    p = p0*(T/T0)**((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

    # Establish some more storage vectors for Q, F, G, and their predictor values
    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))

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

    count = 0 # Set up a count variable and initial time
    t = 0
    # 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:
      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 = N-2
            if i == N-1:
               ip = 1
            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**2/2+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 = N-2
          if i == N-1:
             ip = 1
          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**2/2+v**2)/2
        p = e*(gamma-1)*rho # calculate values of rho, u, e, and p based on predictor Q
        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
        count = count+1

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

    # This loop calculates vorticity (note that this is done only for the final timestep using u and v,
so it is done outside of the time loop)
    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)

# Calculate shadowgraph metric and compressibility measure here
shadow = np.zeros((N,N))
comp = np.zeros((N,N))

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

        shadow[j,i] = (rho[j,ip]-2*rho[j,i]+rho[j,im])/(dx**2)-(rho[jp,i]-
2*rho[j,i]+rho[jm,i])/(dy**2)
        comp[j,i] = (u[j,ip]-u[j,im])/(2*dx)+(v[jp,i]-v[jm,i])/(2*dy)

    return x,y,u,v,T,p,rho,e,omega,L2omega,circerr,shadow,comp
```

Rusanov method code

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

@author: az
"""
def rusanov2Dvortex(N,case):
    import numpy as np

    # These parameters are the same for all cases, so define them here
    L = 1
    Rc = L/10
    xc = L/2
    yc = L/2
    T0 = 298
    p0 = 101300
    gamma = 1.4
    R = 287.058

    # Depending on which "case" parameter is supplied, we set different values for Mac, uinf,
vinf, and t_final
    if case == 'base':
        Mac = 0.3
        uinf = 0
```

```
    vinf = 0
    Tc = T0/(1+((gamma-1)/2)*Mac**2)
    ac = np.sqrt(gamma*R*Tc)
    t_final = Rc/ac # Calculate final time
if case == 'xconv':
    Mac = 0.3
    a0 = np.sqrt(gamma*R*T0)
    uinf = 0.3*a0
    vinf = 0
    t_final = L/uinf
if case == 'yconv':
    Mac = 0.3
    a0 = np.sqrt(gamma*R*T0)
    uinf = 0
    vinf = 0.3*a0
    t_final = L/vinf
if case == 'diagconv':
    Mac = 0.3
    a0 = np.sqrt(gamma*R*T0)
    uinf = 0.3*(np.sqrt(2)/2)*a0
    vinf = 0.3*(np.sqrt(2)/2)*a0
    t_final = (np.sqrt(2))*L/np.sqrt(uinf**2+vinf**2)
if case == 'comp':
    Mac = 1.5
    Tc = T0/(1+((gamma-1)/2)*Mac**2)
    ac = np.sqrt(gamma*R*Tc)
    t_final = Rc/ac
    uinf = 0
    vinf = 0

CFL = 0.5
# First, establish some storage vectors
u = np.zeros((N,N))
u_init = np.zeros((N,N))
v = np.zeros((N,N))
v_init = 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 = T0/(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_init[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_init[j,i] = vinf+Mac*ac*xstar*rstarexp
         T[j,i] = T0*(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

   u[:,:] = u_init
   v[:,:] = v_init
   p = p0*(T/T0)**((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

   # 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))

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

   # Set up initial time
   t = 0
```

```
   # 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:
      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 = N-2
            if i == N-1:
               ip = 1
            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 = N-2
        if i == N-1:
           ip = 1
        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**2/2+v**2)/2
    p = e*(gamma-1)*rho # calculate values of rho, u, e, and p based on predictor Q
    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

  # This loop calculates vorticity (note that this is done only for the final timestep using u and v,
so it is done outside of the time loop)
  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)
```

```
    # Calculate shadowgraph metric and compressibility measure here
    shadow = np.zeros((N,N))
    comp = np.zeros((N,N))

    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

            shadow[j,i] = (rho[j,ip]-2*rho[j,i]+rho[j,im])/(dx**2)-(rho[jp,i]-
2*rho[j,i]+rho[jm,i])/(dy**2)
            comp[j,i] = (u[j,ip]-u[j,im])/(2*dx)+(v[jp,i]-v[jm,i])/(2*dy)

    return x,y,u,v,T,p,rho,e,omega,L2omega,circerr,shadow,comp
```

Plotting code

```
# -*- coding: utf-8 -*-
"""
Created on Tue Apr  7 15:54:14 2020

@author: az
"""
from maccormack2Dvortex import maccormack2Dvortex
from rusanov2Dvortex import rusanov2Dvortex
from init2Dvortex import init2Dvortex
import matplotlib.pyplot as plt
import numpy as np
import string
```

```
case = 'base'
#case = 'xconv'
#case = 'yconv'
#case = 'diagconv'
#case = 'comp'

N = [25,50,100]

L2first_order = np.zeros((3))
L2second_order = np.zeros((3))
circfirst_order = np.zeros((3))
circsecond_order = np.zeros((3))

x0,y0,u0,v0,T0,p0,rho0,e0,omega0,omegaex0,L2omega0,circerr0,shadow0,comp0 =
init2Dvortex(N[2],case)
x1,y1,u1,v1,T1,p1,rho1,e1,omega1,L2omega1,circerr1,shadow1,comp1 =
maccormack2Dvortex(N[0],case)
x2,y2,u2,v2,T2,p2,rho2,e2,omega2,L2omega2,circerr2,shadow2,comp2 =
maccormack2Dvortex(N[1],case)
x3,y3,u3,v3,T3,p3,rho3,e3,omega3,L2omega3,circerr3,shadow3,comp3 =
maccormack2Dvortex(N[2],case)
x4,y4,u4,v4,T4,p4,rho4,e4,omega4,L2omega4,circerr4,shadow4,comp4 =
rusanov2Dvortex(N[0],case)
x5,y5,u5,v5,T5,p5,rho5,e5,omega5,L2omega5,circerr5,shadow5,comp5 =
rusanov2Dvortex(N[1],case)
x6,y6,u6,v6,T6,p6,rho6,e6,omega6,L2omega6,circerr6,shadow6,comp6 =
rusanov2Dvortex(N[2],case)

### Part 1
origin = 'lower'
fig1, ax1 = plt.subplots(2,1,figsize=(6.5,8),constrained_layout=True,sharex=True)
fig1.suptitle('Figure 4: Comparison of $\omega$ contours, baseline case, N = 100',fontsize=10,
weight='bold')
ax1[0].set_title('MacCormack method',fontsize=10)
CS1 = ax1[0].contour(x0, y0, omegaex0, 10, origin=origin ,linestyles='dashed')
CS2 = ax1[0].contour(x3, y3, omega3, 10 ,origin=origin)
ax1[0].set_ylabel('y (m)')
CS3 = ax1[1].contour(x0, y0, omegaex0, 10, origin=origin ,linestyles='dashed')
CS4 = ax1[1].contour(x6, y6, omega6, 10 ,origin=origin)
ax1[1].set_title('Rusanov method',fontsize=10)
ax1[0].clabel(CS1, CS1.levels, inline=True, fontsize=10)
ax1[0].clabel(CS2, CS2.levels, inline=True, fontsize=10)
```

```
ax1[1].clabel(CS3, CS3.levels, inline=True, fontsize=10)
ax1[1].clabel(CS4, CS4.levels, inline=True, fontsize=10)
ax1[1].set_xlabel('x (m)')
ax1[1].set_ylabel('y (m)')
h1,_ = CS1.legend_elements()
h2,_ = CS2.legend_elements()
h3,_ = CS3.legend_elements()
h4,_ = CS4.legend_elements()
ax1[0].legend([h1[0], h2[0]], ['Analytical', 'MacCormack'],loc = 'best')
ax1[1].legend([h3[0], h4[0]], ['Analytical', 'Rusanov'],loc = 'best')
for i, ax in enumerate(ax1):
   ax.text(1.025, 0.5, string.ascii_uppercase[i], transform=ax.transAxes,size=10, weight='bold')

for i in range(0,3):
   L2first_order[i] = L2omega4/(2**i)
   L2second_order[i] = L2omega4/(2**(2*i))
   circfirst_order[i] = circerr4/(2**i)
   circsecond_order[i] = circerr4/(2**(2*i))

fig2, ax2 = plt.subplots(1,2,figsize=(6.5,3.5),constrained_layout=True,sharey=True)
fig2.suptitle('Figure 5: Comparison of $u$ profiles at $x=x_c$, baseline case',fontsize=10,
weight='bold')
ax2[0].plot(y0, u0[:,50],"--k",label='Initial')
ax2[0].plot(y1, u1[:,12],label='N = 25')
ax2[0].plot(y2, u2[:,25],label='N = 50')
ax2[0].plot(y3, u3[:,50],label='N = 100')
ax2[1].plot(y0, u0[:,50],"--k",label='Initial')
ax2[1].plot(y4, u4[:,12],label='N = 25')
ax2[1].plot(y5, u5[:,25],label='N = 50')
ax2[1].plot(y6, u6[:,50],label='N = 100')
ax2[0].set_title('MacCormack method',fontsize=10)
ax2[0].set_xlabel('y (m)')
ax2[0].set_ylabel('$u$ (m/s)')
ax2[1].set_title('Rusanov method',fontsize=10)
ax2[1].set_xlabel('y (m)')
ax2[0].grid(True)
ax2[1].grid(True)
ax2[0].set_xlim([0, 1])
ax2[1].set_xlim([0, 1])
ax2[0].legend(loc='best')
for n, ax in enumerate(ax2):
   ax.text(0.5, -0.3, string.ascii_uppercase[n], transform=ax.transAxes,
         size=10, weight='bold')
```

```
fig3, ax3 = plt.subplots(1,2,figsize=(6.5,3.5),constrained_layout=True,sharey=True)
fig3.suptitle('Figure 6: Comparison of $\omega$ profiles at $x=x_c$, baseline case',fontsize=10,
weight='bold')
ax3[0].plot(y0, omegaex0[:,50],"--k",label='Initial')
ax3[0].plot(y1, omega1[:,12],label='N = 25')
ax3[0].plot(y2, omega2[:,25],label='N = 50')
ax3[0].plot(y3, omega3[:,50],label='N = 100')
ax3[1].plot(y0, omegaex0[:,50],"--k",label='Initial')
ax3[1].plot(y4, omega4[:,12],label='N = 25')
ax3[1].plot(y5, omega5[:,25],label='N = 50')
ax3[1].plot(y6, omega6[:,50],label='N = 100')
ax3[0].set_title('MacCormack method',fontsize=10)
ax3[0].set_xlabel('y (m)')
ax3[0].set_ylabel('$\omega$ (1/s)')
ax3[1].set_title('Rusanov method',fontsize=10)
ax3[1].set_xlabel('y (m)')
ax3[0].grid(True)
ax3[1].grid(True)
ax3[0].set_xlim([0, 1])
ax3[1].set_xlim([0, 1])
ax3[0].legend(loc='upper left')
for n, ax in enumerate(ax3):
   ax.text(0.5, -0.3, string.ascii_uppercase[n], transform=ax.transAxes,
         size=10, weight='bold')

fig4, ax4 = plt.subplots(2,1,figsize=(6.5,4),constrained_layout=True,sharex=True)
fig4.suptitle('Figure 7: Comparison of error convergence rates, baseline case',fontsize=10,
weight='bold')
ax4[0].loglog(N,[L2omega1, L2omega2, L2omega3],"s",label="MacCormack")
ax4[0].loglog(N,[L2omega4, L2omega5, L2omega6],"o",label="Rusanov")
ax4[0].loglog(N,L2first_order,"--k",label="$Δx$")
ax4[0].loglog(N,L2second_order,"-.k",label="$Δx^2$")
ax4[0].set_title('Vorticity L2 error',fontsize=10)
ax4[0].set_ylabel('$log(L2_{\omega})$')
ax4[1].loglog(N,np.abs([circerr1, circerr2, circerr3]),"s",label="MacCormack")
ax4[1].loglog(N,np.abs([circerr4, circerr5, circerr6]),"o",label="Rusanov")
ax4[1].loglog(N,np.abs(circfirst_order),"--k",label="$Δx$")
ax4[1].loglog(N,np.abs(circsecond_order),"-.k",label="$Δx^2$")
ax4[1].set_title('Circulation error',fontsize=10)
ax4[1].set_ylabel('$log(|e|)$')
ax4[1].set_xlabel('$log(N)$')
ax4[0].grid(True, which="both")
ax4[1].grid(True, which="both")
ax4[0].legend(loc='best',ncol=2)
```

```
for n, ax in enumerate(ax4):
    ax.text(1.025, 0.5, string.ascii_uppercase[n], transform=ax.transAxes,
            size=10, weight='bold')
```